DICTIONARY BASED TIME SERIES MODELLING

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

Doctor of Philosophy

BY SHALINI SHARMA

PhD18011

Under the Supervision of

Dr Angshul Majumdar

Associate Professor, IIIT-Delhi

COMPUTER SCIENCE AND ENGINEERING

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY - DELHI

NEW DELHI– 110020

SEPTEMBER, 2022

DICTIONARY BASED TIME SERIES MODELLING

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

Doctor of Philosophy

BY SHALINI SHARMA

PhD18011

Under the Supervision of

Dr Angshul Majumdar

Associate Professor, IIIT-Delhi

COMPUTER SCIENCE AND ENGINEERING

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY - DELHI

NEW DELHI– 110020

SEPTEMBER, 2022

Certificate

This is to certify that the thesis titled **Dictionary Based Time Series Modelling** being submitted by **Shalini Sharma** to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Dr. Angshul Majumdar

September, 2022

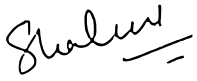
Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Declaration

This is to certify that the thesis titled **Dictionary Based Time Series Modelling** being submitted by **Shalini Sharma** to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.



Shalini Sharma

Indraprastha Institute of Information Technology Delhi

New Delhi 110020

Abstract

Time series analytics is the practice of determining future values of correlated signals. In seminal works, time series were modeled using classical techniques such as ARMA (autoregressive moving average), and its variants ARIMA (autoregressive integrated moving average). ARMA and its variant models often fail to perform well with non-stationary time series data. State Space Models (SSMs) have risen in the last few decades because they can overcome drawbacks of ARMA systems and provides an uncertainty quantification, which is crucial in time series point estimates and gives better-informed decisions. But SSMs are well suited for applications where the structure of the time series is known and understood in advance. They require the incorporation of structural and statistical information on the model. Real-time problems involve noisy samples with diverse sources; it can become difficult for SSMs to assume the model's structural details in advance.

In the last decade, various machine learning and deep learning techniques have also gained attention in solving time series forecasting problems. The structured neural network models, namely recurrent neural network (RNN) and long short-term memory (LSTM), 1D-CNN, DeepAR, TFT, N-Beats, MFNN are now considered as state-of-the-art in the resolution of stock forecasting problems, due to their inherent ability for processing varying length sequences and predicting future trends with no structural assumption in advance. It is worth mentioning that the deep learning models require a rather large dataset to learn parametric functions to forecast efficiently for unseen data. Moreover, those techniques usually provide pointwise estimates without any measure of uncertainty. Both approaches have their benefits but lack in one or other essential aspects to dynamically model the time-series signals.

This thesis has two main objectives: 1) Propose more efficient algorithms to forecast future time stamp signals dynamically, requiring no prior explicit information on model parameters and a less data-hungry approach. 2) Estimate

uncertainty score for each future pointwise prediction for a time-series signal. The work focuses on devising efficient implementation strategies for practical use of the method in the context of stock market time series analysis. The Thesis will walk you through the investigation conducted based on experiments on actual financial data of the performance of the novel proposed approaches.

The first contribution proposes new modeling and inferential tool for dynamical processing of time series. The approach is called recurrent dictionary learning (RDL). The proposed model reads as a linear Gaussian Markovian state-space model involving two linear operators, the state evolution and the observation matrices that we assumed unknown. These two unknown operators (that can be seen interpreted as dictionaries) and the sequence of hidden states are jointly learnt via an expectation-maximization algorithm. The RDL model gathers several advantages: online processing, probabilistic inference, and a high model expressiveness, which is usually typical of neural networks. RDL is particularly well suited for stock forecasting. Its performance is illustrated on two problems: next-day forecasting (regression problem) and next-day trading (classification problem), given past stock market observations.

Second, we propose sequential transform learning (STL). The proposed work is a linear Gaussian Markovian state-space model involving state evolution, observation matrices, and an exogenous control input. The resultant formulation resembles loosely based on transform learning. The proposed work is made recurrent, introducing a feedback loop where learnt transform coefficients for the t^{th} instant is fed back along with the $t+1^{st}$ sample. Furthermore, the formulation is made supervised by the label consistency cost. The approach differs from RDL due to presence of an exogenous input, which helps to establish more informed prior for state-space evolution. Our approach keeps the best of two worlds, marrying the interpretability and uncertainty measure of signal processing with the function approximation ability of neural networks. We have carried out experiments on one of the most challenging problems in dynamical modeling – stock forecasting.

Third, we propose Deep recurrent dictionary learning (DRDL). The proposed work is developed to cater to bottlenecks experienced in recurrent dictionary learning approach. The work overcomes the limitations of RDL and also dives into multi-linear Gaussian state space. In this work, we combine the benefits of both approaches(signal processing and neural network) by introducing a multi-linear Gaussian SSM whose state and evolution operators can be learnt from

the data. We propose factorized forms for the state and evolution operators to cope with possible non-linearity in the observed data and the hidden state. We also introduced expectation-maximization method combined with an alternating block strategy to estimate each involved factor while jointly performing the state inference, generalizing our previous work (Recurrent Dictionary Learning).

Fourth, we propose Deep sequential transform learning. The proposed method is a deep network to model multi-linear Gaussian state space in the presence of an exogenous input. In this work, we propose to model non-linearity using a deep factor model. Thus the proposed approach is a multi-linear Gaussian state space involving state evolution, observation matrices, and an exogenous control input modeled as a deep latent factor model. The model is also made recurrent by introducing a feedback loop where learnt deep factor model parameters for the t^{th} instant is fed back along with the $t + 1^{st}$ sample. The method is developed to overcome the limitations of the Sequential transform learning model and explore the multi-linear Gaussian state-space model in the presence of exogenous input, which differentiates it from the DRDL approach. The method keeps the best of both worlds – the interpretability and ability of SSMs to yield uncertainty estimates with the flexibility of RNNs to learn the underlying operators from the data. This work takes up one of the most challenging problems of today's age – predicting the prices of cryptocurrencies.

The motive of the work presented in this thesis is to propose efficient algorithms to forecast future time stamp signals dynamically, simultaneously estimating uncertainty score with each pointwise prediction to offer a more informed prediction for future time-series signals.

Dedication

Dedicated to my parents and grandparents, who have always believed in me. Thanks for being there as a mentor and a friend to bring out the best of my abilities. Special thanks to my husband "Anshul" for always being there for me in every phase of this journey, holding hands and encouraging me that **if I can believe, I can do!**.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dr. Angshul Majumdar for his constant support, patience, and motivation in my research journey so far. His guidance helped me throughout my Ph.D. and writing of this dissertation. I want to thank him for providing an outstanding research environment and facilities. I would also like to thank him for always being there as a friend to listen and guide me through challenging times. No matter how rough life is, Dr. Angshul Majumdar will always encourage you to see the bright side of it and come up with solutions to solve them. I feel one of the blessed person to have him as my Supervisor, Mentor, and Friend.
I would like to thank my committee members Dr Pushpendra Singh and Dr Sanat Biswas and collaborators Dr. Emilie Chouzenoux and Dr. Victor Elvira. for their insightful comments. I am grateful to the Indraprastha Institute of Information Technology for providing an excellent infrastructure.

I want to thank my Grandparents, Late K.P Sharma and Mrs. Dhana Devi, for continuously showering blessings. I am thankful to my parents (Mr.Narendra Prasad Sharma and Mrs. Anita Sharma) for working hard to provide me with the privilege of having a good life and attaining this prestigious degree. I especially thank my husband, Anshul, for his care, patience, encouragement, and unwavering support throughout this journey. I am also grateful to my siblings Kirti, Mayank for being supportive throughout the journey. I thank my labmates, Vanika Singhal, Megha Gupta Gaur, Jyoti Maggu, Aanchal Mongia, Shikha Singh, Pooja Gupta, and Anurag Goel, for their companionship and never-ending tea-time stories. I gratefully acknowledge their help and support. I want to present my sincere gratitude to my friends Neetesh Pandey, Shreya Mishra, Akshi Kathait, and Littisha Lawrence for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the past few decades, modeling the time series signals has remained challenging for researchers. Numerous studies [1,2] reveal there is a typical repeating pattern that are observed in data collected sequentially over time. These patterns can be explored and analyzed to forecast the future behavior of time-series signals. Time series is based on an elementary presumption that correlation between the sequential points in time can be modeled in terms of the dependence of current values on past values. Time series analysis helps us to model the stochastic behavior to predict or forecast the future series based on the historical data and other related factors.

Stock Forecasting is one of the challenging applications in the time-series domain. Researchers/Investors observe weekly interest rates, daily closing stock prices, monthly price indices, yearly sales figures, and so forth. Economists forecast the future closing price, percentage change in prices, and common repeating patterns in certain quarters based on the observed historical prices.

The stock market has always been challenging due to its very volatile nature and a high dependence on various macro and microeconomic factors like social, political, psychological, and economic variables [3]. Due to its high applicative impact, stock forecasting has been deeply investigated for several decades [4–8]. This thesis will walk you through insights into the bottlenecks in forecasting time series signals. The thesis will help readers understand the importance of forecasting time series signals in the domain of stock market analysis. The potentials and limitations of existing methods are discussed in detail, describing the need for more novel approaches to forecast and model time series given the limited time frame to forecast unseen future behaviors.

1.1 Problem Statement

The research problem addressed in this thesis is concerned with the task of predicting the future behaviour of time series signal dynamically. We also focus on providing the uncertainty score for each point wise prediction. The novel approaches proposed for predicting the time series are applied on very challenging application in real time ie., stock market. The aim is to offer more informed decision to the practitioner/investor in taking risk with stock market.

1.2 Related Work

1.2.1 From ARMA models to state-space models

Classical statistical time series modeling and analysis mostly assumed stationary stochastic processes. An important class of models and related methods is the autoregressive moving average (ARMA) models, which have been deeply studied in the literature and widely applied in countless relevant applications. A survey of these conventional techniques in stock forecasting can be found in [9]. Despite their wide applicability, ARMA-based methods present several disadvantages. First, it is well known that such techniques have a linear nature and they imposed additivity in the residuals, which limits their flexibility to model complex non-linear non-additive processes. ARMA models moreover often fail to perform well with non-stationary time series data. The *autoregressive integrated moving average* (ARIMA) are a wider class of models, developed to overcome some of the aforementioned limitations of ARMA models. There are a few studies that use Box-Jenkins techniques for stock forecasting [10–13]. Box-Jenkins method deploy ARMA and ARIMA model in a diagnostic manner. The iterative process involves identification, estimation, and diagnostic check to evaluate the fitted models for the available data. This approach evaluates which parts of the time series require can be improved. Box-Jenkins methods help to control the overfitting, which is essential for improving the modeling task.

However, ARIMA models still present some limitations, e.g., when dealing with the volatility clustering or fat tails in the noise distributions [14]. Moreover,

the ARIMA models can behave unstably in misspecified models. The main disadvantage of such a Box-Jenkins class of techniques was that it could only handle smooth variations in the data [15, 16]. To overcome the limitations of ARIMA models, regressors were introduced along with current data. These class of models was named as ARIMAX. ARIMAX models often improve the forecasting performance [17]. The main limitation of ARIMAX is the assumed linear relationship between the predictor and the target variable. The model cannot deal with multi-colinearity in the data. Moreover, the inclusion of more regressors for the forecasting task (for the variable of interest and other time-series information) often results in a model overfitting. Finally, a generic limitation of all previous models is the lack of latent variables, which clearly limits the expressiveness of the resulting models.

In the last decades, ARIMA models and its successive variants have been replaced with *state-space models* (SSMs) in many challenging applications, especially for modeling complex spatial-temporal processes. SSMs describe the evolution of a hidden state that evolves over discrete time steps (in principle, in a Markovian manner, but this can be easily extended) [18]. At each time step, an observation related to the given state is received. When the SSMs are linear and with AWGN, the Kalman filter provides the sequence of posterior (filtering) distributions of the hidden state given the previous data (plus some other distribution of interests, such as predictive distributions of different quantities) [19]. The Kalman filter allows for the computation of those pdfs, which are Gaussian, in a sequential (hence efficient) manner, processing each observation at each

time step, without the need of reprocessing past observations [19, 20]. Based on the Kalman algorithm, one can compute the sequence of smoothing distributions (pdf of a hidden state given all observations), which are also Gaussian in the case of the linear-Gaussian model, although in this case, the algorithm requires re-processing all observations. Both algorithms are described in the next section, and more details can be found in [18]. There exist some works considering the linear-Gaussian SSM for stock forecasting [21–23].Note that an important limitation of the linear-Gaussian model is the need to know not only the noise covariances but also the linear operators of the model. In real world applications, this is rarely the case, which prevents a wider application of this useful model [18, Chapter 12].

When the SSM is non-linear (but the noise is still Gaussian), approximations are needed. For instance, the extended Kalman filter [24, 25] or unscented Kalman filter [26] have been proposed to perform the inference tasks in such generic models. The difference between the EKF and the UKF is that, the former approximates about only the mean while the latter approximates about several points including the mean. The Unscented Kalman Filter works on the concept of Sigma Points. Some points are taken on source Gaussian and mapped on target Gaussian after passing points through some non linear function and then the new mean and variance of transformed Gaussian is calculated. This is why UKF is supposed to improve the results over EKF. The detailed explanation and mathematical formulation can be referred here [27]. However, these Kalman-based extensions still present several limitations both in the performance but also

in the range of SSMs where they can be used. Arguably, the most comprehensive tools for inferring the posterior distributions of interests in generic SSMs are the family of particle filters (PFs) [28–33]. PFs have shown outstanding performance where the models have high degree of uncertainty. This Monte Carlo technique can deal with virtually any SSM, including non-linearities and non-Gaussian non-additive noises, often at the expense of increasing the computational complexity. Recent works on PF allow to develop more efficient filters that require fewer particles [34, 35]. Other methods allow for an online adaptation on the number of particles in an online manner [36, 37]. It is worth noting that while more sophisticated models (than the linear-Gaussian one) are useful for advanced applications, this is at the expense of complicating the inference process, including the loss of strong theoretical guarantees, the introduction of approximate errors, and the increased computational complexity. This trade-off suggests that staying within the Kalman framework while broadening the model capabilities is a research direction of high interest. Note that all these techniques provide explicit uncertainty quantification in the estimates.

The modeling and prediction of time series have been applied to stock forecasting in the last five decades. However, there are relatively few references on the topic (compared to its wide application), arguably due to the secrecy practices associated with this competitive field. A comprehensive work on the application of Kalman filtering techniques for the said problem can be found in [38].

Despite the simplicity and versatility of the Kalman model and filter, it can

also fail in many applications due to the linear-Gaussian restriction but also due to the need of knowing the model at each time step. For instance, it is required to know the linear operator that maps from the state at a given time step, to the next time step. The Kalman model does not necessarily impose that this operator (or the operator that maps from the state to the observation at a given time) must remain fixed over time, but the knowledge of different operators at different times is then even more challenging. Knowledge about the model and its estimation procedure is key for the effectiveness of the Kalman filter in predicting future sequences. The above knowledge also helps for providing interpretability to the model. Some works are devoted to estimate some particular parameters, but dealing with model uncertainty in such context is considered to be a tough problem [39–42]. This thesis focuses on developing methods capable of learning time-varying states and observation models with the goal of improving the prediction accuracy of Kalman filters applied to Stock prediction problems.

1.2.2 Machine learning-based techniques

Several machine learning approaches, including neural network solutions, have been proposed to address the problem of financial data prediction. A subset of methods use temporal data over a given period as the input to a network for prediction of data at later time points [7, 43–50]. These aforementioned techniques thus work in a windowed fashion, by processing (possibly overlapped) windows of the time series only modeling static data and fails to incorporate

more future time series in window, hence lacks dynamical modelling behavior. In particular, support vector machines [8,51], MLP [52], random forests [53] and deep learning [54–57] have been successfully used for stock forecasting. The mentioned techniques rely on neural architectures which becomes complex with the introduction of deep layers. These models process huge amount of data and learn complex and highly nonlinear mapping functions from the data in a black box manner. This is limitation to the generalizability of the models and can lead to serious instability issues (e.g., in case of adversarial attacks). Moreover, the learning strategy, requiring to solve non-convex stochastic optimization problem, requires tedious parameter tuning and can be computationally expensive.

Recent studies also rely on a secondary source for stock forecasting, which is the textual information from either social networks or blogs [45–47]. Strictly speaking these are not artificial intelligence techniques for stock forecasting; they depend on human intelligence from blogs and posts. These techniques use natural language processing to extract relevant information from already existing secondary sources and predict based on the sentiment values associated with those sources.

Models based on RNN such as [58–60], are now considered to be high-performance state-of-the-art techniques to forecast time series due to their inherent property to memorize long sequences. RNNs can model the dynamical system continuously without the need for windowing and are thus naturally suitable for financial time-series analysis. However RNNs suffer from the vanishing gradient problem [61]. LSTM [62–64] is one such variant of RNNs, which is

popular due to its memory mimicking model that avoids the vanishing gradient problem in RNN. Another derivative of RNN that does not suffer from its vanishing gradient problem is GRU [65–68]. Let us note that 1D CNN performs better over LSTM and RNN, as they are highly noise resistant models and have the potential to extract highly informative and deep features which are independent from time [69–71]. However, 1D CNNs are not suitable for continuous data and for working in a sliding window fashion. The major advantage of neural networks over SSM is their great function approximation ability. Nonetheless, unlike SSM-based models, neural networks provide point estimates. They cannot quantify the uncertainty about the point estimate. This is a crucial shortcoming in applications such as stock trading.

1.3 Dataset Description

1.3.1 Stock Market

For Chapter2 to Chapter4, we consider dataset obtained from Yahoo finance repository [1] comprising of daily stock prices on a period of about 20 years (between 01/01/1998 to 01/10/2019), for 185 stocks, and gathering stocks from developed markets (USA, UK) and developing markets (India and China). The financial markets of the USA and UK are considered the most advanced markets in the world, while the financial markets of India and China are regarded as new emerging markets. Such a mix of stocks sources is interesting, as the

[1] https://yahoo.finance.com

investors are always curious to invest in a blend of advanced markets (i.e., developed markets) and emerging markets (i.e., developing markets) [72]. The investments in developed markets promise some fixed good returns, whereas investments in developing markets hold the potential of higher growth and higher risks. Sampling the stock market with varying market scenarios such as keeping a mix of advanced markets (top performers), and developing market (mid performers), also aims at limiting the issue of occurrence of bias in the compared computational models, while testing their robustness to different trends. The diversification helps investors and researchers to explore the market and observe model behaviour at critical points with highly non-stationary data [73].

1.3.2 Cryptocurrency Market

In Chapter5, we consider a cryptocurrency dataset comprising of ten cryptocurrencies. The dataset is extracted from the Yahoo finance cryptocurrency repository using Yahoo finance API [2]. It consists of active cryptocurrencies ranging from some old and new cryptocurrencies. The data extracted is about eight years (between 01/01/2014 to 01/06/2021) for Litecoin, Namecoin, Dogecoin, Peercoin, Bitcoin, Ripple, NXT. For Gridcoin and Ethereum, we extracted seven years of data (between 01/01/2015 to 01/06/2021), the time of its first release year. The dataset created is divided into train and test datasets. Training data is from the year 2014 to 2018(2017 December). In contrast, testing data is from 2018 to 2021 is used.

[2] https://finance.yahoo.com/cryptocurrencies

1.4 Research Contributions

Our contributions towards these objectives are as follows:
The first contribution proposes new modeling and inferential tool for dynamical processing of time series. The approach is called recurrent dictionary learning (RDL). The proposed model reads as a linear Gaussian Markovian state-space model involving two linear operators, the state evolution and the observation matrices that we assumed unknown. These two unknown operators (that can be seen interpreted as dictionaries) and the sequence of hidden states are jointly learnt via an expectation-maximization algorithm. The RDL model gathers several advantages: online processing, probabilistic inference, and a high model expressiveness, which is usually typical of neural networks. RDL is particularly well suited for stock forecasting. Its performance is illustrated on two problems: next-day forecasting (regression problem) and next-day trading (classification problem), given past stock market observations.

Second, we propose sequential transform learning (STL). The proposed work is a linear Gaussian Markovian state-space model involving state evolution, observation matrices, and an exogenous control input. The resultant formulation resembles loosely based on transform learning. The proposed work is made recurrent, introducing a feedback loop where learnt transform coefficients for the t^{th} instant is fed back along with the $t+1^{st}$ sample. Furthermore, the formulation is made supervised by the label consistency cost. The approach differs from RDL due to presence of an exogenous input, which helps to establish more informed

prior for state-space evolution. Our approach keeps the best of two worlds, marrying the interpretability and uncertainty measure of signal processing with the function approximation ability of neural networks. We have carried out experiments on one of the most challenging problems in dynamical modeling – stock forecasting.

Third, we propose Deep recurrent dictionary learning (DRDL). The proposed work is developed to cater to bottlenecks experienced in recurrent dictionary learning approach. The work overcomes the limitations of RDL and also dives into multi-linear Gaussian state space. In this work, we combine the benefits of both approaches(signal processing and neural network) by introducing a multi-linear Gaussian SSM whose state and evolution operators can be learnt from the data. We propose factorized forms for the state and evolution operators to cope with possible non-linearity in the observed data and the hidden state. We also introduced expectation-maximization method combined with an alternating block strategy to estimate each involved factor while jointly performing the state inference, generalizing our previous work (Recurrent Dictionary Learning).

Fourth, we propose Deep sequential transform learning. The proposed method is a deep network to model multi-linear Gaussian state space in the presence of an exogenous input. In this work, we propose to model non-linearity using a deep factor model. Thus the proposed approach is a multi-linear Gaussian state space involving state evolution, observation matrices, and an exogenous control input modeled as a deep latent factor model. The model is also made recurrent by introducing a feedback loop where learnt deep factor model parameters for the

t^{th} instant is fed back along with the $t + 1^{st}$ sample. The method is developed to overcome the limitations of the Sequential transform learning model and explore the multi-linear Gaussian state-space model in the presence of exogenous input, which differentiates it from the DRDL approach. The method keeps the best of both worlds – the interpretability and ability of SSMs to yield uncertainty estimates with the flexibility of RNNs to learn the underlying operators from the data. This work takes up one of the most challenging problems of today's age – predicting the prices of cryptocurrencies.

Chapter 2

Recurrent dictionary learning for state-space models with an application in stock forecasting

This chapter presents the concept of Recurrent dictionary learning for state space models. Recurrent dictionary learning (RDL) is applied to forecast and trade stock market. Stock forecasting is a practice to determine the future net value of a firm or any other asset whose value changes with time. However, analyzing the stock market and following directional price trends is challenging. The stock market is highly chaotic and lucrative because of which it has always been a challenging problem for researchers.

In this work, we propose to bridge the gap between SSMs and machine learning strategies, to propose a novel approach for multivariate time series analysis in stock price forecasting and trading, called Recurrent Dictionary Learning (RDL). The RDL model relies on a linear SSM-based method combined

with a dictionary learning step [74]. Following the paradigm of RNN, the model feeds back the representation (output) of the previous instant along with the current input to generate the representation for the current instant. The dictionary learning step allows to learn the model parameters (here, observation and state matrices) from the data itself, instead of imposing explicit values for them. Furthermore, due to the SSM framework, the method is able to provide uncertainty measures on the estimates, whose interest will be illustrated in the context of trading. The proposed RDL algorithm proceeds in two steps, by following an expectation-maximisation strategy. In the expectation step, the estimate and the associated uncertainty are computed, while in the maximization step, those quantities are used to update the dictionaries.

2.1 Proposed method

In SSM-based approaches mentioned earlier in section 1.2, we need to make structural assumptions of the observation and hidden static models, before performing the inference. This is unrealistic in stock market modeling due to the highly volatile data. On the other hand, neural network models are capable of modeling dynamic behavior with few assumptions but they rarely provide uncertainty measure associated with their output. This work aims to propose a novel *Recurrent Dictionary Learning* (RDL) method to predict the behavior of stock market data, by combining advantages from both SSM and machine learning words. The approach of neural network architectures comes from *dictionary*

learning (DL) [75–77]. DL amounts to extracting features from the data using a linear decomposition, typically sparse, onto a dictionary that is to be learnt on a training set. Dictionary learning has been so far used for solving static problems [78–80]. In this work, we present a dictionary learning framework for dynamical models, engineering it to work as a recurrent network. In this section, we present our general model for multivariate time-series prediction along with an associated inference technique relying on the expectation-minimization (EM) approach. We will later particularize our method to the problems of stock forecasting and trading, and we propose a windowing strategy allowing for on-the-fly prediction and decision making.

2.1.1 Recurrent Dictionary Model

Let us consider $(\mathbf{x}_k)_{1\leq k\leq K}$ the observed time-series of vectors of length N_x and $(\mathbf{z}_k)_{1\leq k\leq K}$ is the sequence of vectors of size N_z that we want to infer/estimate, $(\mathbf{v}_k)_{1\leq k\leq K}$ models the noise. Note that we consider here possibly multivariate signals and feature vectors, i.e., N_x and N_z can be greater than 1. Our RDL model is expressed as follows (see Fig. 2.1 for a schematic illustration in the noise-free case).

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= \mathbf{D}_1\mathbf{z}_{k-1} + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{D}_2\mathbf{z}_k + \mathbf{v}_{2,k}. \end{cases} \tag{2.1}$$

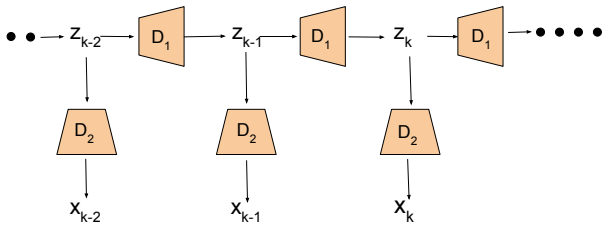


Figure 2.1: Schematic diagram of RDL model (2.1) (assuming zero noise).

We assume that the process noise $(\mathbf{v}_{1,k})_{1\leq k\leq K}$ is zero-mean Gaussian with covariance matrix $\mathbf{Q}$, and that the observation noise $(\mathbf{v}_{2,k})_{1\leq k\leq K}$ is zero-mean Gaussian with covariance matrix $\mathbf{R}$. With this, Eq.(2.1) represents a First order Markovian linear-Gaussian model where $(\mathbf{z}_k)_{1\leq k\leq K}$ is the sequence of K unknown states. The goal is the joint inference, from the observed sequence $(\mathbf{x}_k)_{1\leq k\leq K}$, of the dictionaries $\mathbf{D}_1 \in \mathbb{R}^{N_z\times N_z}$ and $\mathbf{D}_2 \in \mathbb{R}^{N_x\times N_z}$, and of the predicted sequence $(\mathbf{z}_k)_{1\leq k\leq K}$.

2.1.2 RDL inference algorithm

Under Gaussianity assumptions, the inference problem can be viewed as a blind Kalman filtering problem where both the predicted sequence and some model parameters (the observation and state linear operators) must be inferred/estimated from the data. Parametric estimation in state- space models has been considered in the literature through three main types of methods: EM algorithms, optimization based methods [81], and Monte Carlo methods [82]. We choose to apply the EM-based framework since it is well-suited for linear Gaussian models and it is flexible while benefiting from sounded convergence guarantees. EM is a method to iteratively find a maximum likelihood (ML) estimate of the parameters

when the direct optimization of the posterior distribution is not feasible. It is also highly connected to majorization- minimization (MM) approaches, as it can be viewed as a special class for MM approach. The EM algorithm alternates between performing an expectation (E-step), which provides the expectation of the conditional log-likelihood evaluated using the current estimate for the parameters, and a maximization (M-step), which computes parameters maximizing the expected log-likelihood found on the E-step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E-step.The iterative process goes on until convergence. We propose to solve the problem in an alternating manner, following the expectation-minimization scheme introduced in [18, chap.12] (see also [2]). We alternate iteratively between (i) the estimation of the state, the dictionaries $\mathbf{D}_1$ and $\mathbf{D}_2$ being fixed and (ii) the update of the dictionaries, assuming fixed state.

2.1.2.1 State update

At this step we considered the dictionaries $\mathbf{D}_1$ and $\mathbf{D}_2$ to be fixed and known, and the goal is the inference of the hidden state. Assume that the initial state follows $\mathbf{z}_0 \sim \mathcal{N}(\bar{\mathbf{z}}_0, \mathbf{P}_0)$, where $\mathbf{P}_0$ is a symmetric definite positive matrix of $\mathbb{R}^{N_z \times N_z}$ and $\bar{\mathbf{z}}_0 \in \mathbb{R}^{N_z}$. Then, (2.1) reads as a first-order Markovian linear-Gaussian model where $(\mathbf{z}_k)_{1 \leq k \leq K}$ is the sequence of K unknown states. The Kalman filter provides a probabilistic estimate of the hidden state at each time step k,

conditioned to all available data up to time k, through the filtering distribution:

$$p(\mathbf{z}_k|\mathbf{x}_{1:k}) = \mathcal{N}(\mathbf{z}_k; \bar{\mathbf{z}}_k, \mathbf{P}_k). \tag{2.2}$$

A filtering distribution allows to estimate a probability density function of states over time using observations. Readers can find the explanation about the filtering distribution in [chapter 4] [83]. For specific detail, please see Theorem 4.2 in cited reference.

For every $k \in \{1, \ldots, K\}$, the mean $\bar{\mathbf{z}}_k$ and covariance matrix $\mathbf{P}_k$ can be computed by means of the Kalman filter recursions given as follows. For $k = 1, \ldots, K$

Predict state:

$$\begin{cases} \mathbf{z}_k^- &= \mathbf{D}_1 \bar{\mathbf{z}}_{k-1}, \\ \mathbf{P}_k^- &= \mathbf{D}_1 \mathbf{P}_{k-1} \mathbf{D}_1^\top + \mathbf{Q} \end{cases} \tag{2.3}$$

Update state:

$$\begin{cases} \mathbf{y}_k &= \mathbf{x}_k - \mathbf{D}_2 \mathbf{z}_k^-, \\ \mathbf{S}_k &= \mathbf{D}_2 \mathbf{P}_k^- \mathbf{D}_2^\top + \mathbf{R}, \\ \mathbf{K}_k &= \mathbf{P}_k^- \mathbf{D}_2^\top \mathbf{S}_k^{-1}, \\ \bar{\mathbf{z}}_k &= \mathbf{z}_k^- + \mathbf{K}_k \mathbf{y}_k, \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top \end{cases} \tag{2.4}$$

The detailed derivation of the predict state and update state can be referred here [2, 83–85]. Smoothing for a Bayesian network means infering the distribution of a node conditioned on future information [85] The smoother allows one to refine estimates of previous states, in the light of later observations. The goal in

smoothing is to compute $p(z_k|x_{1:K})$ where K $\geq$ k. This is a non-causal process in that we use future observations to estimate the present state. The Rauch-Tung-Striebel (RTS) smoother makes a backward recursion on the data which makes use of the filtering distributions computed by the Kalman filter in order to obtain the smoothing distribution $p(\mathbf{z}_k|\mathbf{x}_{1...K})$. In the following we summarize the RTS recursions:

For $k = K, \ldots, 1$

Backward Recursion (Bayesian Smoothing):

$$\begin{cases} \mathbf{z}_{k+1}^{-} & = \mathbf{D}_1 \bar{\mathbf{z}}_k, \\ \mathbf{P}_{k+1}^{-} & = \mathbf{D}_1 \mathbf{P}_k \mathbf{D}_1^{\top} + \mathbf{Q}, \\ \mathbf{G}_k & = \mathbf{P}_k \mathbf{D}_1^{\top} [\mathbf{P}_{k+1}^{-}]^{-1}, \\ \mathbf{z}_k^s & = \bar{\mathbf{z}}_k + \mathbf{G}_k [\mathbf{z}_{k+1}^s - \mathbf{z}_{k+1}^{-}], \\ \mathbf{P}_k^s & = \mathbf{P}_k + \mathbf{G}_k [\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}] \mathbf{G}_k^{\top}. \end{cases} \tag{2.5}$$

As a result, the smoothing distribution at each time k has a Gaussian closed-form solution given by

$$p(\mathbf{z}_k|\mathbf{x}_{1...K}) = \mathcal{N}(\mathbf{z}_k; \mathbf{z}_k^s, \mathbf{P}_k^s). \tag{2.6}$$

The detailed derivation and motivation for above equation can be found [83, 86]. We recommend to understand the critical details from the references to understand the work better.

2.1.2.2 Dictionary Update

In this step, we derive the update of $\mathbf{D}_1$ and $\mathbf{D}_2$ given the sequence of states estimated in previous step. In order to estimate the parameters in state space, the joint likelihood can be introduced as the following function, more detailed derivation can be found in [section 2] [2](Please refer [chapter 12] [83]):

$$\begin{aligned}
&\mathcal{Q}(\mathbf{D}_1, \mathbf{D}_2) \\
&= \frac{1}{2}\log|2\pi\mathbf{P}_0| + \frac{K}{2}\log|2\pi\mathbf{Q}| + \frac{K}{2}\log|2\pi\mathbf{R}| \\
&+ \frac{1}{2}\text{tr}\left(\mathbf{P}_0^{-1}(\mathbf{P}_0^s + (\mathbf{z}_0^s - \bar{\mathbf{z}}_0)(\mathbf{z}_0^s - \bar{\mathbf{z}}_0)^\top)\right) \\
&+ \frac{K}{2}\text{tr}\left(\mathbf{Q}^{-1}(\mathbf{\Sigma} - \mathbf{C}\mathbf{D}_1^\top - \mathbf{D}_1\mathbf{C}^\top + \mathbf{D}_1\mathbf{\Phi}\mathbf{D}_1^\top)\right) \\
&+ \frac{K}{2}\text{tr}\left(\mathbf{R}^{-1}(\mathbf{\Delta} - \mathbf{B}\mathbf{D}_2^\top - \mathbf{D}_2\mathbf{B}^\top + \mathbf{D}_2\mathbf{\Sigma}\mathbf{D}_2^\top)\right)
\end{aligned} \tag{2.7}$$

where $(\mathbf{\Sigma}, \mathbf{\Phi}, \mathbf{B}, \mathbf{C})$ are defined from the outputs of the previously described RTS recursion:

$$\begin{cases}
\mathbf{\Sigma} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_k^s + \mathbf{z}_k^s(z_k^s)^\top, \\
\mathbf{\Phi} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_{k-1}^s + \mathbf{z}_{k-1}^s(\mathbf{z}_{k-1}^s)^\top, \\
\mathbf{B} &= \frac{1}{K}\sum_{k=1}^{K}x_k(\mathbf{z}_k^s)^\top, \\
\mathbf{C} &= \frac{1}{K}\sum_{k=1}^{K}(\mathbf{P}_k^s\mathbf{G}_{k-1}^\top + \mathbf{z}_k^s(\mathbf{z}_{k-1}^s)^\top).
\end{cases} \tag{2.8}$$

Then, we can show that the above function is a lower bound of the marginal log-likelihood $\varphi(\mathbf{D}_1, \mathbf{D}_2) = \log p(\mathbf{x}_{1:K}|\mathbf{D}_1, \mathbf{D}_2)$, i.e.,:

$$\log p(\mathbf{x}_{1:K}|\mathbf{D}_1, \mathbf{D}_2) \geq \mathbf{Q}(\mathbf{D}_1, \mathbf{D}_2) \quad (\forall(\mathbf{D}_1, \mathbf{D}_2)) \tag{2.9}$$

The update for $\mathbf{D}_1$ and $\mathbf{D}_2$ amounts for maximizing this lower bound, so as to increase value the marginal log-likelihood. Due to the quadratic form of function $\mathcal{Q}(\mathbf{D}_1, \mathbf{D}_2)$, the maximization step takes a closed form, leading to the dictionaries updates:

$$\begin{cases} \mathbf{D}_1^+ &= \mathbf{C}\mathbf{\Phi}^{-1}, \\ \mathbf{D}_2^+ &= \mathbf{B}\mathbf{\Sigma}^{-1}. \end{cases} \tag{2.10}$$

The RDL inference algorithm finally reads as the alternation of the Kalman filter/smoother, with fixed parameters $(\mathbf{D}_1, \mathbf{D}_2)$ (E-step) and the update of the dictionaries (M-step) [18][Alg.12.5]. The model is explained in detail in Algorithm 1. The advantage of the proposed method, as compared to alternating minimization strategies more commonly used in dictionary learning literature [87], is that it allows to make a probabilistic inference of the states and thus accounts explicitly on the uncertainty one may have on the estimates [88], while benefiting from the assessed monotonic convergence guarantees provided by the EM framework [89], and more generally from the majorization-minimization principle [90].

2.1.3 Application to stock forecasting and trading

Let us now focus on the stock market problem, particularly with the aim of performing forecasting and trading tasks from daily observations. We describe in this section how to adapt the RDL model and implementation to map with the specific characteristics of the considered application.

Algorithm* 1. *RDL inference algorithm.

Inputs. *Prior parameters* $(\overline{\mathbf{z}}_0, \mathbf{P}_0)$ *; model noise covariance matrices* $\mathbf{Q}$, $\mathbf{R}$ *; set of observations* $\{\mathbf{x}_k\}_{1\leq k\leq K}$.

Initialization. *Set positive latent factors*

$\{\mathbf{D}_1, \mathbf{D}_2\}$.

Recursive step. *For* $i = 0, 1, \ldots, i_{\max}$*:*

(E step) Run the Kalman filter (2.3)-(2.4) *and RTS smoother* (2.5) *using latent factors* $\{\mathbf{D}_1^{(i)}, \mathbf{D}_2^{(i)}\}$.
Calculate $(\mathbf{\Sigma}^{(i)}, \mathbf{\Phi}^{(i)}, \mathbf{B}^{(i)}, \mathbf{C}^{(i)})$ *using* (2.8).

(M step) Compute $\{\mathbf{D}_1^{(i+1)}, \mathbf{D}_2^{(i+1)}\}$ *using* (2.10).

Output. *State filtering/smoothing pdfs* (2.1.2.1) *and* (2.1.2.1) *along with pointwise estimates of the latent factor from* (2.10).

2.1.3.1 Online implementation

The standard RDL approach presents the drawback of requiring reprocessing the whole dataset in order to apply the update on the linear operators. This implicitly assumes static values over time for those operators during the whole sequence, which may not be well suited for the lack of stationarity in financial data. Furthermore, in such a context, the users may want rapid feedback on the stock price evolution, to immediately decide about to hold, buy or sell. We thus propose here a strategy to make RDL suitable for online processing, reminiscent from online implementations of majorization-minimization algorithms [91].

We set a window (or mini-batch) size τ. At each time step k, the static parameters are estimated using the last τ observations contained in the set $\mathcal{X}_k = \{\mathbf{x}_j\}_{j=k-\tau+1}^{k}$. That is, we run the Kalman filter/smoother, only on the τ

recent observed data, then we update the dictionaries using the smoothing results. This sliding-window strategy has two advantages. First, reducing τ allows for a faster processing. Second, it also allows for better modeling piece-wise linear processes that are varying faster. The price to pay is that a smaller number of observations also limits the estimation capabilities. Hence, there is a trade-off in the seek of an optimal τ, that is generally process-dependent as we will see in the experiments.

A warm start strategy is employed for the Kalman iterations initialization. More precisely, we will set the dictionaries to their last updated value, and we initialize the mean and covariance of the state at $k - \tau + 1$ using the last smoothing results from the last update of the dictionaries in this window.

Note that if $\tau = K$, the algorithm goes back to the original offline version.

2.1.3.2 Forecasting vs trading

The ultimate goal in stock trading is usually to decide whether to buy, hold, or sell a stock in the next day, given the available past data and the expert knowledge, in order to maximize the returns. One can tackle this problem either under the forecasting viewpoint, by considering a prediction (i.e., regression) problem on the future stock price, or under the trading viewpoint, considering the decision making (i.e., classification) among three candidate labels “hold", “buy" and “sell". The regression approach may bring more informed decisions better suited for interpretation and model assessment. Classification in contrast may help in

analyzing the method empirically with metric analysis, and trading simulation. The proposed RDL method can be particularized to either one or the other task, as we explain hereafter.

2.1.3.2.1 Observation model for stock forecasting In order to address the problem of forecasting the value of a given quantity of the market, we will consider the following observation model. For each $k \in \{0, \dots, K-\tau\}$, we observe $(\mathbf{x}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^5$ where $x_j[1]$ is the daily opening price, $x_j[2]$ is the daily adjusted close price, $x_j[3]$ is the daily high value, $x_j[4]$ is the daily low value, and $x_j[5]$ is the daily net asset volume. As explained earlier, running our RDL within this window allows to provide the mean estimate.

$$\hat{\mathbf{x}}_{k+\tau+1} = \mathbf{D}_2 \mathbf{z}^-_{k+\tau} \tag{2.11}$$

associated with a covariance matrix $\mathbf{S}_{k+\tau}$, for the next day (i.e., the day coming right after the end of the observed window) indexed by $k+\tau+1$. Note that, though our model will perform prediction on the whole five-dimensional vector, we will particularly be interested in the ability of our model to perform prediction on a single entry of the vector of interest, in practice the adjusted close price.

2.1.3.2.2 Observation model for trading In the case of trading, we will consider a different set of observed inputs. For each $k \in \{0, \dots, K-\tau\}$, we will observe $(\mathbf{x}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^8$, where $x_j[i]$, $i = 1, \dots, 5$ are the same as in the previous case. Moreover, $[x_j[6], x_j[7], x_j[8]] \in \{0,1\}^3$ represents the decision "hold", "buy", or

"sell", computed daily for each stocks so as to maximize annualized returns. Soft hot encoded scores are used to represent the retained label, e.g., if the decision "hold" is considered for instance at time j, we will observe $x_j[6] = 1$, $x_j[7] = 0$ and $x_j[8] = 0$. Here again, our method is able to provide mean and covariance estimates, for the next day. The class label for the next day trading decision will be simply defined as

$$\ell_{k+\tau+1} = \text{argmax}_{i\in\{1,2,3\}}\, \widehat{x}_{k+\tau+1}[i+5], \tag{2.12}$$

using the same notation as in (2.11).

2.1.3.3 Probabilistic Validation

The probabilistic approach in the Kalman framework provides the distribution of the next observation conditioned to previous data (the so-called predictive distribution of the observations), given by

$$p(\mathbf{x}_k|\mathbf{x}_{1:k-1}) = \mathcal{N}\left(\mathbf{x}_k; \mathbf{D}_2\mathbf{z}_k^-, \mathbf{S}_k\right) \tag{2.13}$$

with $\mathbf{x}_k$ the observation at time k, $\mathbf{S}_k = \mathbf{D}_2(\mathbf{D}_1\mathbf{P}_{k-1}\mathbf{D}_1^\top + \mathbf{Q}) + \mathbf{R}$, and $\mathbf{z}_k^-$, $\mathbf{P}_k$ are defined in Sec. 2.1.2.1 (see [18, Section 4.3] for more details). We can then evaluate the method in a probabilistic manner by inspecting, for each time step, the probability that we had assigned for the events that finally happen. More precisely, from the Gaussian predictive pdf of the observations in Eq. 2.13 provided by the Kalman filter, let us compute the probability mass function

(pmf) $\mathbf{p}_k$ in the two-dimensional simplex, i.e., $0 \leq p_k[i] \leq 1$, $i \in \{1, 2, 3\}$, and $\sum_{i=1}^{3} p_k[i] = 1$. The component $p_k[i]$ contains the probability, estimated by the Kalman filter, that the observation $x_k[i+5]$ will be greater than $x_k[j+5]$, with $j = \{1, 2, 3\} \setminus i$. This can be done by marginalizing the first five components on the predictive distribution Eq. (2.13), obtaining the three dimensional marginal predictive distribution $p(\mathbf{x}_k[6:8]|\mathbf{x}_{1:k-1})$ (slightly abusing the notation). Then, for each dimension, we can compute the probability of the Gaussian r.v. being larger than the other two dimensions. This requires the evaluation of an intractable integral, but can be easily approximated with high precision by direct simulation from $p(\mathbf{x}_k[6:8]|\mathbf{x}_{1:k-1})$. In practice, we will use 10^4 samples from a 3-dimensional standard normal, that can be easily recycled for all time steps (by simply coloring and shifting according to the covariance and mean, respectively). Once we have determined $\mathbf{p}_k$, we can validate the method by using the standard cross-entropy loss as

$$\text{log-loss} = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{3} -(L_k[i] \log(p_k[i])), \tag{2.14}$$

where $\mathbf{L}_k \in \{0, 1\}^3$ represents the ground truth labels at time k (again using soft hot encoding representation).

2.2 Experimental Results

This section will walk you through the experimental setup and empirical results obtained when applied to very challenging application of stock forecasting.

2.2.1 Problem description

We will focus on two distinct problems: (i) the forecasting of next day adjusted close price, and (ii) the prediction of the best next day trading decision (among "hold", "buy" and "sell"). In both cases, we observe daily values of five features of each stock, namely its adjusted close price, opening price, low price, high price, and net asset volume. As an illustration, we display in Fig. 2.2 the evolution of the five observed features, along time, for the symbol `600684.SS`. One can notice that four out the five features (except the net asset volume) are highly correlated, while the net asset volume presents a significantly different evolution. Moreover, the volatility of all features, in particular the adjusted close price we aim at forecasting, can be noticed, thus making the problem quite challenging.

In order to proceed to the trading problem, we also include in the observation vector the ground truth labels that we simply determined using the procedure of [92, Alg.1] aiming at retrospectively maximizing the annualised returns.

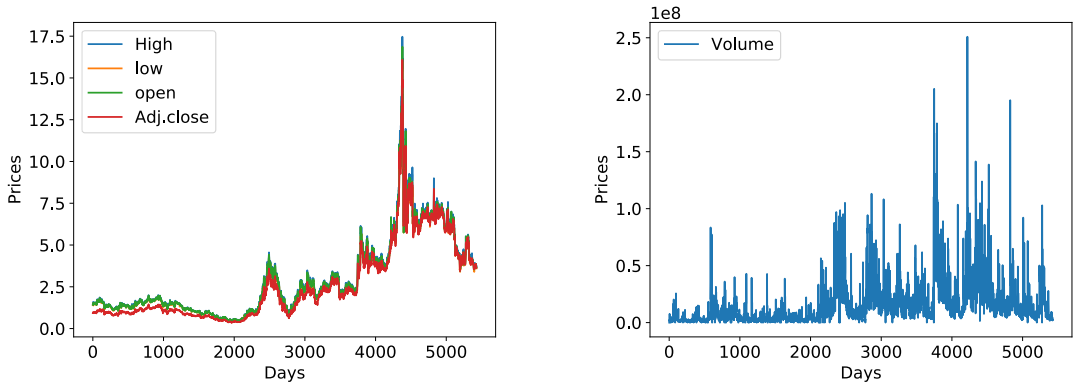


Figure 2.2: Symbol `600684.SS`. (left) Evolution of features adjusted close price, opening price, low price and high price ; (right) Evolution of feature net asset volume.

2.2.2 Compared Methods

We perform comparisons with several methods from the field of machine learning and signal processing, namely CNN-TA [92], MFNN [93], LSTM [57] and ARIMA [10]. Note that those methods are supervised, in the sense that they need

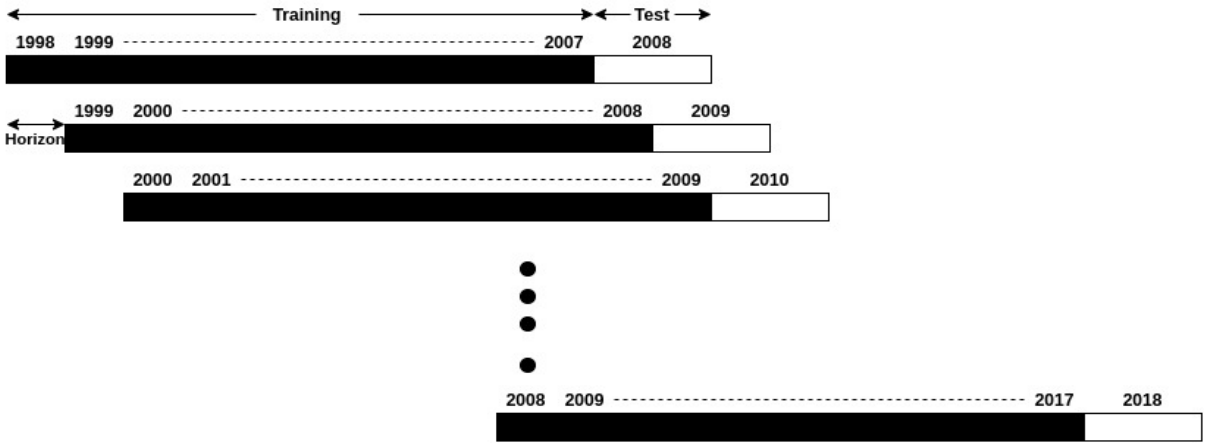


Figure 2.3: Sliding walk-forward validation technique used for hyperparameters tuning of benchmark methods.

a phase of training, from the observation of rather large time series of ground truth features. In contrast, RDL can predict from almost the beginning of the time sequence.

We adopt a "walk-forward" approach, for training ARIMA, CNN-TA, MFNN and LSTM, following the methodology of [92]. It is important to note that we have implictly compared our methods with RNN as MFNN [93] uses a deep network of combination of convolutional neural network and Recurrent neural network. We will be using MFNN in future chapters as benchmark also. The strategy is depicted in Fig. 2.3. We randomly initialize the weights of benchmark methods, for each stock's training. Since these methods are supervised, a large enough time horizon of data is needed as an input to learn the model weights.

Here, we set an horizon of 10 years, which shows a good compromise between model stability and memory requirements. We train the models for 10 years and test for the subsequent 1 year data. Then, we slide by a 1 year horizon, training and testing in the same fashion, and repeating the process until the year 2018. Therefore, in total, 11 years of data (from 2008 to 2018) were used as test data, and have associated prediction results. The training of our proposed RDL approach is done in an online manner, processing the data of a particular window size and then predicting the future time step, in a sliding fashion. This has the major advantage of avoiding the load of very large data, and of providing on-the-fly forecasting results. Let us emphasize that, for the sake of a fair analysis, we evaluate results between RDL and its competitors, for the same 11 years period (2008-2018).

2.2.3 Practical settings

We will apply our RDL method using either the regression model for next day adjusted close price forecasting (i.e., $N_x = 5$) or the classification model for next day trading (i.e., $N_x = 8$), using the models depicted in Sec. 2.1.3.2. We will make use of the sliding window strategy described in Sec. 2.1.3.1, for different values for τ. In all the experiments, $\bar{\mathbf{z}}_0$ is initialized as a zero vector, and $\mathbf{P}_0$, $\mathbf{Q}$ and $\mathbf{R}$ are set as multiple of identity matrix with scale values 10^{-5}, 10^{-2} and 10^{-2}, respectively. We set $N_z = 5$ for the dimensionality of the hidden state, also corresponding to the number of features considered per observation, as it was observed empirically to yield the best results. Moreover, we will set

$\mathbf{D}_1 = \mathrm{Id}$, and will focus on the estimation of $\mathbf{D}_2$. The entries of the latter matrix are randomly initialised at time 0 from a uniform random distribution on $[0, 10^{-1}]$. Warm start strategy is used to initialize for the next processed windows, as explained in Sec. 2.1.3.1. All presented results are averaged on 50 random trials.

For the forecasting problem, we compare the proposed RDL with both ARIMA and LSTM. We set up the ARIMA parameters to $(p, d, q) = (5, 1, 5)$ as it was observed to lead to the best practical performance. LSTM has been modified from its original version, in order to perform forecasting instead of classification. We have removed the softmax output layer and include instead a fully-connected layer to obtain a one node output. Adam optimizer with learning rate 10^{-4}, 200 epochs and batch-size 16 is used to minimize the mean square error (MSE) loss function. The performance are compared in terms of mean absolute error (MAE) and root mean square error (RMSE), on the prediction of the next day adjusted close price, in the test period, averaged over all stocks.

For the trading problem, we compared RDL with CNN-TA, MFNN and LSTM, using their original implementations and settings, described respectively in [92], [93] and [57]. The performance of the methods will be evaluated considering the empirical performance of the model, i.e., how well the model is able to distinct between the three classes. We will also assess the performance of the models in terms of trading efficiency by measuring and comparing their annualised returns. Finally, we will show how the uncertainty quantification provided by RDL (see Sec. 2.1.3.3) can be used efficiently to let the trader decide

where to invest in the market, and diversify his portfolio.

All presented scores in both problems are averaged on 50 random initializations for all methods. Standard deviations of the scores, over the 50 runs, are also provided.

2.2.3.1 Software and hardware specifications

Data pre-processing and RDL model simulations are conducted in Python 3.6, equipped with the Sklearn, NumPy packages and pandas libraries. CNN-TA is developed with Keras, while MFNN, LSTM, are developed with PyTorch. ARIMA simulations are conducted using the statsmodel[1] Python library. We will also rely on Sklearn, Ta-lib [2] and Ta4j [3] libraries for evaluating the quantitative indicators. All the experiments are carried on using a Dell T30, Xeon E3-1225V5 3.3GHz with GeForce GT 730, and Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz 16GB RAM, 200GB HDD, equipped with an Nvidia 1080 8GB and Ubuntu OS.

2.2.4 Numerical results for the forecasting model

We first present the analysis for stock forecasting (regression). We will evaluate the methods in their ability to predict a single feature value, here the daily adjusted close price. The five input features are normalized so that their values range in the same scale, to limit bias towards large values features such as the net asset volume. We analyse the performance of RDL approach for different

[1] www.statsmodels.org
[2] http://ta-lib.org
[3] http://www.ta4j.org

window size and we compare it with two state of the art methods for stock forecasting, namely LSTM and ARIMA.

2.2.4.1 Influence of window size

Table 2.1 displays the performance, in terms of MAE and RMSE, of our RDL method when using different window size τ in the online implementation (see Sec. 2.1.3.1). We can observe that the performance is improving as the window size increases, until reaching a plateau. This trend is confirmed by Figure 2.4, which illustrates the forecasting results of RDL for several values of τ, on four representative cases. While it is clear that $\tau = 5$, 10 or 15 may not be sufficient to learn appropriately the model, we can observe that from $\tau = 25$, the forecasting results start getting satisfying. In the further experiments on forecasting, we will set $\tau = 50$, as it leads to a good compromise between time complexity and prediction accuracy.

2.2.4.2 Comparison with benchmark models

Table 2.2 provides the comparative performance of prediction of next day closing price feature when using RDL, LSTM or ARIMA. It is noticeable that RDL outperforms the two benchmark models. It is also very stable, as can be seen in the reported standard deviation values. Note that the LSTM approach is rather computationally expensive since it took 8 days to train for 185 stocks for 10 years dataset, in contrast with RDL (resp. ARIMA) which requires around 2 hours (resp. 3 hours) for the same task. Fig. 2.5 illustrates the closing price

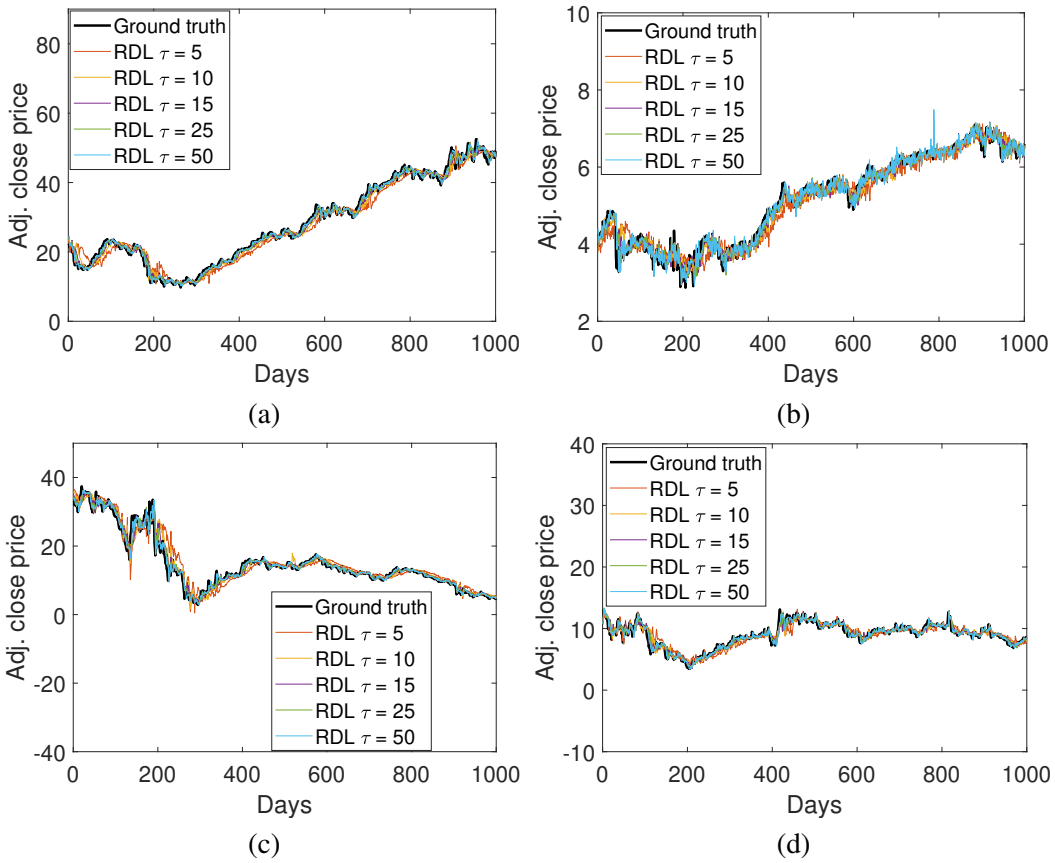


Figure 2.4: Forecasting results of RDL method, using different values for the window size τ: (a) `AAPL`, (b) `ABX`, (c) `BAC`, (d) `600841.SS`. We only display here a period of 1000 days for the sake of readibility.

prediction results, using the proposed RDL, LSTM and ARIMA method for four representative cases. We can see that LSTM tends to underestimate the adjusted close price. ARIMA performs well on simple cases (b-c), but has difficulty to track the price evolution in other cases (a-d). In contrast, RDL is able to provide satisfying results in all the considered examples.

Window τ	MAE	RMSE
5	0.14	13.75
10	0.13	12.67
15	0.12	11.97
25	0.09	11.93
50	0.05	10.25
75	0.03	10.14
100	0.02	10.12
125	0.02	10.11

Table 2.1: Forecasting results of RDL for different window size.

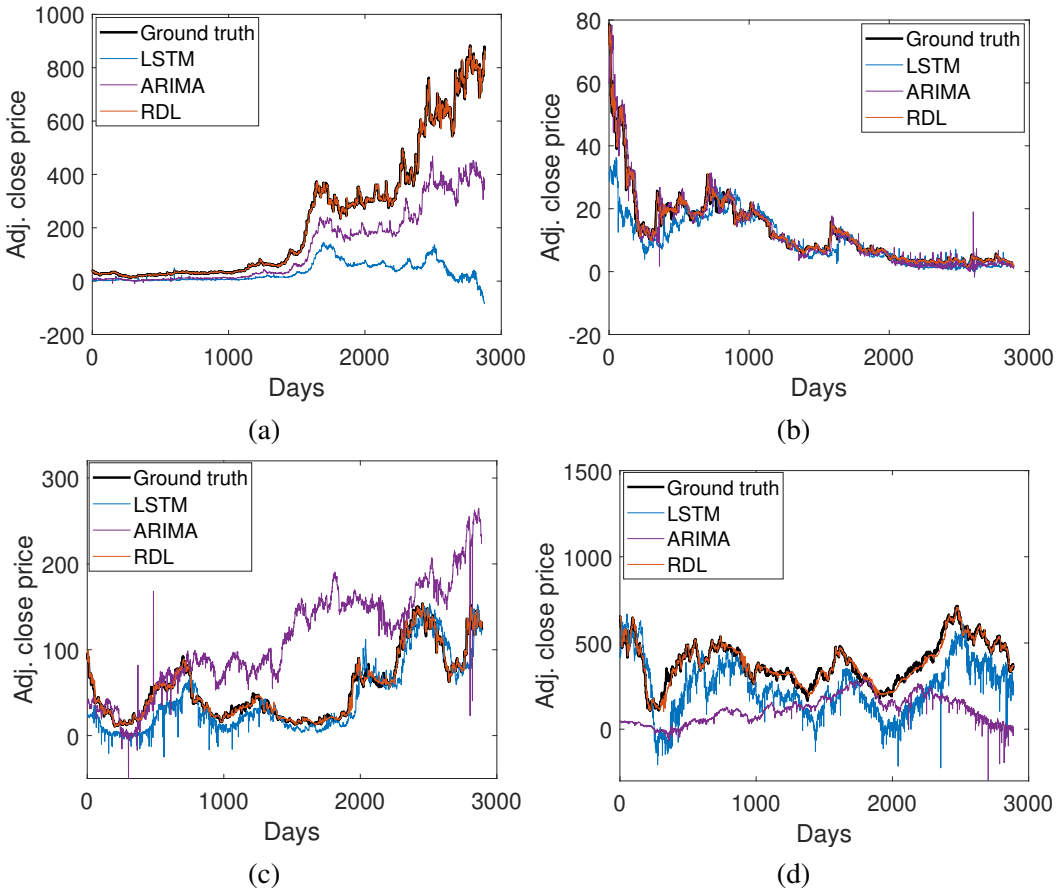


Figure 2.5: Forecasting results of RDL, LSTM and ARIMA models: (a) `ALKYLAMINE.BO`, (b) `ALOKTEXT.BO`, (c) `SPICEJET.BO`, (d) `TATASTEEL.NS`.

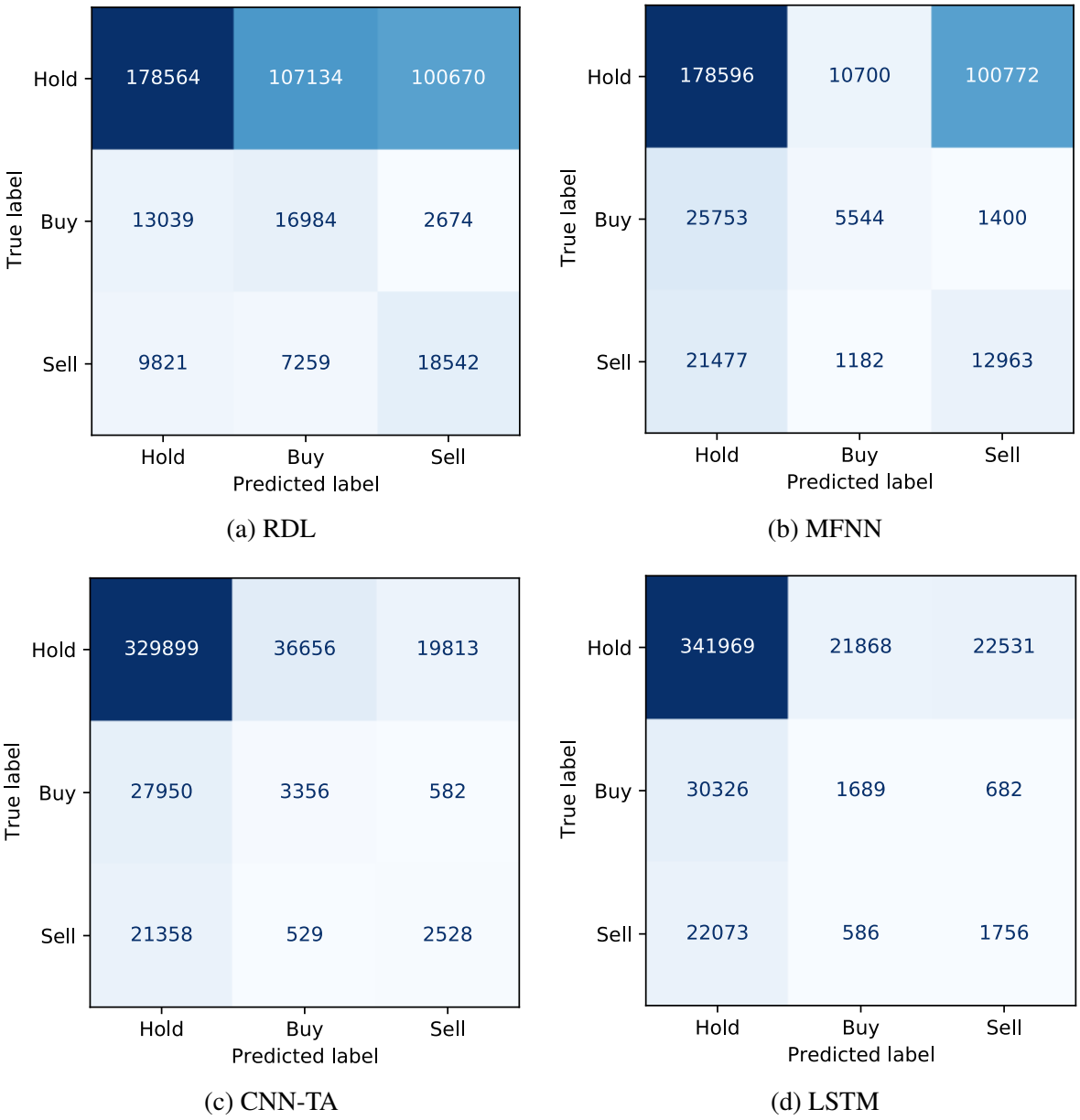
2.2.5 Numerical results for the trading model

We now present the results when focusing on the trading problem, that is the prediction for the decision “hold”, "buy” or “sell” for next day.

Method	MAE	RMSE
RDL	**0.05 (0.002)**	**10.25 (0.32)**
ARIMA	1.23 (0.04)	78.6 (1.28)
LSTM	6.12 (0.02)	297.9 (1.27)

Table 2.2: Comparison of methods for the forecasting problem: mean (standard deviation) of MAE and RMSE scores, over 50 random initializations

.



(a) RDL (b) MFNN

(c) CNN-TA (d) LSTM

Figure 2.6: Confusion matrices for trading decision.

2.2.5.1 Influence of the window size

Table 2.3 depicts the experimental performance of RDL, while Fig 2.7 shows the computational time required to train and test RDL method, on the whole dataset, for different window sizes. As in the forecasting case, the window size plays also plays an important role, as it is essential to analyze the apt window

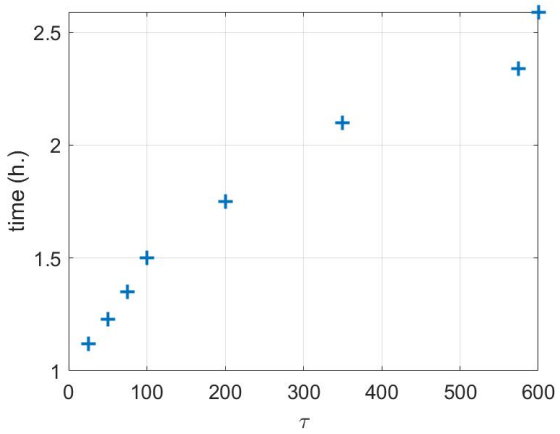


Figure 2.7: Time for processing the dataset (train and test), in hours, using RDL with different window sizes τ.

size to produce optimal predictions while preserving a reasonable computational time. The performance tends to improve as the window size increases, since the model incorporates more data. Moreover, the computational time increases when τ increases, but in a reasonable manner. We notice then a stabilization of the performance, from $\tau = 575$. This value will be retained in the upcoming experiments.

Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
25	0.85	0.10	0.10	0.85	0.12	0.12	0.84	0.10	0.10
50	0.85	0.10	0.10	0.74	0.17	0.14	0.80	0.10	0.12
75	0.85	0.10	0.10	0.74	0.18	0.15	0.78	0.11	0.12
100	0.85	0.10	0.10	0.74	0.20	0.14	0.78	0.10	0.12
200	0.85	0.10	0.09	0.68	0.25	0.22	0.72	0.13	0.14
350	0.82	0.10	0.10	0.62	0.30	0.31	0.68	0.15	0.15
575	0.88	0.15	0.12	0.45	0.52	0.51	0.59	0.19	0.23
600	0.87	0.15	0.14	0.45	0.52	0.52	0.59	0.19	0.22

Table 2.3: Classification results of RDL for different window sizes.

2.2.5.2 Empirical result discussion

We present in Fig. 2.6 the confusion matrices which summarized the classification performance for the 185 stocks by the proposed approach, as well as for the

three competitors. First, we see that for all the approaches, the Hold class is better predicted than the two other classes, LSTM getting the best scores over the competitors in terms of false negative. However, LSTM (and to a mild extent, CNN-TA and MFNN) present a large number of false positive for the Hold class. Those methods seem to adopt an over-conservative strategy privileging the Hold class over the others. This contrasts with the results obtained by RDL, which is able to preserve a certain balance among classes. This can be seen in the confusion matrices, by noticing that the maximum values are taken in the diagonal, as it should be expected by a valid classifier. To complete this analysis, we present in Table 2.4 other classification metrics, namely recall, precision, and F1 Score of the proposed approach against the three deep learning models. RDL clearly outperforms the competitors, which is consistent with the previous observations. In this trading problem, all methods report rather small standard deviation values. LSTM and CNN-TA appear to be slightly more stable than the other methods. But this is however at the price of very poor classification scores for the classes Sell and Buy.

Method	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
RDL	**0.88** (0.01)	**0.15** (0.01)	**0.12** (0.01)	**0.45** (0.02)	**0.52** (0.01)	**0.51** (0.013)	**0.59** (0.01)	**0.19** (0.016)	**0.23** (0.012)
MFNN	0.79 (0.017)	0.11 (0.017)	0.04 (0.007)	0.47 (0.008)	0.37 (0.011)	0.16 (0.011)	0.58 (0.012)	0.11 (0.008)	0.06 (0.012)
LSTM	0.84 (0.06)	0.07 (0.016)	0.06 (0.007)	0.89 (0.009)	0.05 (0.007)	0.05 (0.005)	0.86 (0.005)	0.05 (0.0014)	0.05 (0.0034)
CNN-TA	0.84 (0.008)	0.11 (0.009)	0.09 (0.015)	0.85 (0.007)	0.07 (0.01)	0.10 (0.014)	0.85 (0.008)	0.08 (0.023)	0.09 (0.009)

Table 2.4: Comparison of methods for the trading problem; mean (standard deviation) of Precision, Recall and F1 Score, over 50 random initializations

.

Additionally, in Table 2.5 we present the computational time required by RDL, for the retained $\tau = 575$, and the benchmark models, to process the whole dataset. As can be seen, neural network models requires a lot of time while

training-testing due to their complex architectures and huge amount of data that has to be processed. In contrast, the online training procedure we proposed for our RDL model allows fast processing of the data, in a sliding window fashion. Moreover, the resulting train model is a simple linear SSM, thus less costly to evaluate during testing phase than complex deep networks.

Method	Time cost (h.)	Acc. Buy	Acc. Sell
RDL	**2.34**	0.65	0.63
MFNN	3.87	0.44	0.34
LSTM	9.5	0.33	0.31
CNN-TA	7.54	0.45	0.48

Table 2.5: Averaged time over 50 random runs for processing the dataset (train and test), in hours, for RDL and its competitors.

2.2.5.3 Annualised returns analysis

Investors consider one more parameter while trading for stocks, ie.. annualized returns (AR). AR is a critical factor as it helps the investors to decide the monetary gain/loss they are going to incur while investing in the stock. The predictions (buy, sell, hold) provided by the model are then used to carry out the market simulation where a similar virtual environment is created by buying, selling, and holding the stock as per the predictions. The market simulations are a strategy that mimics the investor/trader who makes trading decisions according to signals from the algorithmic model using real money, which then allows to compute AR. Table 2.6 presents the AR results obtained when applying the market simulations strategy from [92], to the trading predictions provided by our proposed model and the other benchmark techniques, on 9 randomly selected stocks, as well as the average AR for all stocks. The best-annualized returns are highlighted

in bold. One can notice that the proposed method leads to higher returns in averaged during the test period of 10 years when compared to AR obtained using the trading predictions from the deep learning techniques.

Stock symbols	RDL	CNN-TA	MFNN	LSTM
WIPRO.BO	**1.37**	-18.14	-27.81	-47.74
AAPL	**13.44**	0	12.92	0
AMZN	0.06	**30.64**	-20.85	-0.15
IOC.BO	**6.28**	-3.03	-26.42	-3.1
TATACHEM.BO	**3.91**	-1.54	-8.32	0
SPICEJET.BO	**10.58**	-24.08	-28.21	0
ATML	**3.76**	-33.25	-27.07	-33.82
DOM.L	1.76	0.11	**8.22**	0.47
INDRAMEDCO.BO	**5.61**	-14.22	-3.53	-50.86
Average on all 185 stocks	**5.19**	-5.08	-11.45	-13.02

Table 2.6: Annualized Returns

	Stock symbols	Log-Loss
Small-cap	ALOKTEXT.BO	1.11
	ALKYLAMINE.BO	1.22
	ZEEMEDIA6.BO	1.03
	PVP.BO	0.42
Mid-cap	IOC.BO	1.25
	TATACHEM.BO	0.56
	SPICEJET.BO	0.47
	BHEL.BO	0.01
Large-cap	AAPL	0.85
	AMZN	0.27
	HINDZINC.BO	1.03
	ONGC.BO	0.007
	SIEMENS.NS	0.001

Table 2.7: Log-loss values provided by RDL, for the stocks with available market capitalization categories.

2.2.5.4 Portfolio diversification

Many investment professionals claim that the key to smart trading is "diversification of portfolio" [72]. Diversification allows investors to maximize returns by investing in different domains that would react differently in the same market events, thus managing better the risk. Ideally, a diversified portfolio would

contain stocks with various levels of market capitalization. Market capitalization refers to the total market value of all outstanding shares [73]. Companies can be divided according to their market capitalization value, into small-cap, mid-cap, large-cap, as we will explain hereafter in detail. It is always beneficial to have a diversified portfolio that includes stocks from the three classes of companies.

Small-cap companies are young and seek to expand aggressively. As comes the growth potential, the risk factor follows in a direct proportion. Small-caps are more volatile and vulnerable to losses during the negative trends (downtime) of the financial market.

Mid-cap is a pooled investment that focuses on including stocks with a middle range of market capitalization. Mid-cap offers investors more enormous growth potential than large-cap stocks, but with less volatility and risk as compared to small-caps.

Large-cap companies are typically large, well established, and financially sound. Large-cap is the market leader, relatively less volatile, and has a steady risk-return factor with relatively lower risk compared to mid-cap, small-cap.

As explained in Sec 2.1.3.3, the proposed RDL methodology allows to provide through probabilistic quantification, a piece of information that can be used by a knowledgeable practitioner to trade accordingly to have a diversified portfolio. Such probabilistic validation can help maximize the returns by having a mix of small-cap, mid-cap, large-cap stocks.

We provide in Table 2.7 the log-loss value (the smaller, the better), computed as described in Sec. 2.1.3.3, computed for RDL results, for the few stocks of

the dataset for which the market capitalization categories were available.[4] The log-loss quantifies the accuracy of the probabilistic predictions of the proposed method, penalizing more the situations where the method assigns a low probability to an event that finally occurs. One can notice that the log-loss value can reach very low value (i.e., good prediction accuracy) when trading the large cap stocks, probably due to the less volatile nature of such stocks. In contrast, the log-loss is in average higher for small caps, as they are highly volatile.

2.2.5.5 Result Discussion

One important difference between RDL and the competitors is that our approach is not supervised, in the sense that we do not need to load a long time series to learn the model in a batch manner. Therfore, RDL processes data as it comes and in a sequential manner. This is convenient in terms of memory and processing cost. This perspective helps to explain the good results of RDL, as it always uses all available information to make a one day ahead prediction, while keeping a low complexity. The challenge for RDL, that we also tackle in this work, is to adequately learn the linear matrices of the model from small time windows (maximum 600 days in our tests), and jointly, perform the forecasting. The efficient learning and prediction tasks are supported by Kalman filter/smoother combined with the EM. The flexibility of RDL, with potentially different linear operators along time (the linear operators are updated from a window to the next one), is beneficial to model non-linear and non-stationary time series. Moreover,

[4] https://finance.yahoo.com/screener

the Gaussian linear SSM yields simple updates for the EM, reducing considerably the 'training' time. In contrast, if we trained the benchmark models, on each small sliding window, it would be very costly, and most likely with a reduced performance (the models would be over-parametrized). Under this perspective, we can consider that RDL deals with a piece-wise linear state-space model with smooth changes of the linear operators (that are estimated over time), while still retaining the linear-Gaussian structure which allows to extract all the benefits of the Kalman filtering/smoothing.

Chapter 3

Sequential Transform learning

In this chapter we will discuss sequential transform learning. This work is proposed to enhance the efficiency of recurrent state space models to classify time series signals. The proposed work is a linear Gaussian Markovian state-space model involving state evolution, observation matrices, and an exogenous control input. The resultant formulation resembles loosely based on transform learning. The addition of exogenous input along with the state evolution matrices differentiates the method from RDL approach (discussed in chapter 2). The presence of exogenous input helps to establish more informed prior for state-space evolution. Our approach keeps the best of two worlds, marrying the interpretability and uncertainty measure of signal processing with the function approximation ability of neural networks. The proposed method is applied to very challenging application in finance domain ie., stock forecasting. Stock forecasting is an evolutionary endeavor in predicting real-time returns for an asset. The stock market has always been complex and challenging to forecast due to data intensity, non-stationarity, lack of structure, noise, high degree

of uncertainty and hidden relationships between the variables [94]. However, forecasting the future has always been the holy grail of financial investment due to which it has been one of the challenging problems for researchers and traders since several decades [5–7, 95–99].

In this work we propose a novel approach to bridge the gap between state space models and neural network approaches. We propose time series analysis technique called *Sequential Transform Learning* (STL). Our approach has the interpretability of multivariate signal processing coupled with the ability to learn from data. Our approach needs no structural assumptions in advance and can quantitatively model uncertainty, thereby keeping the best of both worlds.

3.1 Proposed Method

The SSM approaches mentioned earlier in section 1.2 requires the specification of the underlying deterministic equation for state evolution and observation models. Defining such equations and model behavior in advance is difficult as the system's nature is highly volatile with real-time applications like stock forecasting. The neural network based approaches do not require specifying the dynamical model (they can learn from data) but they cannot specify the uncertainty (covariance) around the point estimates they provide. Modeling this uncertainty is crucial in financial analysis [100, 101]. In this work, we propose *Sequential Transform Learning*(STL) for time series modeling. The model incorporates the advantages of both SSMs and neural networks. The neural

network type formulation comes from the definition of 'Transform Learning'. It is defined as a technique where a transform (T) analyses the data (u) to produce coefficients (z), i.e. Tu = z. This establishes the neural network type structure of our approach. In Fig3.1 'z' represents the extracted coefficient(features), 'T'

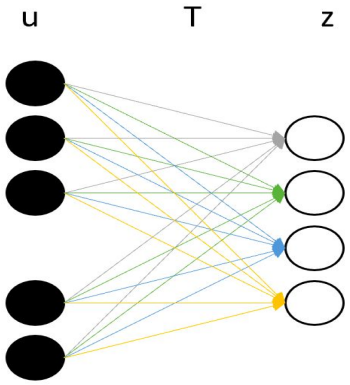


Figure 3.1: Schematic Diagram of Transform Learning

represents the transform(weights) and 'u' represents the Data. Schematically this is shown in Fig3.1. The model does not need any assumptions about the system's dynamical nature, readily learns from the data, and provides feedback to the model from the past prediction, just like a recurrent neural network. This section describes in detail the multivariate time-series prediction followed by the inference technique called the expected minimization (EM) approach.

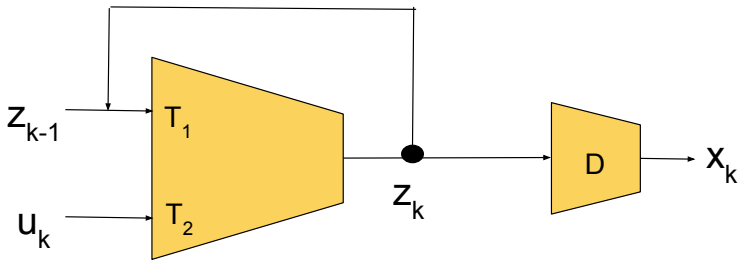


Figure 3.2: Block Diagram representing Sequential Transform Learning. U_k represents the external input,Z_{k-1} represents the state space vector. T_1 and T_2 are the transforms, mapping D . x_k represents the output that we wish to estimate.

3.1.1 Sequential transform learning

The input to our model consists of the current data-point ($\mathbf{u}_k$) and the representation from the previous instance (z_{k-1}). This is transformed (by $\mathbf{T}$) to the current representation $\mathbf{z}_k$. The representation $\mathbf{z}_k$ is mapped (by D) to the output $\mathbf{x}_k$. The schematic diagram is represented in fig3.2.

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= \mathbf{T}\mathbf{Y} + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{D}\mathbf{z}_k + \mathbf{v}_{2,k}, \end{cases} \tag{3.1}$$

where $\mathbf{T} = \begin{bmatrix} \mathbf{T}_1 | \mathbf{T}_2 \end{bmatrix}$ and $\mathbf{Y} = \begin{bmatrix} \mathbf{z}_{k-1} \\ \mathbf{u}_k \end{bmatrix}$.

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= \mathbf{T}_1\mathbf{z}_{k-1} + \mathbf{T}_2\mathbf{u}_k + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{D}\mathbf{z}_k + \mathbf{v}_{2,k}, \end{cases} \tag{3.2}$$

The resultant formulation loosely resembles transform learning [102] and assume both the representation and the mapping functions are corrupted by Gaussian noise $(\mathbf{v}_{1,k})_{1 \leq k \leq K}$ with covariance matrix $\mathbf{Q}$ and $(\mathbf{v}_{2,k})_{1 \leq k \leq K}$ with covariance matrix $\mathbf{R}$ respectively. Note the similarity of our model to that of a linear state-space system with exogenous inputs. With this, Eq.(3.2) represents a First order linear-Gaussian model where $(\mathbf{z}_k)_{1 \leq k \leq K}$ is the sequence of K unknown states. Our objective is the joint inference, from the observed sequence

$(\mathbf{x}_k)_{1\leq k\leq K}$ and $(\mathbf{u}_k)_{1\leq k\leq K}$, of the transforms $\mathbf{T}_1 \in \mathbb{R}^{N_z\times N_z}$ and $\mathbf{T}_2 \in \mathbb{R}^{N_z\times N_x}$, matrix $\mathbf{D} \in \mathbb{R}^{N_x\times N_z}$ and of the predicted sequence $(\mathbf{z}_k)_{1\leq k\leq K}$. Note that the mapping term stems from the label consistent transform learning formulation introduced in [103]

3.1.2 STL inference Algorithm

Sequential transform learning is used when both the predicted sequence and the linear operators must be inferred from the data and the external control input assuming gaussianity. We propose to solve the problem in an alternating manner, following the expectation-Maximization scheme introduced in [18, chap.12] (see also [2]).

3.1.2.1 Representation Update

The model Sequential Transform Learning assumes other variables to be fixed for estimating the state $\mathbf{z}_k$. Assume that the initial state follows $\mathbf{z}_0 \sim \mathcal{N}(\bar{\mathbf{z}}_0, \mathbf{P}_0)$, where $\mathbf{P}_0$ is a symmetric definite positive matrix of $\mathbb{R}^{N_z\times N_z}$ and $\bar{\mathbf{z}}_0 \in \mathbb{R}$. The process noise $(v_{1,k})_{1\leq k\leq K}$ is zero-mean Gaussian with covariance matrix Q, and that the observation noise $(v_{2,k})_{1\leq k\leq K}$ is zero-mean Gaussian with covariance matrix R. Then, (3.2) reads as a first-order Markovian linear-Gaussian model where $(\mathbf{z}_k)_{1\leq k\leq K}$ is the sequence of K unknown states. The Kalman filter provides a probabilistic estimate of the hidden state at each time step k, conditioned

to all available data and external control input information up to time k.

$$p(\mathbf{z}_k|\mathbf{x}_{1:k}, \mathbf{u}_{1:k}) = \mathcal{N}(\mathbf{z}_k; \bar{\mathbf{z}}_k, \mathbf{P}_k). \tag{3.3}$$

For every $k \in \{1, \ldots, K\}$, the mean $\mathbf{z}_k$ and covariance matrix $\mathbf{P}_k$ can be computed by means of the Kalman filter recursions given as follows.

For $k = 1, \ldots, K$

Predict state:

$$\begin{cases} \mathbf{z}_k^- &= \mathbf{T}_1 \bar{\mathbf{z}}_{k-1} + \mathbf{T}_2 \mathbf{u}_k, \\ \mathbf{P}_k^- &= \mathbf{T}_1 \mathbf{P}_{k-1} \mathbf{T}_1^\top + \mathbf{Q} \end{cases} \tag{3.4}$$

Update state:

$$\begin{cases} \mathbf{y}_k &= \mathbf{x}_k - \mathbf{D}\mathbf{z}_k^-, \\ \mathbf{S}_k &= \mathbf{D}\mathbf{P}_k^- \mathbf{D}^\top + \mathbf{R}, \\ \mathbf{K}_k &= \mathbf{P}_k^- \mathbf{D}^\top \mathbf{S}_k^{-1}, \\ \mathbf{z}_k &= \mathbf{z}_k^- + \mathbf{K}_k y_k, \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top. \end{cases} \tag{3.5}$$

The Rauch-Tung-Striebel (RTS) smoother makes a backward recursion on the data which makes use of the filtering distributions computed by the Kalman filter in order to obtain the smoothing distribution $p(\mathbf{z}_k|\mathbf{x}_{1\ldots K}, \mathbf{u}_{1\ldots K})$. In the following we summarize the RTS recursions:

For $k = K, \ldots, 1$

Backward Recursion (Bayesian Smoothing):

$$\begin{cases} \mathbf{z}_{k+1}^{-} & = \mathbf{T}_1 \bar{\mathbf{z}}_k + \mathbf{T}_2 \mathbf{u}_k, \\ \mathbf{P}_{k+1}^{-} & = \mathbf{T}_1 \mathbf{P}_k \mathbf{T}_1^{\top} + \mathbf{Q}, \\ \mathbf{G}_k & = \mathbf{P}_k \mathbf{T}_1^{\top} [\mathbf{P}_{k+1}^{-}]^{-1}, \\ \mathbf{z}_k^s & = \mathbf{z}_k + \mathbf{G}_k [\mathbf{z}_{k+1}^s - \mathbf{z}_{k+1}^{-}], \\ \mathbf{P}_k^s & = \mathbf{P}_k + \mathbf{G}_k [\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}] \mathbf{G}_k^{\top}. \end{cases} \tag{3.6}$$

3.1.2.2 Transform and map Update

For estimating the transform $\mathbf{T}$ and the mapping $\mathbf{D}$, we assume the sequence of states ($\mathbf{z}_k$) to be fixed, following the methodology of [18][Th.12.4] which implements expectation-minimization(EM) approach. In order to estimate the parameters in state space, the joint likelihood can be introduced as the following function, more detailed derivation can be found in [section 2] [2](Please refer [chapter 12] [83]):

$$\begin{aligned} &\mathbf{Q}(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D}) \\ &= \frac{1}{2} \log |2\pi \mathbf{P}_0| + \frac{K}{2} \log |2\pi \mathbf{Q}| + \frac{K}{2} \log |2\pi \mathbf{R}| \\ &+ \frac{1}{2} \mathrm{tr} \left(\mathbf{P}_0^{-1} (\mathbf{P}_0^s + (\mathbf{z}_0^s - \bar{\mathbf{z}}_0)(\mathbf{z}_0^s - \bar{\mathbf{z}}_0)^{\top}) \right) \\ &+ \frac{K}{2} \mathrm{tr} \left(\mathbf{Q}^{-1} (\mathbf{\Sigma} - \mathbf{T}_1^{\top} \mathbf{C} - \mathbf{A} \mathbf{T}_2^{\top} - \mathbf{T}_1 \mathbf{C}^{\top} + \mathbf{T}_1 \mathbf{\Phi} \mathbf{T}_1^{\top} \right. \\ &\left. + \mathbf{T}_1 \mathbf{F} \mathbf{T}_2^{\top} - \mathbf{T}_2 \mathbf{A}^{\top} + \mathbf{T}_2 \mathbf{F}^{\top} \mathbf{T}_1^{\top} + \mathbf{T}_2 \mathbf{I} \mathbf{T}_2^{\top}) \right. \\ &+ \frac{K}{2} \mathrm{tr} \left(\mathbf{R}^{-1} (\mathbf{\Delta} - \mathbf{B} \mathbf{D}^{\top} - \mathbf{D} \mathbf{B}^{\top} + \mathbf{D} \mathbf{\Sigma} \mathbf{D}^{\top}) \right) \end{aligned} \tag{3.7}$$

where $(\mathbf{\Sigma}, \mathbf{\Phi}, \mathbf{B}, \mathbf{C}, \mathbf{A}, \mathbf{F}, \mathbf{I})$ are defined from the outputs of the previously described RTS recursion:

$$\begin{cases} \mathbf{\Sigma} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{P}_k^s + \mathbf{z}_k^s(\mathbf{z}_k^s)^\top, \\ \mathbf{\Phi} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{P}_{k-1}^s + \mathbf{z}_{k-1}^s(\mathbf{z}_{k-1}^s)^\top, \\ \mathbf{B} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_k(\mathbf{z}_k^s)^\top, \\ \mathbf{C} &= \frac{1}{K}\sum_{k=1}^{K} (\mathbf{P}_k^s\mathbf{G}_{k-1}^\top + \mathbf{z}_k^s(\mathbf{z}_{k-1}^s)^\top). \\ \mathbf{A} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{z}_k^s\mathbf{u}_k^\top \\ \mathbf{F} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{z}_{k-1}^s\mathbf{u}_k^\top \\ \mathbf{I} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{u}_k\mathbf{u}_k^\top \\ \mathbf{\Delta} &= \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_k\mathbf{x}_k^\top \end{cases} \tag{3.8}$$

Then, the above function is a lower bound of the marginal log-likelihood $\varphi(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D}) = \log p(\mathbf{x}_{1:K}|\mathbf{T}_1, \mathbf{T}_2, \mathbf{D})$, i.e.,:

$$\log p(\mathbf{x}_{1:K}|\mathbf{T}_1, \mathbf{T}_2, \mathbf{D}) \geq \mathcal{Q}(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D}) \quad (\forall(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D})) \tag{3.9}$$

The update for $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{D}$ amounts for maximizing this lower bound, so as to increase value the marginal log-likelihood (more details can be found in [eq. 12.23] [83]). Due to the quadratic form of function $\mathcal{Q}(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D})$, the maximization step takes a closed form, leading to the transform updates:

$$\begin{cases} \mathbf{T}_1^+ &= (\mathbf{C} - \mathbf{T}_2\mathbf{F}^\top)\mathbf{\Phi}^{-1}, \\ \mathbf{T}_2^+ &= (\mathbf{A} - \mathbf{T}_1\mathbf{F}), \\ \mathbf{D}^+ &= \mathbf{B}\mathbf{\Sigma}^{-1}. \end{cases} \tag{3.10}$$

The EM algorithm finally reads as the alternation of the Kalman filter/smoother, with fixed parameters $(\mathbf{T}_1, \mathbf{T}_2, \mathbf{D})$ (E-step) and the update of the transforms (M-step) [18] [Alg.12.5]. The advantage of the proposed method to estimate transform updates, as compared to alternating minimization strategies more commonly used in transform learning literature is threefold: (i) the EM algorithm, being based on a majoration-minimization strategy, is guaranteed to converge to a local optimum, (ii) it makes an explicit use of the covariance estimation provided by Kalman, and thus accounts explicitly on the uncertainty one may have on the estimates, (iii) it has a low complexity thanks to a smart use of the filtering/smoothing sequences.

3.2 Application to stock forecasting

This section focus on STL application to stock forecasting. The model aim of performing trading task from from technical indicators. We describe in this section how STL is adapted and implemented to map the specified characterstics of the challenging application.

3.2.1 Online implementation

The standard STL approach is applied to the entire dataset, but it presents a drawback of reprocessing the entire dataset to update the linear operators. This requires assuming some fixed values of those operators for the whole process, which may not be correct due to absence of stationarity in the data. It is important

that the linear operator should also evolve as the trend changes. This timely evolution of linear operator ensures better model behavior and rapid feedback on stock price evolution. The rapid feedback on model behavior helps to follow the trend and make immediate decisions to hold, buy, or sell. We propose a strategy to make STL more flexible to online implementation. The strategy provides the freedom to update the linear operators from the hidden states, observations, and external control input.

To implement the online processing , we propose a sliding window strategy. We set a window (or mini-batch) size τ. At each time step k, the static parameters are estimated using the last τ observations contained in the set $\mathcal{X}_k = \{\mathbf{x}_j\}_{j=k-\tau+1}^{k}$ and $\mathcal{U}_k = \{\mathbf{u}_j\}_{j=k-\tau+1}^{k}$. The model then runs Kalman filter/smoother, only on the τ recent observed data, followed by updating the transforms and observation transition space using the smoothing results. The online implementation strategy reduces the processing time as it is faster to process τ days window rather than processing the entire sequence. These windows are chosen in overlapping fashion, moving ahead with stride one. The overlapping methodology allows for better modeling of piece-wise linear processes that are varying faster. The method requires to pay a cost of estimation capabilities with less number of observation. Hence, there is a trade-off in the seek of an optimal τ, which is generally process-dependent, as we will see in the experiments.

A warm start strategy is employed for the Kalman iterations initialization. More precisely, we will set the linear operators(transforms and observation transition space) to their last updated value, which helps transfer the temporal

information to the next windows. We initialize the mean and covariance of the state at $k - \tau + 1$ using the last smoothing results from the last update of the linear operators(transforms and observation transition space) in this window.

The ultimate goal in stock trading is usually to decide whether to buy, hold, or sell a stock in the next day, given the available past data and the expert knowledge, to maximize the returns. The problem can be viewed as a trading problem considering the decision making (i.e., classification) among three candidate labels, "Buy", "Sell," and "Hold". Classification of signals into these three class help in analyzing the method empirically with metric analysis, and trading simulation.

3.2.1.1 Simple Moving Average

Simple moving average (SMA) calculates the average of the a fixed range of prices, usually closing price by the number of period in that range.

$$SMA = \frac{c_1 + c_2 + ... + c_n}{n} \tag{3.11}$$

where c_n= closing price of an asset for period n and n= number of total periods

3.2.2 Observation model for trading

Sequential transform learning is applied to stock forecasting in an online sliding window fashion. We present two versions of STL approach.

3.2.2.1 Classical approach (STL_1)

For implementing online trading through STL_1 approach, we will consider individual stocks for trading. The model predicts one day ahead label prediction for the number of days for each stock available in the dataset. This will help the investor decide better for which day they should perform, buy, sell, or hold action from the data available for each stock to maximize their profit.

For each $k \in \{0, \ldots, K-\tau\}$, we will observe $(\mathbf{x}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^1$ where x_j is the represent the decision "hold"(label:0), "buy"(label:-1), or "sell"(label:1), computed daily for each stocks so as to maximize annualized returns and $(\mathbf{u}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^1$ represent the values of technical indicator (SMA) computed daily for close price. In the experiment, $\bar{\mathbf{z}}_0$ is initialized as a zero vector, and $\mathbf{P}_0$, $\mathbf{Q}$ and $\mathbf{R}$ are set as multiple of identity matrix with scale values 10^{-5}, 10^{-2} and 10^{-2}, respectively. We set $N_z = 3$ for the dimensionality of the hidden state and $N_x = 1$ which corresponds label feature for which model evaluates one day ahead label prediction as described in Sec. 3.1.2. The transform T and mapping D are randomly initialised at time 0 from a uniform random distribution. The last updated values of transforms is used to initialize for the next processed windows, as explained in Sec. 3.2.1. All presented results in Sec.3.3.4.2 for (STL_1) are averaged on 50 random trials. Here again, our method is able to provide mean and covariance estimates, for the next day. Note that state-of-the-art methods like CNN-TA, MFNN and LSTM follows same strategy of experimentation.

3.2.2.2 Market Prediction approach (STL_2)

To analyze the market behavior and provide investors a better-informed decision, we evaluated our approach STL_2 over the entire market. The STL_2 approach predicts one day ahead label predictions for a list of N stocks in the dataset ie., for each day, the method predicts one day ahead label prediction for N stocks simultaneously.

For each $k \in \{0, \ldots, K-\tau\}$, we will observe $(\mathbf{x}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^N$ where $x_j[i]$, $i = 1, \ldots, N$ represent the decision "hold"(label:0), "buy"(label:-1), or "sell"(label:1), computed daily for N stocks so as to maximize annualized returns and $(\mathbf{u}_j)_{k \leq j \leq k+\tau} \in \mathbb{R}^N$ represent the values of technical indicator SMA computed daily for N stocks closing price. The model predicts one day ahead label prediction for a list of N stocks. This will help the investor to decide better for which stock they should perform buy, sell or hold action from a list of N stocks to maximize their profit. Here again, our method can provide mean and covariance estimates for the next day. In the experiment, $\bar{\mathbf{z}}_0$ is initialized as a zero vector, and $\mathbf{P}_0$, $\mathbf{Q}$ and $\mathbf{R}$ are set as multiple of identity matrix with scale values 10^{-5}, 10^{-2} and 10^{-2}, respectively. We set $N_z = 3$ for the dimensionality of the hidden state and $N_x = 185$, which corresponds to the number of stocks for which model evaluates simultaneously one day ahead label prediction as described in Sec. 3.1.2. The transform T and mapping D are randomly initialized at time 0 from a uniform random distribution. The last updated values of transforms is used to initialize for the next processed windows, as explained in Sec. 3.2.1. All

presented results in Sec.3.3.4.2 for (STL_2) are averaged on 50 random trials. Each day, the approach provides the investor with the list of stocks for which the action buy, hold or sell should be performed. The investor can choose where to invest from the list of stocks based on precision, recall, and F1-score metric.

3.2.2.3 Label evaluation

The label identification strategy is same for both STL_1 and STL_2. We apply a tanh activation function for the predicted observed sequence to scale down the weights from -1 to 1. The observed sequence scaling can be seen as $\widehat{x}_{k+\tau+1} = tanh(x_{k+\tau+1})$. The class label for the next day trading decision are calculated by thresholding and will be simply defined as :

$$\ell_{k+\tau+1} = \begin{cases} -1 & \widehat{x}_{k+\tau+1}[i] < -0.5 \\ 0 & -0.5 < \widehat{x}_{k+\tau+1}[i] < 0.5 \\ 1 & \widehat{x}_{k+\tau+1}[i] > 0.5 \end{cases} \tag{3.12}$$

3.2.3 Probabilistic validation

The probabilistic approach in Kalman helps in estimating the uncertainty associated with the prediction. Kalman framework provides the distribution of the next observation conditioned to previous data (the so-called predictive distribution of

the observations), given by

$$p(\mathbf{x}_k|\mathbf{x}_{1:k-1}) = \mathcal{N}\left(\mathbf{x}_k; \mathbf{D}\mathbf{z}_k^-, \mathbf{S}_k\right) \tag{3.13}$$

with $\mathbf{x}_k$ the observation at time k, $\mathbf{S}_k = \mathbf{D}(\mathbf{T}_1\mathbf{P}_{k-1}\mathbf{T}_1^\top + \mathbf{Q}) + \mathbf{R}$, and $\mathbf{z}_k^-$, $\mathbf{P}_k$ are defined in Sec.3.1.2.1 (see [18, Section 4.3] for more details).

In this case, we can quantify our uncertainty about the prediction given by the model for buy, hold and sell label. In other words, the probabilistic validation helps us to quantify how much the model is uncertain when it says to buy, hold or sell the stock. In particular, we can define :

$$\widehat{p}_k = \int_{x_k}^{\infty} \mathcal{N}\left(x_k; \mathbf{D}\mathbf{z}_k^-, \mathbf{S}_k\right) dy = 1 - \text{CDF}(x_k|\mathbf{D}\mathbf{z}_k^-, \mathbf{S}_k), \tag{3.14}$$

as the probability that our forecasting method gives to the stock to whether hold, buy or sell in the next time step. We provide that information, and a knowledgeable practitioner can use the information to trade accordingly. Once we have determined $\mathbf{p}_k$, we can validate the method by using the standard cross-entropy loss as:

$$\text{log-loss} = -\frac{1}{K}\sum_{k=1}^{K}(L[k]\log(p_k[k])), \tag{3.15}$$

where $\mathbf{L} \in \{-1, 0, 1\}$ represents the ground truth labels at time k.

3.3 Experiment and performance evaluation

3.3.1 Curated Dataset

The dataset is same as decribed in Chapter1 section1.3.1. We observe daily values of feature of each stock, namely its close price for calculating the technical indicator value. Technical Indicators are crucial component in determining the future trends of stocks in the market due to the inherent property of providing an early signal for entry and exit. Furthermore, technical indicator SMA has been utilised to understand the directional movement. We use this important information as external control input(u_k) for our model.

3.3.2 Compared methods

We perform comparisons with several methods from the field of machine learning and signal processing, namely CNN-TA [92], MFNN [93], LSTM [57]. We chose these three techniques since they are the most recent state-of-the-art methods published within the last two years. These papers [57, 92, 93] have compared with several other shallow and deep techniques and have excelled over them. Therefore, it is enough to show that we improve over these; we do not need to compare the techniques benchmarked in mentioned papers. The benchmark models train on 10 years of data and test on 10 years of data (2009-19). Although our model works on a windowed fashion and hence can predict for the entire twenty years (sans the first window), we test it only on the common 10 years of test data (2009-19). For training the state of the art methods like CNN-TA,

MFNN and LSTM, following the methodology of [92]. For the sake of fair analysis, we will evaluate results between STL_1, STL_2 and its competitors, only for this 10 years period.

3.3.3 Practical Specification

3.3.3.1 Software and Hardware specifications

Data pre-processing and STL_1 and STL_2 model simulations is wholly conducted in Python 3.6 relying on the libraries of Sklearn, packages of numpy and pandas. For the simulation of benchmark methodologies we used Pytorch libraries which is one of the powerful libraries to carry out large scale machine learning experiments. We also rely on Ta-lib[1] which is a python variant to evaluate technical indicators and Ta4j[2] which is a Java variant for evaluating the technical indicators. All the experiments are carried on using a Dell T30, Xeon E3-1225V5 3.3GHz with GeForce GT 730, and Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz 16GB RAM, 200GB HDD, equipped with an Nvidia 1080 8GB and Ubuntu OS.

3.3.4 Numerical results for the trading model

This section presents the analysis of stock forecasting. We will evaluate the method's ability to distinguish between "hold", "buy" and "sell" for the next day of every window. The external control input (SMA indicator) is normalised

[1] http://ta-lib.org
[2] http://www.ta4j.org

so that we have all the values boiling down to same scale. The work presents two versions of Sequential transform learning as STL_1 and STL_2 as described in section 3.2.2. It is important to note that we carried out experiments with both the approach and compared with state-of-the-art method namely CNN-TA, MFNN and LSTM which is presented in this section 3.3.4.2.

3.3.4.1 Influence of the window size

We will use STL_1 and STL_2 method for carrying out trading simulation (classification) using the model described in Sec. 3.2.2.2. The model can process the entire sequence length, but there are certain drawbacks associated with this offline strategy, as we already discussed in Sec. 3.2.1.

It is important to pre-process the data and section them into windows to update the linear operators as the time progresses regularly. The window size plays an important role as it is essential to analyze the appropriate window size to produce optimal predictions while preserving a reasonable computational time. Table 3.1 and Table. 3.2 depicts the experimental performance of STL_1 and STL_2 respectively for different window sizes. The performance tends to improve as the window size increases since the model incorporates more data. We notice then stabilization of the performance, from $\tau = 10$. We believe that 10 days window size is a good compromise between highly volatile stocks (very non-stationary) and very stable (smooth) ones. This value will be retained in the upcoming experiments. When the window size increases further, the volatility influences (degrades) the piece-wise linear nature of the data, and hence the

performance deteriorates.

Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
5	0.81	0.09	0.10	0.81	0.19	0.20	0.80	0.12	0.14
10	**0.90**	**0.15**	**0.14**	**0.47**	**0.51**	**0.52**	**0.61**	**0.23**	**0.21**
15	0.87	0.14	0.13	0.43	0.50	0.51	0.57	0.21	0.20
20	0.88	0.16	0.15	0.49	0.51	0.50	0.62	0.18	0.23

Table 3.1: Comparison results of online STL_1 for different window size (averaged over 185 stocks in datatset).

Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
5	0.88	0.10	0.10	0.72	0.21	0.23	0.78	0.13	0.14
10	**0.91**	**0.16**	**0.14**	**0.48**	**0.54**	**0.53**	**0.62**	**0.24**	**0.22**
15	0.90	0.16	0.14	0.46	0.51	0.52	0.59	0.24	0.22
20	0.90	0.16	0.14	0.48	0.53	0.52	0.62	0.23	0.22

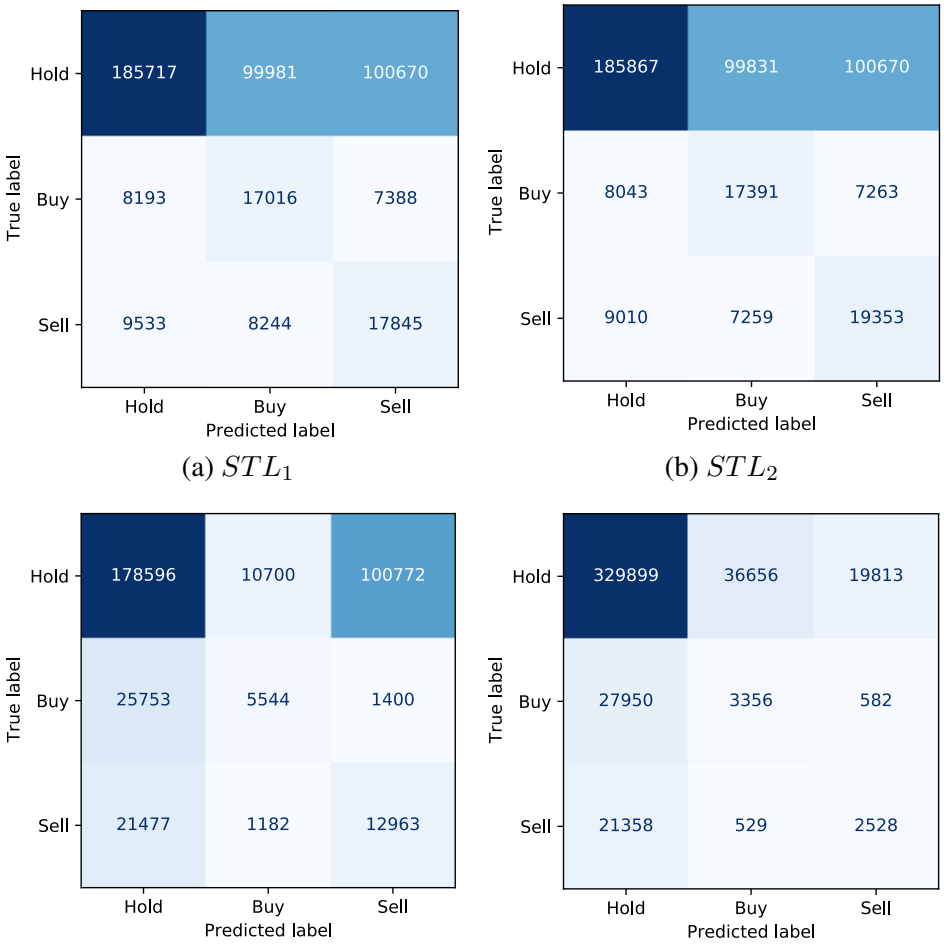
Table 3.2: Comparison results of online STL_2 for different window size (averaged over 185 stocks in datatset).

Method	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
STL_1	0.90	0.15	0.14	0.47	0.51	0.52	0.61	0.23	0.21
STL_2	0.91	0.16	0.14	0.48	0.54	0.53	0.62	0.24	0.22
MFNN	0.79	0.11	0.04	0.47	0.37	0.16	0.58	0.06	0.11
LSTM	0.84	0.07	0.06	0.89	0.05	0.05	0.86	0.05	0.05
CNN-TA	0.84	0.11	0.09	0.85	0.07	0.10	0.85	0.09	0.08

Table 3.3: Comparison of classification metric for different methods in trading problem

3.3.4.2 Classification metrics

The classification performance of the model is summarized in Fig. 3.3, which represents the confusion matrices obtained from STL_1, STL_2 approach, and three state-of-the-art methods. We see that the Hold class is better predicted for all the methods than the two other classes, as the investors use more of the hold signals. They wait for some extraordinarily turning points to out-stand among their competitors. LSTM is getting the best scores over the competitors in terms of false negative. However, LSTM (and to a mild extent, CNN-TA



(a) STL_1 (b) STL_2

(c) MFNN (d) CNN-TA

Figure 3.3: Confusion matrices for trading decision.

and MFNN) present a large number of false positive for the Hold class. Those methods seem to adopt an over-conservative strategy privileging the Hold class over the others. However, STL_1 and STL_2 models are able to preserve a certain balance among classes. This can be seen in the confusion matrices, by noticing that the maximum values are taken in the diagonal, as it should be expected by a valid classifier. The comparison results are presented in Table 3.3 for STL_1, STL_2 and the three competitors with evaluation metrics, namely recall, precision, and F1 Score of the proposed approach against the three deep learning models. STL_2 outperforms STL_1 and the competitors, consistent with the previous

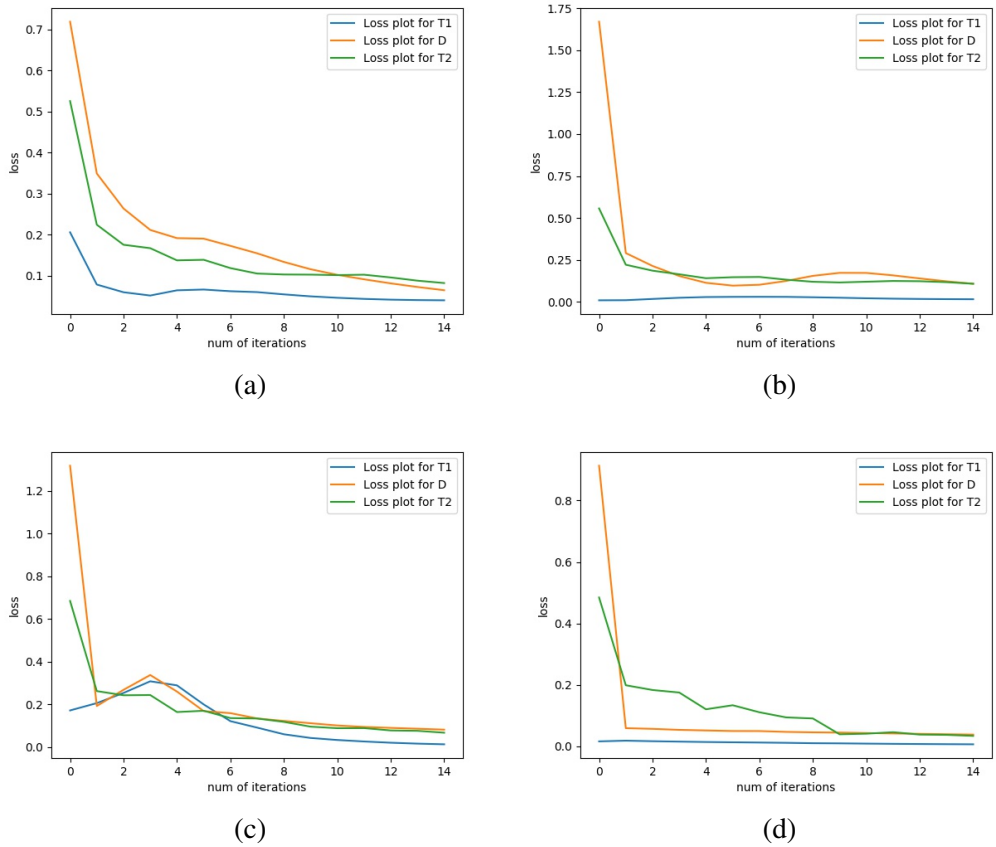


Figure 3.4: Loss plots for the convergence of T1,T2 and D. The X-axis represent the number of iterations and Y-axis represents the loss for each iteration for continous 4 days. The blue color represents the loss trend for T_1 transform, orange represents the loss trend for mapping D and the green represents the loss trends for T_2 transform.

observations. The STL_2 approach is better suited for the market. With STL_2 investor has access to next day label prediction for N stocks simultaneously and has a broader look at the market; hence can capture the trend better than individual stocks. This helps in making well-informed decisions where to invest in the market at every timestamp to maximize profit. The performance of the methods will be evaluated, considering the model's empirical performance, i.e., how well the model can distinguish between the three classes. We will also assess the models' performance in terms of trading efficiency by measuring and comparing their annualized returns based on model predictions. Fig. 3.4

represents the convergence plots for linear operator T_1, T_2 and D.

3.3.4.3 Annualised returns analysis

This is the last phase of our algorithm evaluation, where we used the predicted signal for calculating the returns, which the predicted signals will bring to the investor. Investors consider one more parameter while trading for stocks, ie.. annualized returns (AR). AR is a critical factor as it helps the investors to decide the monetary gain/loss they are going to incur while investing in the stock. If the predicted label is "Sell", the stock is sold at that price. If the predicted signal is "Hold", no action is taken at this signal. The algorithm is explained well in [92] in the financial evaluation section. We present the annualized returns achieved on the test data comparing with the benchmark models and the ones achieved from the proposed approach on nine randomly selected stocks in Table. 3.4, as well as the average AR for all stocks. The best-annualized returns are highlighted in bold. The proposed method's annualised returns are better than what produced by state of the art deep learning methods.

Stock Symbol	STL_1	STL_2	CNN-TA	MFNN	LSTM
WIPRO.BO	-28.4	**-24.56**	-29.14	-27.81	-47.74
AAPL	11.3	**18.1**	0	12.92	0
AMZN	-14.8	-12.7	**30.64**	-20.85	-0.15
IOC.BO	-12.43	**-6.62**	-31.03	-26.42	-31.1
TATACHEM.BO	-1.67	**3.89**	-1.54	-8.32	0
SPICEJET.BO	1.84	**2.36**	-24.08	-28.21	0
ATML	**10.12**	8.02	-33.25	-27.07	-33.82
DOM.L	-3.56	-4.58	0.11	**8.22**	0.47
INDRAMEDCO.BO	2.86	**4.65**	-14.22	-3.53	-50.86
Average(185 stocks)	2.93	**3.56**	-5.65	-1.87	-6.45

Table 3.4: Annualized Returns

3.3.4.4 Portfolio diversification

Diversification of portfolio is a smart way of trading [72] as it allows investors to maximize the returns by investing in different domains that would react differently in the same market events, thus managing the risk better. The market is therefore divided into three major categories ie., small cap stocks, mid cap stocks and large cap stocks. Small-caps are more volatile, vulnerable to losses during market downtime trends but these stocks offer a great opportunity for growth and high returns. Therefore, risk factor associated with small caps are higher. Mid cap stocks offers investors more growth potential than large-cap stocks, but with less volatility and risk as compared to small-caps. Large cap stocks are the ones that offer fixed returns with minimal volatility associated. Large cap stocks are the assets from financially sound firms and has a steady risk-return factor with relatively lower risk compared to mid-cap, small-cap.

	Stock symbols	STL_1 Log-Loss	STL_2 Log-Loss
Small-cap	ALOKTEXT.BO	1.87	**1.76**
	ALKYLAMINE.BO	1.11	**1.09**
	ZEEMEDIA6.BO	1.23	**1.18**
	PVP.BO	**1.01**	1.04
Mid-cap	IOC.BO	**1.02**	1.05
	TATACHEM.BO	**0.78**	0.83
	SPICEJET.BO	0.65	**0.61**
	BHEL.BO	**1.11**	1.14
Large-cap	AAPL	0.13	**0.09**
	AMZN	0.11	**0.07**
	HINDZINC.BO	1.08	**1.02**
	ONGC.BO	**1.01**	1.03
	SIEMENS.NS	1.03	**0.06**

Table 3.5: Log-loss values provided by STL_1 and STL_2, for the stocks with available market capitalization categories.

As discussed earlier in Sec 3.2.3 the proposed STL_1 and STL_2 methodology

allows to provide through probabilistic quantification, a piece of information that can be used by a knowledgeable practitioner to trade accordingly to have a diversified portfolio. Such probabilistic validation can help maximize the returns by having a mix of small-cap, mid-cap, large-cap stocks.

We provide in Table 3.5 the log-loss value (the smaller, the better), computed by both the approaches STL_1 and STL_2 as described in Sec. 3.2.3, computed for the few stocks of the dataset for which the market capitalization categories were available.[3]. The Log-loss validates the uncertainty associated with model prediction. The log-loss value can reach low value when the uncertainty associated with the prediction is very low (ie., good prediction) when trading the large cap stocks, probably due to the less volatile nature of such stocks. In contrast, the log-loss is in average higher for small caps, as they are highly volatile.

3.3.5 Discussions

Our proposed approach combines the benefits of two worlds – the uncertainty measures and interpretability of classical signal processing with the function approximation capability of neural networks. The proposed technique has been applied on the challenging problem of stock forecasting. An interesting experimental analyses was suggested by one of the reviewer. We have created a plot of probabilities versus average recall (Please refer to Fig3.5). We did it for recall, since it is the most important metric in stock trading.

Recall values can also be understood as the quantification of positive class

[3] https://finance.yahoo.com/screener

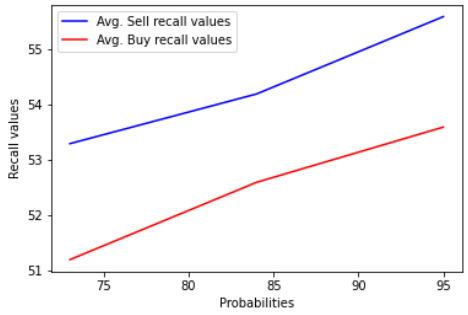


Figure 3.5: Plot of recall values vs probabilities

predictions made out of positive data points.We have seen better recall values for the class " Sell" and "Buy" as compared to class " Hold". In market analysis, it is very important to identify the critical turning points in the market trend, i.e., accurate prediction of entry and exit points. The precise prediction of entry and exit points helps us analyze what the algorithmic model guarantees return. Every algorithmic model's final goal is to maximize the annualized returns with predicted sell and buy signals.

Our proposed algorithm has successfully captured accurate entry and exit points as compared to the benchmark models. The finance Dataset is an imbalanced dataset, due to which we have more number of hold data points and very few critical "Buy and Sell" class datapoints. Most of the neural network models used as a benchmark generated many false alarms for non-existence entry and exit points. In comparison, our model captured most of the entry and exit points intelligently. This can be understood as hold class has more number of samples, neural network labeled most of the data points as hold class at the time of the testing phase. Whereas Sequential transform learning uses the information of technical indicators along with the prior estimate. The combination of technical

indicator information and prior estimate resulted in better modeling of posterior. Sequential Transform learning could handle the imbalanced dataset and resulted in better sensitivity score(Recall) for critical class(Buy and Sell). From Fig3.5 we can say that when the model is becoming more and more certain the recall values is also improving. This shows the importance of uncertainty quantification in challenging application. The uncertainty score makes the investor more confident in investing with the predicted signals.

Chapter 4

Deep Recurrent Dictionary Learning

In this chapter we follow Deep recurrent dictionary learning. Modeling dynamical systems has been a topic of interest to signal processing, machine learning and control engineering researchers for more than five decades. This work is proposed to overcome the bottlenecks experienced in chapter 2 (assuming the model to be linear) and gains motivation in diving deeper in multi-linear ssm to propose more adaptive recurrent approaches in time series forecasting. Recent papers such as [104, 105] are showing how deep neural networks excel over shallow networks in terms of function approximation. They are developed to approximate highly non-linear function in high-dimensional spaces [106, 107]. We propose a deep extension of the shallow model Recurrent dictionary learning (discussed in chapter 2); multilinear Gaussian SSM which handles non-linearity in the time series. The work introduces a new dynamical modeling technique that combines the advantages of state-of-the-art deep learning tools with those of traditional state-space models. The proposed tool is particularly well suited to model stock market time series.

Modeling the stock market is a well-known challenging problem [108]. The difficulty lies in the non-stationary and non-linearity of the underlying dynamical process. Moreover, financial markets are not only influenced by consumer behavior but also by a myriad of external factors like natural disasters, administrative policies, political decisions, international relations, etc., to name a few. Therefore developing reliable algorithmic models for stock trading still remains a challenging yet interesting topic from the point of view of both finance and machine learning/signal processing [3, 109].

4.0.1 Contribution

A partial solution to SSM limitations has been addressed in [2, 110–112]. In these works, a linear Gaussian SSM was considered, and the observation and state operators were unknown and estimated from data using an expectation-maximization (EM) methodology. In our recent work [110], we show the practical advantage of such approach in the context of stock market time series processing. A limitation in the aforementioned works is that they can only consider linear models. There is a crucial need for new strategies to cope with the curse of dimensionality in the learning of SSM model parameters. In order to address this shortcoming, in this work we consider instead multi-linear observation and state operators, to be learnt jointly with the state inferential task. In this work, we propose to impose a structured prior on the observation/state operators involved in an Linear Gaussian state space model (LG-SSM). We introduce deep non-negative matrix factorized (deep NMF) models for both operators. Deep NMF [113] is a generalized form

of NMF [114] that models latent representations from complex data through a product of a (usually small) number of linear operators (called latent factors) satisfying positivity constraints. Deep NMF has been employed with success on various unsupervised machine learning tasks [115–119]. When embedded into an NN structure, it leads to the so-called deep ReLu networks [120–122]. Deep NMF shares connections with the recently introduced deep dictionary learning (deep DL) [123, 124], the main difference being in the priors imposed in the latent factors (positivity, in the case of deep NMF, low-rank/sparsity in the case of deep DL). In our work, deep NMF is neither used for unsupervised representation learning nor in an NN framework. In contrast, it is embedded into a Gaussian SSM to model, allowing to track and predict complex latent phenomena in time series. A novel algorithm is proposed, that learns the positive latent factors jointly with the probabilistic state inferential task induced by our SSM. We call this modelling and inference tool *Deep Recurrent Dictionary Learning* (DRDL). The tool is further specialized to make it practically efficient in the context of large and volatile time series arising in stock market data. In particular, we perused the online training strategy we previously introduced in [110]. The contributions of this work are:

- Introduce an LG-SSM model involving deep positive latent factors;
- Propose a new EM-based inference method to jointly perform the time series prediction task and the deep linear positive factors estimation;
- Devise efficient implementation strategies for practical use of the method in

the context of stock market time series analysis;

- Providing a quantitative assessment of the performance of DRDL approach on real financial data.

4.1 Proposed Work

4.1.1 DRDL model

Let us consider $(\mathbf{x}_k)_{1\leq k\leq K}$, an observed sequence of vectors of size N_x and $(\mathbf{z}_k)_{1\leq k\leq K}$ a sequence of unknown feature vectors of size N_z that we want to infer/estimate. We consider here possibly multivariate state and observation vectors, i.e., N_x and N_z can be greater than 1. Our DRDL model is expressed as the following multi-linear Gaussian SSM[1] .

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_{k-1} + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{H}_0\mathbf{H}_1\mathbf{H}_2\mathbf{z}_k + \mathbf{v}_{2,k}. \end{cases} \tag{4.1}$$

Hereabove, the process noise $(\mathbf{v}_{1,k})_{1\leq k\leq K}$ is assumed to be zero-mean Gaussian with symmetric definite positive covariance matrix $\mathbf{Q}$, and the observation noise $(\mathbf{v}_{2,k})_{1\leq k\leq K}$ is also zero-mean Gaussian with symmetric definite positive covariance matrix $\mathbf{R}$. Then, the model in (4.1) can be seen as a first-order Markovian multi-linear Gaussian model where $(\mathbf{z}_k)_{1\leq k\leq K}$ is the sequence of

[1]Throughout the work, we consider three-terms factorizations, for the sake of readibility. The proposed modeling and inference methodology can actually be straightforwardly extended to any number, greater or equals to one, of factors.

K hidden states. As discussed earlier, classical inference approaches for SSM require specifying every model parameters. In the above model, that would mean setting the observation and state matrices $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$. In practical applications such as stock market analysis, this is challenging and one might prefer to learn these factors from the observed data. In a nutshell, the goal is the joint inference, from the observed sequence $(\mathbf{x}_k)_{1\leq k\leq K}$, of the latent factor matrices $\mathbf{D}_0 \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{D}_1 \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{D}_2 \in \mathbb{R}^{N_z\times N_z}$, of the observation matrices $\mathbf{H}_0 \in \mathbb{R}^{N_x\times N_z}$, $\mathbf{H}_1 \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{H}_2 \in \mathbb{R}^{N_z\times N_z}$ and of the sequence $(\mathbf{z}_k)_{1\leq k\leq K}$.

4.1.2 Discussion on the model

We now discuss the main characteristics of the proposed DRDL method. The model is mathematically described in Eq. (4.1) and displayed in Fig. 4.1. The top equation describes the hidden state evolution, assuming Markovianity between two consecutive hidden states. The second equation links the hidden and observed states. A first interesting aspect, inherited from the SSM paradigm, is that two Gaussian noise terms are explicitly introduced in DRDL to cope with model uncertainty, which is in contrast with most deep learning models for time series processing (e.g., LSTM). A second novel feature of (4.1) lies in using deep NMF models instead of generic matrices (in the linear case) or functions (in the non-linear case), as it is usually the case in SSMs [125–127], taking advantage on the acknowledged representation power of deep NMF [113]. One important benefit of the proposed approach w.r.t. most existing methods in

the literature is that we avoid Monte Carlo simulation or complex optimization procedure, which is known to suffer more severely the curse of dimensionality. In our method, each latent factor can be understood as representations in abstract spaces of the phenomena occurring between both pairs of variables. Third, in contrast with the typical usage of deep NMF in machine learning, relying on backpropagation for their model training [118, 119], DRDL model allows the construction of an handcrafted training strategy (see the next section), which benefits from a low computational cost, sound optimality guarantees (in terms of Bayesian estimator), and enables uncertainty quantification.

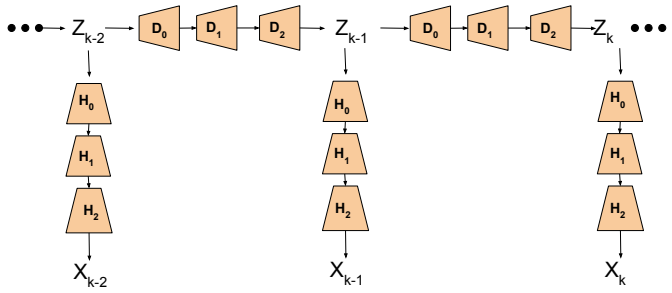


Figure 4.1: Schematic Diagram for Deep Recurrent Dictionary Learning

4.1.3 DRDL inference algorithm

Using SSM models for time series processing (e.g., for a prediction task) amounts to solving the so-called smoothing/filtering problem, i.e., the probabilistic estimation of the hidden state $(\mathbf{z}_k)_{1\leq k\leq K}$. In our context, as the deep NMF factors $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$ involved in the construction of the state transition and the observation transition models are most often unknown, we must also infer them from the observed data, jointly with the hidden states (through the aforementioned filtering/smoothing procedure). To do so, we propose an expectation-

maximization (EM) approach (see [83, chap.12] and [2]).

The EM method alternates iteratively between the probabilistic inference of the state $(\mathbf{z}_k)_{1\leq k\leq K}$, while the factors $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$ are fixed (E-step), and the update of these factors, assuming fixed state (M-step). More precisely, the E-step consists in fixing the linear operators obtained in the previous M-step and applying the classical Kalman/RTS recursions, in order to obtain the filtering/smoothing distributions $p(\mathbf{z}_k|\mathbf{x}_{1:k})$ and $p(\mathbf{z}_k, \mathbf{x}_{1:K})$, respectively. Then, the M-step updates the operators $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$ by maximizing an upper bound of:

$$\varphi_K(\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2) \\ = \log p(\mathbf{x}_{1:K}|\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2). \quad (4.2)$$

We explicitly show hereafter the construction of the $i+1$-th EM update, given estimates from the previous iteration i.

4.1.3.1 E-step: Kalman/RTS inference

At this step, we considered the factors $\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}, \mathbf{H}_0^{[i]}, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}$ to be fixed (either from the previous M-step or from the initialization at the first iteration). The goal is the probabilistic inference of the hidden state. Following the standard SSM framework, we assume that the initial state follows $\mathbf{z}_0 \sim \mathcal{N}(\bar{\mathbf{z}}_0, \mathbf{P}_0)$, where $\mathbf{P}_0$ is a symmetric definite positive matrix of $\mathbb{R}^{N_z \times N_z}$ and $\bar{\mathbf{z}}_0 \in \mathbb{R}$. Then, (4.1) reads as a first-order Markovian linear Gaussian model, with observation matrix

$\mathbf{H}_0\mathbf{H}_1\mathbf{H}_2$, evolution/state matrix $\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2$, and $(\mathbf{z}_k)_{1\leq k\leq K}$ the sequence of K unknown states. Kalman filter provides a probabilistic estimate of the hidden state at each time step $k \in \{1, \ldots, K\}$, conditioned to all available data up to time k, through the Gaussian filtering distribution:

$$p(\mathbf{z}_k|\mathbf{x}_{1:k}) = \mathcal{N}(\mathbf{z}_k; \bar{\mathbf{z}}_k, \mathbf{P}_k). \tag{4.3}$$

For every $k \in \{1, \ldots, K\}$, the mean $\mathbf{z}_k$ and covariance matrix $\mathbf{P}_k$ can be computed by means of Kalman filter recursions given as follows:

For $k = 1, \ldots, K$

Predict state:

$$\begin{cases} \mathbf{z}_{k|k-1} &= \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\bar{\mathbf{z}}_{k-1}, \\ \mathbf{P}_{k|k-1} &= \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{P}_{k-1}(\mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]})^\top + \mathbf{Q}. \end{cases} \tag{4.4}$$

Update state:

$$\begin{cases} \mathbf{y}_k &= \mathbf{x}_k - \mathbf{H}_0^{[i]}\mathbf{H}_1^{[i]}\mathbf{H}_2^{[i]}\mathbf{z}_{k|k-1}, \\ \mathbf{S}_k &= \mathbf{H}_0^{[i]}\mathbf{H}_1^{[i]}\mathbf{H}_2^{[i]}\mathbf{P}_{k|k-1}(\mathbf{H}_0^{[i]}\mathbf{H}_1^{[i]}\mathbf{H}_2^{[i]})^\top + \mathbf{R}, \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}(\mathbf{H}_0^{[i]}\mathbf{H}_1^{[i]}\mathbf{H}_2^{[i]})^\top\mathbf{S}_k^{-1}, \\ \bar{\mathbf{z}}_k &= \mathbf{z}_{k|k-1} + \mathbf{K}_k\mathbf{y}_k, \\ \mathbf{P}_k &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top. \end{cases} \tag{4.5}$$

Hereabove, $\mathbf{y}_k$ represents the measurement pre-fit residual, $\mathbf{S}_k$ represents the pre-fit covariance, $\mathbf{K}_k$ represents Kalman gain, $\bar{\mathbf{z}}_k$ represents the updated (a posteriori) state estimate, $\mathbf{P}_k$ represents the updated (a posteriori) estimate covariance. The RTS smoother makes a backward recursion in order to obtain

the smoothing distribution $p(\mathbf{z}_k|\mathbf{x}_{1\ldots K})$ by refining the filtering distributions computed by the Kalman filter:

For $k = K, \ldots, 1$

Backward Recursion:

$$\begin{cases} \mathbf{z}_{k+1}^{-} & = \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\bar{\mathbf{z}}_k, \\ \mathbf{P}_{k+1}^{-} & = \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{P}_k(\mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]})^\top + \mathbf{Q}, \\ \mathbf{G}_k & = \mathbf{P}_k(\mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]})^\top[\mathbf{P}_{k+1}^{-}]^{-1}, \\ \mathbf{z}_k^s & = \bar{\mathbf{z}}_k + \mathbf{G}_k[\mathbf{z}_{k+1}^s - \mathbf{z}_{k+1}^{-}], \\ \mathbf{P}_k^s & = \mathbf{P}_k - \mathbf{G}_k[\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}]\mathbf{G}_k^\top. \end{cases} \tag{4.6}$$

As a result, the smoothing distribution at each time step $k \in \{1, \ldots, K\}$ has a closed-form Gaussian solution given by:

$$p(\mathbf{z}_k|\mathbf{x}_{1:K}) = \mathcal{N}(\mathbf{z}_k; \mathbf{z}_k^s, \mathbf{P}_k^s). \tag{4.7}$$

4.1.3.2 M-step: Evolution operators update

The M-step performs an optimization step to increase the likelihood of the matrix parameters positive latent factors $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$, given the smoothing distribution obtained in the E-step. To do so, it proceeds by building the upper-bound:

$$\varphi_k(\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2) \geq \mathcal{Q}(\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2; \boldsymbol{\Theta}^{[i]}). \tag{4.8}$$

Hereabove, $\boldsymbol{\Theta}^{[i]} = \{\boldsymbol{\Sigma}^{[i]}, \boldsymbol{\Phi}^{[i]}, \mathbf{B}^{[i]}, \mathbf{C}^{[i]}, \boldsymbol{\Delta}^{[i]}\}$ gathers five quantities defined from the outputs of the E-step described in Sec. 4.1.3.1):

$$\begin{aligned}\mathcal{Q}(\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2; \boldsymbol{\Theta}^{[i]}) = \\ -\frac{K}{2}\mathrm{tr}\Big(\mathbf{Q}^{-1}\boldsymbol{\Sigma}^{[i]} - \mathbf{C}^{[i]}(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2)^\top - \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2(\mathbf{C}^{[i]})^\top \\ +\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\boldsymbol{\Phi}^{[i]}(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2)^\top\Big) \\ -\frac{K}{2}\mathrm{tr}\Big(\mathbf{R}^{-1}\boldsymbol{\Gamma}^{[i]} - \mathbf{B}^{[i]}(\mathbf{H}_0\mathbf{H}_1\mathbf{H}_2)^\top - \mathbf{H}_0\mathbf{H}_1\mathbf{H}_2(\mathbf{B}^{[i]})^\top \\ +\mathbf{H}_0\mathbf{H}_1\mathbf{H}_2\boldsymbol{\Sigma}^{[i]}(\mathbf{H}_0\mathbf{H}_1\mathbf{H}_2)^\top\Big), \quad (4.9)\end{aligned}$$

with:

$$\begin{aligned}\boldsymbol{\Sigma}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_k^s + \mathbf{z}_k^s(\mathbf{z}_k^s)^\top, \\ \boldsymbol{\Phi}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_{k-1}^s + \mathbf{z}_{k-1}^s(\mathbf{z}_{k-1}^s)^\top, \\ \mathbf{B}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k(\mathbf{z}_k^s)^\top, \\ \mathbf{C}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_k^s\mathbf{G}_{k-1}^\top + \mathbf{z}_k^s(\mathbf{z}_{k-1}^s)^\top, \\ \boldsymbol{\Delta}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k\mathbf{x}_k^\top. \end{aligned} \quad (4.10)$$

The updates $\{\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}, \mathbf{H}_0^{[i+1]}, \mathbf{H}_1^{[i+1]}, \mathbf{H}_2^{[i+1]}\}$ given the knowledge of $\boldsymbol{\Theta}^{[i]}$, amounts to maximizing $\mathcal{Q}(\cdot; \boldsymbol{\Theta}^{[i]})$ under positivity constraints on the factors. In contrast with the linear unconstrained model case studied in [83, Chapter 12], the maximization problem here does not have a closed-form solution.

It is highly non-convex due to the multi-linearity of our model. Luckily, it happens to be convex with respect to each of the factors. We thus propose to resort to the following alternating maximization step:

$$\begin{aligned}
\mathbf{D}_0^{[i+1]} &= \underset{\mathbf{D}_0 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}, \mathbf{H}_0^{[i]}, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}) \\
\mathbf{D}_1^{[i+1]} &= \underset{\mathbf{D}_1 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1, \mathbf{D}_2^{[i]}, \mathbf{H}_0^{[i]}, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}) \\
\mathbf{D}_2^{[i+1]} &= \underset{\mathbf{D}_2 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2, \mathbf{H}_0^{[i]}, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}) \\
\mathbf{H}_0^{[i+1]} &= \underset{\mathbf{H}_0 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}, \mathbf{H}_0, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}) \\
\mathbf{H}_1^{[i+1]} &= \underset{\mathbf{H}_1 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}, \mathbf{H}_0^{[i+1]}, \mathbf{H}_1, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}) \\
\mathbf{H}_2^{[i+1]} &= \underset{\mathbf{H}_2 \geq 0}{\operatorname{argmax}} \mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}, \mathbf{H}_0^{[i+1]}, \mathbf{H}_1^{[i+1]}, \mathbf{H}_2; \boldsymbol{\Theta}^{[i]})
\end{aligned}$$

Our approach ensures by construction the following inequality:

$$\begin{aligned}
&\mathcal{Q}(\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}, \mathbf{H}_0^{[i+1]}, \mathbf{H}_1^{[i+1]}, \mathbf{H}_2^{[i+1]}; \boldsymbol{\Theta}^{[i]}) \\
&\qquad\qquad \geq \mathcal{Q}(\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}, \mathbf{H}_0^{[i]}, \mathbf{H}_1^{[i]}, \mathbf{H}_2^{[i]}; \boldsymbol{\Theta}^{[i]}), \quad (4.11)
\end{aligned}$$

which is key to guarantee the convergence properties for the EM iteration. Indeed, the proposed updates yield an increase of the lower bound of the marginal likelihood, so as a consequence, an increase of the marginal log-likelihood itself. The overall procedure is thus guaranteed to yield a monotonic increase of the marginal log-likelihood function φ_K and classical results about majorization-

minimization methods allow to ensure convergence guarantees to a stationary point of φ_K [128]. The six sub-problems are quadratic programming (convex) problems and can be solved through several available solvers. We decided to use the simple and fast alternating least squares approach [114], reminiscent from the literature of deep nonnegative matrix factorization [118], and the deep ReLu neural networks models [121], both showing a satisfactory behavior in preliminary experiments. We start by computing each subproblem solution ignoring the positivity constraints, and then capped the negative entries of the obtained factors. This yields the following analytic updates:

$$
\begin{aligned}
\mathbf{D}_0^{[i+1]} &= \mathrm{ReLu}\left(\mathbf{C}^{[i]}(\mathbf{D}_2^{[i]})^\top(\mathbf{D}_1^{[i]})^\top(\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{\Phi}^{[i]}(\mathbf{D}_2^{[i]})^\top(\mathbf{D}_1^{[i]})^\top)^\dagger\right),\\
\mathbf{D}_1^{[i+1]} &= \mathrm{ReLu}\Big(((\mathbf{D}_0^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{D}_0^{[i+1]})^\dagger(\mathbf{D}_0^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{C}^{[i]}(\mathbf{D}_2^{[i]})^\top\\
&\qquad\times(\mathbf{D}_2^{[i]}\mathbf{\Phi}^{[i]}(\mathbf{D}_2^{[i]})^\top)^\dagger\Big),\\
\mathbf{D}_2^{[i+1]} &= \mathrm{ReLu}\Big(((\mathbf{D}_1^{[i+1]})^\top(\mathbf{D}_0^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{D}_0^{[i+1]}\mathbf{D}_1^{[i+1]})^\dagger(\mathbf{D}_1^{[i+1]})^\top\\
&\qquad\times(\mathbf{D}_0^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{C}^{[i]}(\mathbf{\Phi}^{[i]})^{-1}\Big),\\
\mathbf{H}_0^{[i+1]} &= \mathrm{ReLu}\left(\mathbf{B}^{[i]}(\mathbf{H}_2^{[i]})^\top(\mathbf{H}_1^{[i]})^\top(\mathbf{H}_1^{[i]}\mathbf{H}_2^{[i]}\mathbf{\Sigma}^{[i]}(\mathbf{H}_2^{[i]})^\top(\mathbf{H}_1^{[i]})^\top)^\dagger\right),\\
\mathbf{H}_1^{[i+1]} &= \mathrm{ReLu}\Big(((\mathbf{H}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{H}_0^{[i+1]})^\dagger(\mathbf{H}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{B}^{[i]}(\mathbf{H}_2^{[i]})^\top\\
&\qquad\times(\mathbf{H}_2^{[i]}\mathbf{\Sigma}^{[i]}(\mathbf{H}_2^{[i]})^\top)^\dagger\Big),\\
\mathbf{H}_2^{[i+1]} &= \mathrm{ReLu}\Big(((\mathbf{H}_1^{[i+1]})^\top(\mathbf{H}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{H}_0^{[i+1]}\mathbf{H}_1^{[i+1]})^\dagger\\
&\qquad\times(\mathbf{H}_1^{[i+1]})^\top(\mathbf{H}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{B}^{[i]}(\mathbf{\Sigma}^{[i]})^{-1}\Big).
\end{aligned}
\tag{4.12}
$$

Hereabove, $(\cdot)^\dagger$ denotes the pseudo-inverse operator. Moreover, $\mathrm{ReLu}\,(\cdot)$ states for the rectified linear unit function, that projects each entry of its input to the positive orthant.

4.1.4 The DRDL algorithm summarized

We summarize in Alg. 2 the DRDL algorithm, for the probabilistic inference of the sequence of hidden state $(\mathbf{z}_k)_{1\leq k\leq K}$, jointly with the point-wise estimation of the latent factors $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$, assuming the data follows the DRDL model (4.1). In practice, DRDL algorithm is ran for a maximum number of iterations $i_{\max}$, set so as to reach stabilisation of the latent factors.

Algorithm 2. DRDL (3 layers) inference algorithm.

Inputs. *Prior parameters* $(\overline{\mathbf{z}}_0, \mathbf{P}_0)$ *; model noise covariance matrices* $\mathbf{Q}$, $\mathbf{R}$ *; set of observations* $\{\mathbf{x}_k\}_{1\leq k\leq K}$.

Initialization. *Set positive latent factors*

$\{\mathbf{D}_0^{(0)}, \mathbf{D}_1^{(0)}, \mathbf{D}_2^{(0)}, \mathbf{H}_0^{(0)}, \mathbf{H}_1^{(0)}, \mathbf{H}_2^{(0)}\}$.

Recursive step. *For* $i = 0, 1, \ldots, i_{\max}$*:*

(E step) *Run the Kalman filter* (4.4)-(4.5) *and RTS smoother* (4.6) *using latent factors* $\{\mathbf{D}_0^{(i)}, \mathbf{D}_1^{(i)}, \mathbf{D}_2^{(i)}, \mathbf{H}_0^{(i)}, \mathbf{H}_1^{(i)}, \mathbf{H}_2^{(i)}\}$.
Calculate $(\mathbf{\Sigma}^{(i)}, \mathbf{\Phi}^{(i)}, \mathbf{B}^{(i)}, \mathbf{C}^{(i)}, \mathbf{\Delta}^{(i)})$ *using* (4.10).

(M step) *Compute* $\{\mathbf{D}_0^{(i+1)}, \mathbf{D}_1^{(i+1)}, \mathbf{D}_2^{(i+1)}, \mathbf{H}_0^{(i+1)}, \mathbf{H}_1^{(i+1)}, \mathbf{H}_2^{(i+1)}\}$ *using* (4.12).

Output. *State filtering/smoothing pdfs* (4.3) *and* (4.7) *along with pointwise estimates of the latent factor from* (4.12).

4.2 Application to Stock Trading

We now particularize the DRDL inference algorithm to the stock trading applications. In particular, we address the forecasting/trading tasks given a set of K

daily (i.e., k is a day index) observations of stock market data.

4.2.1 Online implementation

First, in order to better cope with high volatility of stock market quantities and allow immediate feedback to the users for on-the-fly trading, we propose here an online implementation of our DRDL approach. To do so, we adopt a simple yet efficient sliding window strategy. More precisely, we set a window size of $\tau \in \{1, \ldots, K\}$ time steps. For every $k \in \{0, \ldots, K-\tau\}$, the model parameters are estimated using the τ observations contained in the set $\mathcal{X}_k = \{\mathbf{x}_j\}_{j=k+1}^{k+\tau}$ via the EM strategy described above. Such sliding window strategy provides two advantages. First, choosing τ number of data points in one window allows for a faster processing. Second, it also allows for a better modeling, since the constant factors assumption is likely to better model the time series if τ is small. The price to pay if τ is too small is that a smaller number of observations may limit the estimation capabilities. Hence, it is essential to find a tradeoff in the seek of an optimal τ, as we will show in our experiments. For the implementation of the windowing strategy, we use a warm start strategy for the Kalman iteration initialization. More precisely, we set the factors to their most recent updated value, and we initialize the mean and covariance of the state for processing $\mathcal{X}_{k+1}$ using the EM results from $\mathcal{X}_k$. Note that if $\tau = K$, the algorithm goes back to the original offline version, that is the EM inference tool is ran only once.

4.2.2 Modeling and post-processing for stock market analysis tasks

Stock market data processing typically amounts to solving two distinct applicative problems, namely daily stock price forecasting and stock trading decision (among 3 options: buy/hold/sell) estimation. We hereafter explain how to post-process DRDL results to tackle both above-stated problems.

4.2.2.1 Stock forecasting

Let us first specify the observation model in stock forecasting. For a given window size $\tau > 0$, for each $k \in \{0, \ldots, K-\tau\}$, we observe $(\mathbf{x}_j)_{k+1 \leq j \leq k+\tau} \in \mathbb{R}^{15}$, gathering 14 technical indicators [2] as well as the adjusted close price. Running our DRDL on the considered window yields the following mean estimate of the 15 quantities for the next day (i.e., the day coming right after the end of the observed window) indexed by $k+\tau+1$:

$$\hat{\mathbf{x}}_{k+\tau+1} = \mathbf{H}_0\mathbf{H}_1\mathbf{H}_2\mathbf{z}_{k+\tau|k+\tau-1}, \tag{4.13}$$

associated with the covariance matrix

$$\mathbf{S}_{k+\tau+1} = \mathbf{H}_0\mathbf{H}_1\mathbf{H}_2(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{P}_{k+\tau}(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2)^\top + \mathbf{Q}) + \mathbf{R}. \tag{4.14}$$

Hereabove, $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$ are the factors estimated during the M-step of our EM-based inference method, and $(\mathbf{z}_{k+\tau|k+\tau-1}, \mathbf{P}_{k+\tau})$ are byproducts

[2] We retained the relative strength index (RSI), the William percentage range, the absolute price oscillator (APO), the commodity channel index, the Chande momentum oscillator (CMO), the directional movement Indicator (DMI), the ultimator oscillator, the WMA, the exponential moving average (EMA), the Simple Moving Average (SMA), the triple EMA, the moving average convergence (MAC), the percentage price oscillator, the rate of change (ROC). Detailed definitions can be found in https://www.investopedia.com/terms/t/technicalindicator.asp

of the Kalman prediction step (4.4)-(4.5), computed during the E-step of the EM. The proposed methodology aims at predicting the entire 15-dimensional vector. However, stock forecasting is typically focused on the prediction of a single quantity such as the adjusted close price.

4.2.2.2 Stock trading

Stock trading works on a different set of observed inputs. For each window index $k \in \{0, \ldots, K-\tau\}$, we observe $(\mathbf{x}_j)_{k+1\leq j\leq k+\tau} \in \mathbb{R}^{17}$, where $x_j[i]$, for $i \in \{1, \ldots, 14\}$, are the same 14 technical indicators as in the previous case. Moreover, $[x_j[15], x_j[16], x_j[17]] \in \{0,1\}^3$ gathers the decisions "hold", "buy", or "sell", computed daily for each stocks so as to maximize annualized returns. Soft hot encoded scores are used to represent the retained label, e.g., if the decision "hold" is retained for instance at time j, we observe $x_j[15] = 1$, $x_j[16] = 0$ and $x_j[17] = 0$. Here again, our method is able to provide mean and covariance estimates, for the next day, for the 17 dimensional vector, following (4.13) and (4.14). The class label for the next day trading decision is simply defined as:

$$\ell_{k+\tau+1} = \text{argmax}_{i\in\{1,2,3\}}\, \widehat{x}_{k+\tau+1}[i+14]. \tag{4.15}$$

4.2.3 Probabilistic assessment of stock trading decision

We now describe the procedure to assess the uncertainty quantification associated to the DRDL predictions. Let $k \in \{0, \ldots, K-\tau\}$ be the window index on

which Algorithm 2 has been run. The probabilistic estimation of the quantities of interest for the next time step (i.e., one-day ahead prediction) $\mathbf{x}_{k+\tau+1}$ conditioned to the data observed in the window $\mathbf{x}_{k:k+\tau}$, reads as a multivariate Gaussian distribution

$$p(\mathbf{x}_{k+\tau+1}|\mathbf{x}_{k:k+\tau}) = \mathcal{N}\left(\mathbf{x}_{k+\tau+1}; \hat{\mathbf{x}}_{k+\tau+1}, \mathbf{S}_{k+\tau+1}\right), \tag{4.16}$$

with mean and covariance $(\hat{\mathbf{x}}_{k+\tau+1}, \mathbf{S}_{k+\tau+1})$, given by (4.13) and (4.14), respectively. Equation (4.16) assigns a probability score to any decision (e.g., trading) based on the prediction output of the DRDL method. Let us focus on the particular example of assessing the uncertainty of the stock trading decision at time index $k+\tau+1$, given observations at indexes $j \in \{k+1, \ldots, k+\tau\}$. The trading decision relies on the discrete maximization step (4.15). Let us express the probability mass function (pmf) of this decision, from the gaussian predictive probability density function (pdf) of the observed data points in Eq. (4.16). The pmf can here be summarized as $\mathbf{p}_{k+\tau+1} \in [0,1]^3$ where each $p_{k+\tau+1}[i]$, $i \in \{1,2,3\}$ is a probability, and $\sum_{i=1}^{3} p_{k+\tau+1}[i] = 1$. Each $p_{k+\tau+1}[i]$ represents the probability inferred by DRDL that the true value $x_{k+\tau+1}[i+14]$ is greater than $x_{k+\tau+1}[j+14]$, for $j = \{1,2,3\} \setminus i$. According to Eqs. (4.16) and (4.15), $\mathbf{p}_{k+\tau+1}$ can be obtained through

$$(\forall i \in \{1,2,3\}) \quad p_{k+\tau+1}[i] = \int_{\mathcal{Y}_i} \mathcal{N}(\mathbf{y}; \hat{\mathbf{x}}_{k+\tau+1}[15:17], \mathbf{S}_{k+\tau+1}[15:17, 15:17]) d\mathbf{y}, \tag{4.17}$$

with

$$\mathcal{Y}_i = \left\{ \mathbf{y} \in \mathbb{R}^3 \mid \mathbf{y}[i] \geq \mathbf{y}[j], j = \{1,2,3\} \setminus i \right\}. \tag{4.18}$$

Due to the intricate form of the constrained set in (4.18), the integral in (4.17) is intractable. It can be easily approximated with high precision by direct simulation. In practice, we sampled 10^4 three-dimensional sample from a normal standard distribution. The samples can be re-used for all time steps using coloring and shifting according to the covariance and mean, respectively. Thanks to this procedure, we can infer $\mathbf{p}_{k+\tau+1}$ for every k, and then assess the next day stock trading outcome by using the standard cross-entropy loss:

$$\text{log-loss} = \frac{1}{K-\tau+1} \sum_{k=0}^{K-\tau} \sum_{i=1}^{3} -(L_{k+\tau+1}[i] \log(p_{k+\tau+1}[i])), \tag{4.19}$$

where the true labels are denoted $\mathbf{L}_{k+\tau+1} \in \{0,1\}^3$ for each time $k+\tau+1$ (hereagain, we use soft hot encoding representation).

4.2.4 Summarized pipeline

We provide in Alg. 3 the summary of our proposed pipeline for applying DRDL, in Algorithm 2, in the context of stock forecasting (steps a-b with $N_x = 15$) and trading (steps a-b-c-d with $N_x = 17$).

Algorithm 3. DRDL (3 layers) method for stock forecasting and trading.

Inputs. *Prior parameters* $(\overline{\mathbf{z}}_0, \mathbf{P}_0)$ *; model noise covariance matrices* $\mathbf{Q}$, $\mathbf{R}$ *; set of observations* $\{\mathbf{x}_k\}_{1\leq k\leq K}$ *; windows size* τ.

Initialization. *Set positive latent factors*

$\{\mathbf{D}_0^{(0)}, \mathbf{D}_1^{(0)}, \mathbf{D}_2^{(0)}, \mathbf{H}_0^{(0)}, \mathbf{H}_1^{(0)}, \mathbf{H}_2^{(0)}\}$.

Window processing. *For* $k = 0, 1, \ldots, K-\tau$*:*

a. *Run DRDL algorithm 2 on sequence* $(\mathbf{x}_j)_{k+1\leq k\leq k+\tau}$*, initialized with estimates from* $k-1$*th window (warm start).*
b. *Calculate one-step ahead predicted mean* $\widehat{\mathbf{x}}_{k+\tau+1}$ *and its covariance* $\mathbf{S}_{k+\tau+1}$ *using* (4.13)-(4.14).
c. *Compute one-step ahead predicted label* $\ell_{k+\tau+1}$ *using* (4.15).
d. *Compute* $\mathbf{p}_{k+\tau+1}$ *using* (4.17)-(4.18).

Output. *Forecasting/trading predictions and log-loss value* (4.19).

4.3 Experimental Results

4.3.1 Curated Dataset

The Dataset is same as decribed in Chapter1 section1.3.1. From the knowledge of the close prices, we build two observation sequences associated to the resolution of two specific problems, namely stock forecasting and stock trading, as described in Sec. 4.2.2. Note that the 14 technical indicator values are normalized so that their values range within the same scale. For both problems, we will compare DRDL and several state-of-the-art methods arising from signal processing and machine learning literature. In all experiments, each of the 180 observed time series is split into two parts, namely a train phase made of the first recorded 2546 days, and a test phase made of the next 2882 days. The train phase is used to learn the models parameters (for instance, the linear factors

involved in DRDL), while the test phase is used to evaluate the performance of the learnt models, their parameters being fixed. More details about DRDL and the retained benchmark methods setting are provided in the next subsection. Figure 4.2 displays the evolution of 4 of the 14 technical indicators used as input of the inference tools, during the test phase. One can notice the high volatility in the observed data.

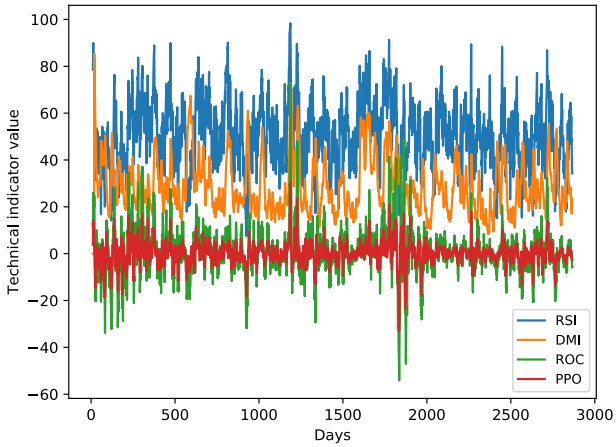


Figure 4.2: Evolution of four (among 14) observed technical indicators during the test phase.

4.3.2 Practical Settings

4.3.2.1 DRDL settings

As described in Sec. 4.2.2, DRDL can be specified to tackle both stock forecasting problem, in which case $N_x = 15$, and stock trading problem where $N_x = 17$. Three variants of DRDL will be compared, depending on the number of linear factors (i.e., layers) in the multi-linear model. More specifically, we will distinguish in our experiments:

DRDL (1 layer): $\mathbf{D}_0 = \mathbf{D}_1 = \mathbf{D}_2 = \mathbf{Id}$ and $\mathbf{H}_1 = \mathbf{H}_2 = \mathbf{Id}$ fixed and $\{\mathbf{H}_0\}$ is estimated;

DRDL (2 layers): $\mathbf{D}_0 = \mathbf{D}_1 = \mathbf{D}_2 = \mathbf{Id}$ and $\mathbf{H}_2 = \mathbf{Id}$ fixed and $\{\mathbf{H}_0, \mathbf{H}_1\}$ are estimated;

DRDL (3 layers): $\mathbf{D}_0 = \mathbf{D}_1 = \mathbf{D}_2 = \mathbf{Id}$ and $\{\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2\}$ are estimated.

Note that, ignoring the positivity constraint, DRDL (1 layer) would identify with our previously published method RDL [110]. We implement the sliding window approach described in Sec. 4.2.1, for various choices of τ described hereafter. In all experiments, the prior mean $\bar{\mathbf{z}}_0$ is initialized as a zero vector, and $\mathbf{P}_0$, $\mathbf{Q}$ and $\mathbf{R}$ are set as multiple of identity matrix with scale values 10^{-7}, 10^{-2} and 10^{-2}, respectively. Moreover, we set the dimension of the state as $N_z = 14$, which also corresponds to the number of measured technical indicators, as it was observed empirically to yield the best results. The entries of the linear factors to estimate are initialized at time 0 using independent realizations of a uniform distribution on $[0, 10^{-1}]$. As mentioned in Sec. 4.2.1, warm start strategy is used to initialize for the next processed windows. The estimation of the linear factors (i.e. M-step of the EM method) is only conducted during the training phase. A maximum of 50 iterations of the EM loop are used in Alg. 2, which was observed sufficient to reach stability of the estimated factors. During the test phase, the linear latent factors are fixed, and only the Kalman/RTS inference is ran (i.e., we inhibit M-step in Alg. 2). The scores shown are arithmetic means of 10 random trials, and are computed only during the test phase.

4.3.2.2 Software and hardware specifications

Dataset preparation and DRDL model inference are ran using Python 3.6 code, equipped with the Sklearn, NumPy packages, and pandas libraries. CNN-TA is implemented in Keras, while MFNN and LSTM, are implemented in PyTorch. We rely on Sklearn, Ta-lib[3] and Ta4j[4] libraries for evaluating the technical indicators. All the experiments are carried on using a Dell T30, Xeon E3-1225V5 3.3GHz with GeForce GT 730, and Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz 16GB RAM, 200GB HDD, equipped with an Nvidia 1080 8GB and Ubuntu OS.

4.3.3 Compared methods

We compare DRDL with state of the art methods from the field of machine learning and signal processing namely CNN-TA [129], MFNN [93], LSTM [57] and ARIMA [10], depending on the problem at stake. For the stock forecasting problem, we compare DRDL with both ARIMA and LSTM. We set up the ARIMA parameters to $(p, d, q) = (5, 1, 5)$ as we observed it to lead to the best practical performance. LSTM has been modified from its original version to perform regression instead of classification, by replacing the softmax output layer by fully connected layer with a single node output. We used Adam optimizer with learning rate 10^{-4}, 200 epochs, and batch-size 16 to minimize the mean square error (MSE) loss function. The performance of the methods is compared in

[3] http://ta-lib.org
[4] http://www.ta4j.org

terms of mean absolute error (MAE), root means square error (RMSE), SMAPE (Symmetric mean absolute percentage error), and Pearson correlation factor on the prediction of the next day adjusted close price, evaluated in the test phase only and averaged over all stocks. For the stock trading problem, we compared DRDL with CNN-TA, MFNN, and LSTM, using their original implementations and settings, described respectively in [129], [93] and [57]. The performance of the methods is evaluated considering both the empirical classification performance of the models, and their trading efficiency in terms of annualized returns. On this problem, we additionnaly present the log-loss values provided by DRDL (see Sec. 4.2.3) to illustrate how such information can be used to let the trader decide where to invest in the market and diversify his portfolio.

4.3.4 Numerical results for stock forecasting problem

Window size τ	r	RMSE $\downarrow$	MAE (%) $\downarrow$	SMAPE (%) $\downarrow$
250	0.45	29.43	0.47	31.8
300	0.49	27.81	0.23	28.6
350	0.53	21.61	0.29	25.5
500	0.69	13.79	0.13	23.4
650	**0.71**	**13.35**	**0.11**	**18.4**
700	0.72	13.62	0.10	18.5

Table 4.1: Results of DRDL (3 layers) on stock forecasting problem for different window size. Scores averaged on test phase, on all the 180 stocks.

4.3.4.1 Influence of window size

The choice of window size is an essential aspect as it can enhance and limit the methodology's potential. To understand the model behavior, we present Table 4.1 which provides detailed information on the performance of the model on varying window sizes. The table offers an analysis of various metrics like

Model	r	RMSE↓	MAE(%) ↓	SMAPE(%)↓
ARIMA	0.13	78.6	1.23	65.5
LSTM	0.24	297.5	6.12	47
DRDL (1 layer)	0.65	23.24	0.19	35.3
DRDL (2 layers)	0.69	14.2	0.15	23.2
DRDL (3 layers)	**0.71**	**13.35**	**0.11**	**18.4**

Table 4.2: Comparison of methods for stock forecasting problem: Pearson correlation score (r), MAE, RMSE and SMAPE scores on the prediction of next day adjusted close price, averaged over the test phase and the stocks.

.

Pearson correlation (r), RMSE (Root Mean Square Error), MAE (Mean Absolute Error), and SMAPE (Symmetric mean absolute percentage error) for different window sizes τ. To understand how window size influences the model behavior in Online implementation (see Sec. 4.2.1). The model's performance improves as the window size increases until reaching a convergence point. Although $\tau = 250$, 300, or 350 may not be sufficient to learn the model appropriately, we can observe that the forecasting results start getting satisfying from $\tau = 500$. In the further experiments on forecasting, we set $\tau = 650$, leading to a good compromise between time complexity and prediction accuracy.

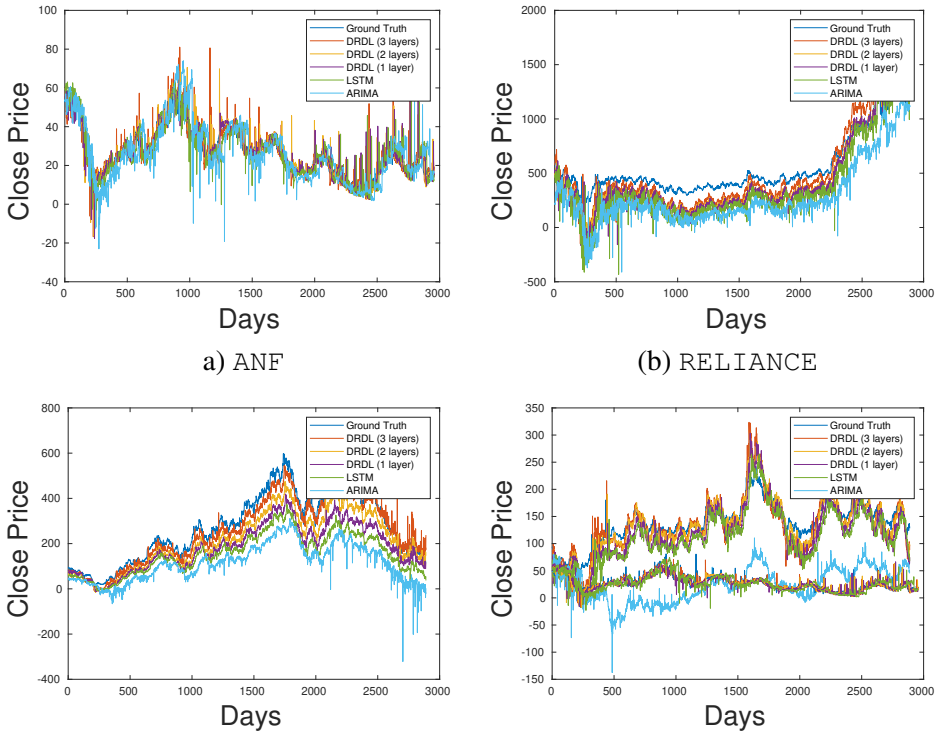
Method	Train Time cost (h.)	Test Time cost (min.)
DRDL (3 layers)	2.37h	20 min
DRDL (2 layers)	2.12h	18.4 min
DRDL (1 layer)	1.78h	15.8 min
ARIMA	2.01h	36 min
LSTM	8 days	45 min
DeepAR	2.45h	20 min
TFT	2.25h	27 min
Nbeats	3.12h	25 min

Table 4.3: Averaged time over 10 random runs for processing the dataset (train(hrs) and test(min)), for DRDL and its competitors.

4.3.4.2 Comparison with benchmark models

To understand better, we present table 4.2 which provides comprehensive analysis on performance estimation on the stock forecasting problem using DRDL, LSTM,

ARIMA, DeepAR [130], Nbeats [131], and TFT [132]. Table 4.2 presents comparison in terms of Pearson correlation factor r, RMSE, MAE and SMAPE. We can see that DRDL (3 layers) architecture outperforms DRDL (2 layers), DRDL (1 layer) as well as the other benchmarks models. We also notice that the average performance of TFT is comparable to DRDL (1 layer) architecture. Fig 4.4 displays the Pearson correlation analysis between ground truth daily adjusted close price time series and predicted ones along test phase, using DRDL (3 layers) for four representive stock cases.

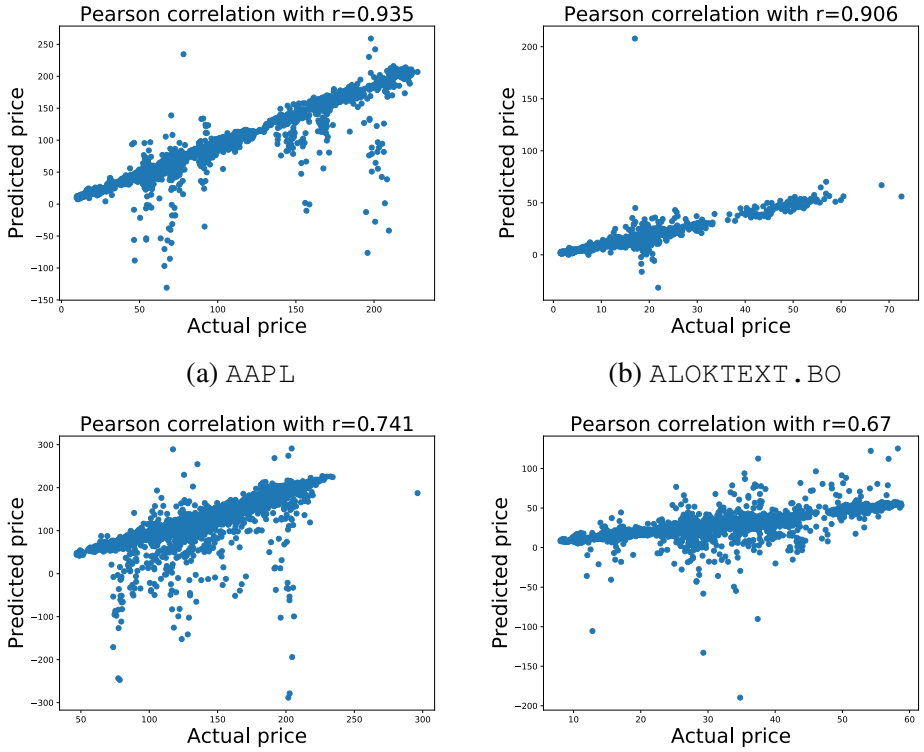


a) ANF (b) RELIANCE

(c) TATAMOTORS (d) ONGC

Figure 4.3: Ground truth and inferred adjusted close price on the test phase for four different stocks, using DRDL with 1 to 3 layers, LSTM or ARIMA.

Table 4.3 presents the computational time for forecasting the next day closing price for our dataset. We distinguish the time required to train the methods (on the first ten years) and to test them (on the next ten years) using the walk-



(a) AAPL (b) ALOKTEXT.BO

(c) BHEL.BO (d) ANF

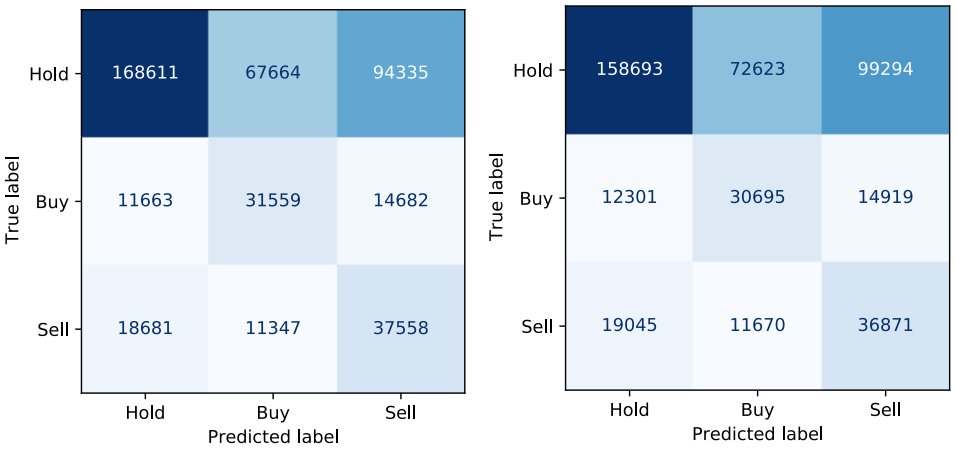
Figure 4.4: Pearson correlation graph between ground truth adjusted close price and predicted one with DRDL (3 layers), during test phase, for four different stocks.

Method	F1 Score			Precision			Recall			Training time (in hrs)	Testing Time (in min)
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy		
DRDL (3 layers)	**0.61**	**0.30**	**0.29**	**0.85**	**0.26**	**0.29**	**0.51**	**0.54**	**0.54**	4.12	10.17
DRDL (2 layers)	0.61	0.32	0.34	0.83	0.23	0.26	0.48	0.51	0.53	3.56	12.50
DRDL (1 layer)	0.59	0.19	0.23	0.88	0.15	0.12	0.45	0.52	0.51	2.00	11.56
MFNN	0.58	0.11	0.06	0.79	0.11	0.04	0.47	0.37	0.16	5.34	14.23
LSTM	0.86	0.05	0.05	0.84	0.07	0.06	0.89	0.05	0.05	12.53	13.50
CNN-TA	0.85	0.08	0.09	0.84	0.11	0.09	0.85	0.07	0.10	4.57	14.36

Table 4.4: Comparison of classification scores of different methods on the stock trading problem. All scores are averaged over 180 stocks and over the days of the test phase.

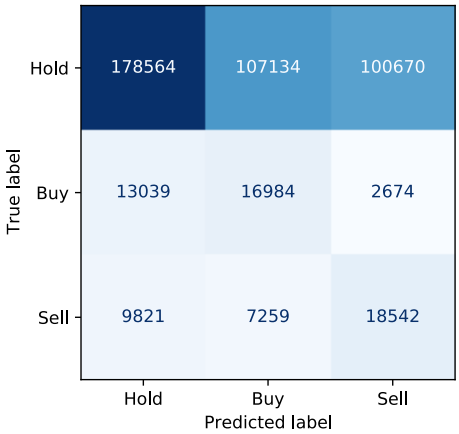
Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
250	0.91	0.15	0.12	0.46	0.51	0.50	0.61	0.23	0.19
300	0.91	0.15	0.12	0.46	0.50	0.51	0.61	0.23	0.19
350	0.89	0.18	0.19	0.47	0.53	0.52	0.61	0.26	0.27
500	0.89	0.18	0.19	0.47	0.52	0.53	0.61	0.26	0.27
650	**0.85**	**0.26**	**0.29**	**0.51**	**0.54**	**0.54**	**0.61**	**0.30**	**0.29**
700	0.87	0.21	0.20	0.48	0.53	0.54	0.61	0.30	0.29

Table 4.5: Classification scores of DRDL (3 layers) for different window size. All scores are averaged over stocks and over the days of the test phase.

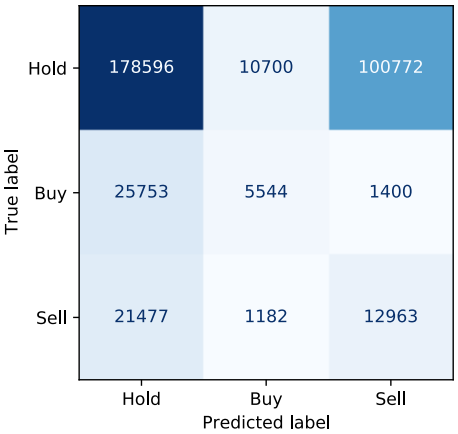


(a) DRDL (3 layers)

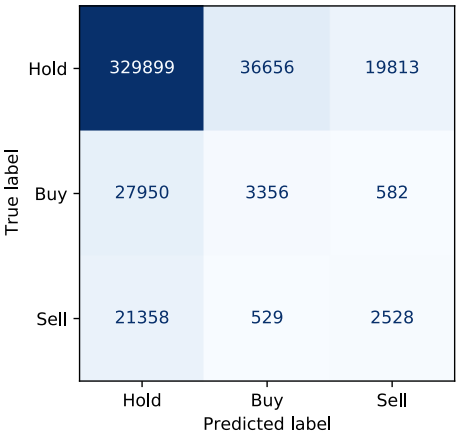
(b) DRDL (2 layers)



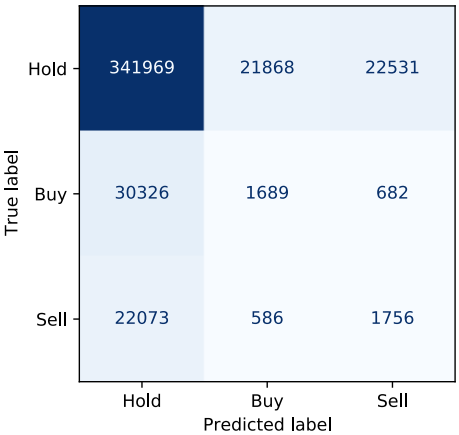
(c) DRDL (1 layer)



(d) MFNN



(e) CNN-TA



(f) LSTM

Figure 4.5: Confusion matrices on stock trading classification task (averaged over days in test phase and over stocks) for DRDL with 1 to 3 layers, and state-of-the art methods MFNN, CNN-TA and LSTM.

Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
250	0.91	0.14	0.12	0.47	0.49	0.48	0.61	0.21	0.19
300	0. 90	0.16	0.13	0.47	0.50	0.49	0.61	0.24	0.20
350	0.89	0.17	0.16	0.48	0.51	0.52	0.62	0.25	0.24
500	0.89	0.17	0.16	0.48	0.52	0.51	0.62	0.25	0.24
650	**0.83**	**0.23**	**0.26**	**0.48**	**0.51**	**0.53**	**0.61**	**0.32**	**0.34**
700	0.84	0.20	0.23	0.46	0.53	0.51	0.59	0.33	0.32

Table 4.6: Classification scores of DRDL (2 layers) for different window size. All scores are averaged over stocks and over the days of the test phase.

Window τ	Precision			Recall			F1 Score		
	Hold	Sell	Buy	Hold	Sell	Buy	Hold	Sell	Buy
250	0.85	0.10	0.10	0.85	0.12	0.12	0.84	0.10	0.10
300	0.85	0.10	0.10	0.74	0.17	0.14	0.80	0.10	0.12
350	0.82	0.10	0.10	0.62	0.30	0.31	0.68	0.15	0.15
500	0.86	0.15	0.14	0.46	0.51	0.51	0.59	0.18	0.22
650	**0.88**	**0.15**	**0.12**	**0.45**	**0.52**	**0.51**	**0.59**	**0.19**	**0.23**
700	0.90	0.16	0.14	0.46	0.51	0.52	0.59	0.24	0.22

Table 4.7: Classification scores of DRDL (1 layer) for different window size. All scores are averaged over stocks and over the days of the test phase.

Stock symbols	DRDL (3 Layers)	DRDL (2 Layers)	DRDL (1 Layer)	CNN-TA	MFNN	LSTM
WIPRO.BO	**-13.89**	-23.26	-29.14	-18.14	-27.81	-47.74
AAPL	**19.12**	11.3	10.14	0	12.92	0
AMZN	-13.23	-11.92	21.23	**30.64**	-20.85	-0.15
IOC.BO	-13.48	-23.28	**-2.68**	-3.03	-26.42	-3.1
TATACHEM.BO	1.23	**3.83**	2.19	-1.54	-8.32	0
SPICEJET.BO	**11.92**	10.17	-8.63	-24.08	-28.21	0
ATML	**-4.13**	-5.78	-10.19	-33.25	-27.07	-33.82
DOM.L	4.56	**9.34**	2.83	0.11	8.22	0.47
INDRAMEDCO.BO	-5.78	-10.34	-3.65	-14.22	**-3.53**	-50.86
Average on all 180 stocks	**3.87**	2.67	2.34	-5.08	-11.45	-13.02

Table 4.8: Annualized returns resulting from the stock trading decisions of different methods during the test phase.

forward method described in [110, Section 4.2.1]. We observed the highest computational time with the LSTM approach. The other methods have rather similar computational time, DRDL (1 layer) being the fastest. The computational time of DRDL (3 layers), reaching the best performance metrics, stays reasonable, and is comparable with the one of DeepAR and TFT. Here, we must recall that,

in contrast with most of its competitors (except ARIMA), our implementation of DRDL method (for both train/test phases) does not exploit GPU facilities such as PyTorch. Complexity reductions could certainly occur if this was the case.

The stock forecasting results are presented in Fig. 4.3. The comparison is carried out between the proposed DRDL for different layer number, LSTM, ARIMA, DeepAR, TFT, Nbeats method. We observed that in some cases (c-d), LSTM approach failed to reach satisfying results which might be due to vanishing gradient issues. In cases (a-b), ARIMA performs quite good when compared to its performance in other cases (c-d). In contrast, DRDL (3 layers) reaches stable and satisfactory outcomes. DRDL (2 layers) outperforms DRDL (1 layer) and both benchmark methods but remains lower quality than its 3-layers variant.

4.3.5 Numerical results for the stock trading problem

4.3.5.1 Influence of the window size

The window size plays an important role as it is essential to analyze the apt window size to produce optimal predictions while preserving a reasonable computational time. Table 4.5 depicts the experimental performance of DRDL (3 layers) architecture, Table 4.6 depicts the experimental performance of DRDL (2 layers) architecture, Table 4.7 depicts the experimental performance of DRDL (1 layer) architecture for different window sizes. The performance tends to improve as the window size increases since the model incorporates more data. We then

notice a stabilization of the performance, from $\tau = 650$. This value is being retained in the upcoming experiments.

4.3.5.2 Classification metrics

To explain the empirical analyses of the trading process (classification), we present confusion matrices. The trading process involves classifying the signal into three classes, namely "Buy," "hold," and "sell" classes. The summarized performance for 180 stocks by DRDL and other state-of-the-art methods is presented in Fig. 4.5. Among the three classes, we see the prediction of hold class is captured efficiently when compared to the other classes. The LSTM approach predicts the best score over the other state-of-the-art methods when compared to false negatives scores. However, in LSTM and CNN-TA approaches are highlighted many false positives for the "hold" class. It can be noted that these deep learning techniques have labeled most signals as hold class, jeopardizing the model behavior for the other classes ("buy," "sell"). The nature of the finance market is highly volatile and non-linear; hence we get to see a highly imbalanced dataset. However, we noticed that the DRDL approach handles it by imposing an activation function on the operators. These operators are expected to evolute continuously as we grow deeper with time sequence. The Table. 4.4 and Fig. 4.5 adds more weight to the analysis. The results state that the DRDL approach managed to predict the highly unbalanced data. The sensitivity score (Recall) is presented well by the DRDL approach compared to the state-of-the-art methods. The diagonals of the confusion matrix of the DRDL

approach also takes the maximum values, which a valid classifier should expect. When dealing with highly imbalanced dataset such as finance dataset, it is more important to study classification metrics like F1 Score, Precision and Recall for each class. This helps in analyzing the model behavior for each class. The Table 4.4 presents analysis of these classification metrics for DRDL compared to other deep learning state-of-the-art methods. We conclude that DRDL outperforms the other methods stated.

In Table 4.4, we also present the computational times (train and test) for conducting trading simulations for our dataset. Hereagain, LSTM is the more demanding method at training. All methods have rather comparable test times, despite DRDL is implemented on CPU only. In particular, besides its probabilistic output, DRDL is not more costly than its competitors. Adding more layers to DRDL slightly increases its train time, but does not affect much the test time.

4.3.5.3 Annualized Returns

Stock market aims to analyze and evaluate the return on investment for a given stock. Every trader is indeed interested in evaluating his investment returns and taking risks accordingly. We simulate market scenarios [129] by evaluating the annualized returns by the predicted stock trading decisions provided by DRDL using 1 to 3 layers as well as the decisions from from the benchmark models. Table 4.8 presents a detailed study of nine stocks for DRDL methodology and state-of-the-art methods. We display only empirical values for nine stocks and the average results over the 180 stocks. To make it easy for readers we have

highlighted best annualized returns in bold. It is clearly evident that the DRDL approach yields higher returns when tested for a duration of 10 years when compared to annualized returns obtained from deep learning state-of-the-art methods predictions.

4.3.5.4 Portfolio diversification

	Stock symbols	DRDL 3 layers	DRDL 2 layers	DRDL 1 layer
Small-cap	ALOKTEXT.BO	**1.04**	1.20	1.09
	ALKYLAMINE.BO	**1.17**	1.19	1.34
	ZEEMEDIA6.BO	0.89	0.99	**0.23**
	PVP.BO	**1.34**	1.98	2.78
Mid-cap	IOC.BO	1.02	1.05	**0.87**
	TATACHEM.BO	0.76	**0.45**	0.94
	SPICEJET.BO	**0.34**	0.65	1.20
	BHEL.BO	**0.20**	0.51	1.15
Large-cap	AAPL	1.13	**0.98**	1.11
	AMZN	**0.11**	0.41	0.43
	HINDZINC.BO	**0.03**	0.65	0.45
	ONGC.BO	0.20	0.13	**0.09**
	SIEMENS.NS	0.12	**0.02**	0.11

Table 4.9: Log-loss values provided by DRDL using 1 to 3 layers, for different stocks with available market capitalization categories. The log-loss is computed over the test phase.

Small-cap companies are young and seek to expand aggressively. As comes the growth potential, the risk factor follows in a direct proportion. Small-caps are more volatile and vulnerable to losses during the financial market's negative trends (downtime).

Mid-cap is a pooled investment that focuses on including stocks with a middle range of market capitalization. Mid-cap offers investors more enormous growth potential than large-cap stocks but with less volatility and risk than small-caps.

Large-cap companies are typically large, well established, and financially sound.

Large-cap is the market leader, relatively less volatile, and has a steady risk-return factor with relatively lower risk than mid-cap, small-cap.

As explained in sec. 4.2.3, the proposed DRDL methodology evaluates probabilistic quantification. This quantification allows a piece of information that a knowledgeable practitioner can use to trade accordingly to have a diversified portfolio. In addition, such probabilistic validation can help maximizing the returns by having a mix of small-cap, mid-cap, large-cap stocks.

We provide in Table 4.9 the log-loss value (the smaller, the better), computed as described in Sec. 4.2.3, calculated for DRDL results for different layers architecture, for the few stocks of the dataset for which the market capitalization categories were available.[5] The log-loss quantifies the accuracy of the probabilistic predictions of the proposed method, penalizing more the situations where the method assigns a low probability to an event that finally occurs. One can notice that the log-loss value can reach a meager value (i.e., good prediction accuracy) when trading the large-cap stocks, probably due to the less volatile nature of such stocks. In contrast, the log-loss is on average higher for small caps, as they are highly volatile.

4.3.6 Discussion

In this work, time-series sequences are modeled with a flexible multi linear-Gaussian SSM. The transition matrices (state and observation models) are unknown, and are estimated thanks to an expectation-minimization strategy,

[5] https://finance.yahoo.com/screener

assuming a particular deep NMF structure. The DRDL approach inherits advantages from sophisticated modeling techniques while quantifying the uncertainty in the predictions. We have then adapted the DRDL approach to deal with a challenging large scale financial time series problem, to target stock forecasting and trading tasks. In particular, the method is able to successfully operate in an online processing manner, allowing to capture piece-wise linear characteristics in the data. The results show that the proposed method outperforms the state-of-the-art techniques. Given these promising results, we plan as future work to delve deeper into the area of financial forecasting, including the application of our technique in forecasting derivatives.

Chapter 5

Deep Sequential Transform Learning

In this chapter, we propose Deep sequential transform learning (DSTL). The proposed method is a deep network to model multi-linear Gaussian state space in the presence of an exogenous input inherited from Sequential transform learning. This work is proposed in the focus of overcoming the challenges faced by Sequential transform learning discussed in chapter 3. After a comprehensive analysis of the shallow approach, we observed that the performance of the approach was restricted and hindered due to a stringent assumption of the model being a linear state space. This limitation motivated us to explore the multi-linear Gaussian state-space model. The real-time series is non-linear, and it's important to come up with recurrent approaches to capture non-linearity and forecast for future sequences. Deeper neural network architectures are known to yield better results than their shallow counterparts [133, 134]. They are engineered to approximate highly non-linear function in high-dimensional spaces and are supposed to be more suitable for challenging problems [104–107]. Relying on the power of deep learning methods, we modeled the non-linearity in the

multi-linear Gaussian state-space model using the deep latent factor model. Thus the proposed approach is a multi-linear Gaussian state space involving state evolution, observation matrices, and an exogenous control input modeled as a deep latent factor model. The model is also made recurrent by introducing a feedback loop where learnt deep factor model parameters for the t^{th} instant is fed back along with the $t + 1^{st}$ sample. The method is developed to overcome the limitations of the Sequential transform learning model and explore the multi-linear Gaussian state-space model in the presence of exogenous input, which differentiates it from the DRDL approach discussed in chapter 4. The underlying multi-linear SSM structure ensures that we can both predict a point estimate and quantify the uncertainty about it, making it a perfect fit for cryptocurrency forecasting.

At a cursory glance it might seem that the work is just a deep extension of the shallow model proposed in [135]. However, this deep extension is not trivial. The proposed work introduces deep non-negative matrix factorization (deep NMF) models for learning the approximations on operators. Deep NMF [113] is equivalent to Deep Rectified Linear Unit (ReLU) networks. There is existing literature [123,124] which establishes its connection with deep dictionary learning; the first paper is a more generalised non-linear version of the later. The work have utilised the potential of deep NMF / ReLU in the proposed work by embedding it into a Gaussian SSM. This allows for modelling non-linearity of the underlying dynamical process. The main difference between the prior shallow model and the proposed deep one is that the proposed work is more generalized

version of the former. The shallow model can only approximate piece-wise linear functions; the proposed one, being formulated on ReLU network can approximate arbitrary non-linear and non-smooth functions. The price we pay for the generalisation ability of our approach is the difficulty in training. The shallow model allowed for closed form solutions; the proposed deeper extension does not. Hence, the work resort to alternating direction method of multipliers (ADMM) to solve this. The proposed approach is named as Deep State Space Model for Predicting Cryptocurrency Price (DeCrypt).

We applied the very novel approach to solve the most challenging problems of today's age – predicting the prices of cryptocurrencies. The biggest challenge with cryptocurrencies is that, unlike stocks, funds, indices, etc., cryptocurrencies are unrelated to any fundamental commodity; therefore, the rise and fall of its prices are seemingly random. Investopedia defines cryptocurrency as "a digital or virtual currency that is secured by cryptography, which makes it nearly impossible to counterfeit or double-spend" and is built on "decentralized networks based on blockchain technology—a distributed ledger enforced by a disparate network of computers". The reason cryptocurrencies are volatile is because they do not have any intrinsic value. Their prices are mainly dependent on the emotion of investors, and in such a scenario, tweets from influencers can play a major role in swaying their prices; one example of how tweets from a major influencer can drive prices high or low can be seen from [136, 137].

5.1 Proposed Method

This section will discuss the proposed approach (Deep State Space Model for Predicting Cryptocurrency Price (DeCrypt)) in detail. For convenience the proposed method will be referred by name **DeCrypt**.

5.1.1 Model Details

The proposed work is based on standard state space model (SSM). It can be expressed as :

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= f(\mathbf{z}_{k-1}) + g(\mathbf{u}_k) + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= h(\mathbf{z}_k) + \mathbf{v}_{2,k}, \end{cases} \tag{5.1}$$

The goal is to infer $(\mathbf{z}_k)_{1 \leq k \leq K}$, a sequence of unknown latent space vector of size $N_z \geq 1$ given the input $(\mathbf{u}_k)_{1 \leq k \leq K}$ vector of size $N_y \geq 1$ and observed sequence$(\mathbf{x}_k)_{1 \leq k \leq K}$ of vector of size $N_x \geq 1$. The work assume process noises $(\mathbf{v}_{1,k})_{1 \leq k \leq K}$, $(\mathbf{v}_{2,k})_{1 \leq k \leq K}$ to have a Gaussian distribution with zero-mean and covariance matrix Q and R respectively. The covariance matrices are a symmetric definite positive. Here K is the total number of data to be processed (window size in our case).

Traditional solutions to SSM required the functions $f(.), g(.)$ and $h(.)$ to be known. When the functions are linear, prior studies [111,138] proposed a solution called blind Kalman filtering; blind since the functions/matrices were assumed

to be unknown. This work removes the linearity restriction, by embedding ReLU deep neural networks (DNNs) in place of the functions. Our model thus takes the form:

For every $k \in \{1, \ldots, K\}$:

$$\begin{cases} \mathbf{z}_k &= \mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12}\mathbf{z}_{k-1} + \mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22}\mathbf{u}_k + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_k + \mathbf{v}_{2,k}, \end{cases} \tag{5.2}$$

This is a multi-linear Gaussian model; everything except the input $(\mathbf{u}_k)_{1\leq k\leq K}$ and observed sequence $(\mathbf{x}_k)_{1\leq k\leq K}$ are unknown. The primary objective is to jointly learn the latent factor matrices $\mathbf{T}_{10} \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{T}_{11} \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{T}_{12} \in \mathbb{R}^{N_z\times N_z}$ are three positive-valued linear factors leading to a multi-linear state operator $\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12}$, of the control input transition matrices $\mathbf{T}_{20} \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{T}_{21} \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{T}_{22} \in \mathbb{R}^{N_z\times N_y}$ are three positive-valued linear factors leading to a multi-linear control operator $\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22}$ and similarly observation matrices $\mathbf{D}_0 \in \mathbb{R}^{N_x\times N_z}$, $\mathbf{D}_1 \in \mathbb{R}^{N_z\times N_z}$, $\mathbf{D}_2 \in \mathbb{R}^{N_z\times N_z}$ are three positive-valued linear factors yielding the multi-linear observation model $\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2$ and the sequence $(\mathbf{z}_k)_{1\leq k\leq K}$, from observed sequence $(\mathbf{x}_k)_{1\leq k\leq K}$ and $(\mathbf{u}_k)_{1\leq k\leq K}$. [1] The inference problem can be categorised as a blind filtering problem, where the objective is to infer the time series predictions and unknown model parameters from the given input data and observed sequences. As stated earlier, the classical SSM techniques need prior assumptions and information on model parameters. In the proposed model described here, this would imply explic-

[1] Throughout the work, three-terms factorizations is considered , for the sake of readability. The 3-layers modeling and inference methodology has the great advantage of being generic enough to be straightforwardly extended to any number, greater or equals to one, of factors.

itly setting some prior values to the positive latent factor matrices matrices $\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ involved in both state, control and observation models. In real-world applications, especially in areas as complex as financial modelling this is never known; this is largely owing to the non-stationarity and volatility of the process. The main objective of the proposed work is to provide a point-wise estimate of the positive latent factor matrices $\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ and obtain a probabilistic estimate of sequence $(\mathbf{z}_k)_{1\leq k\leq K}$, given the observed sequence $(\mathbf{x}_k)_{1\leq k\leq K}$ and control input $(\mathbf{u}_k)_{1\leq k\leq K}$. Therefore the work propose to jointly solve for (i) the three deep NMF problems, and (ii) the filtering/smoothing problem.

5.1.2 Model Analysis

This section will describe the fundamental characteristics of the proposed De-Crypt approach. We have used the discrete time invariant state space model with exogenous input. The mathematical fundamentals of the model in Eq. (5.2) and schematic diagram in Fig 5.1. The former part of the equation works on the evolution of the hidden state parameters, where the model assumes Markovianity between two consecutive hidden states. The later part defines the relationship between the hidden and observed states. We depart from all prior works in SSM based dynamical modelling in the way we define the functions. Usually a matrix is used when the functions are assumed to be non-linear and an explicit non-linear function otherwise [125–127]. Here we model non-linearity by a deep ReLU network; alternately this can be also seen as a deep NMF [113]. The

reason for using a deep ReLU network is its universal function approximation capbility [120–122]. It is essential to note here that classical state-space models uses Monte Carlo simulation or Variable Bayes type techniques, doe non-linear SSM. These techniques are complex and usually do not scale well. In contrast, in the proposed method, each layer is modeled and learnt in the form of matrix that can be estimated using alternating direction method of multipliers (ADMM).

When compared to existing literature in machine learning approaches, DNN mostly utilizes in backpropagation for its training [118, 119]. Consequently while modelling dynamical systems, backpropagagtion through time (BPTT) needs to be employed. We are all aware of the pitfalls of BPTT. This is the reason we resort to ADMM instead. Unlike BPTT, ADMMM (under certain conditions) at least guarantees convergence to a local minimum.

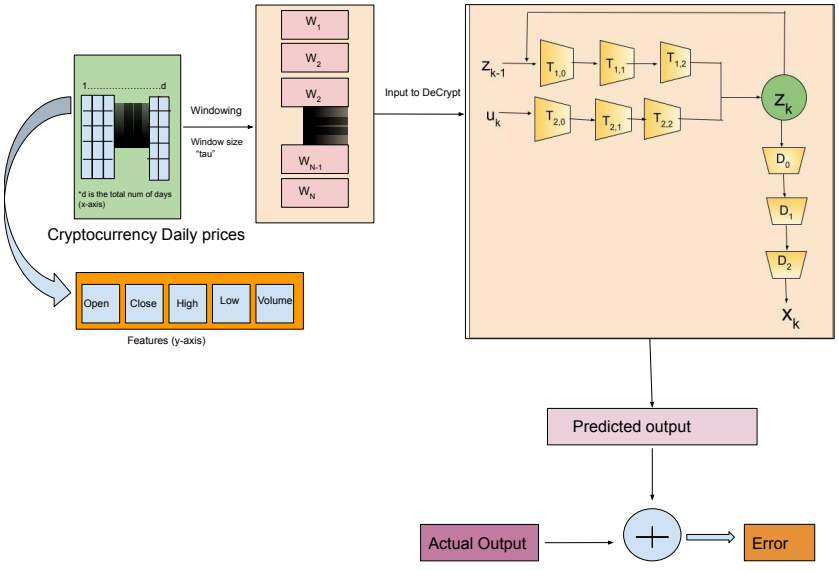


Figure 5.1: Schematic Diagram for Proposed model DeCrypt.

5.1.3 Model Inference Algorithm

The inference problem can be viewed as smoothing/filtering problem where we aim to infer probabilistic estimate of the hidden state $(\mathbf{z}_k)_{1\leq k\leq K}$. In this work, we have also introduced deep NMF factors (described earlier) which are unknown. We aim to jointly infer both probabilistic distribution on hidden state and deep NMF factors estimation from the data. To estimate the state matrices, control input transition matrices and observation transition matrices, we use Expectation-maximization strategy (for more details [83, chap.12] and [2]. The EM strategy operates in two steps namely E-step where we assume the positive latent factor matrices $\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ to be fixed and estimate probabilistic inference for the state representation $(\mathbf{z}_k)_{1\leq k\leq K}$. M-step involves updating these matrices assuming fixed state (learnt from E-step). The E-step is akin to that of a Kalman filter / smoother. The M-step updates the matrices $\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ by maximizing the upper bound :

$$\begin{aligned}\varphi_K(\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2) \\ = \log p(\mathbf{x}_{1:K}|\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2). \quad (5.3)\end{aligned}$$

It is important to note that the inference of the $i+1$-th EM update is obtained from the estimates from the previous iteration i. We explain the EM algorithm in more detail.

5.1.3.1 E-step: Kalman/RTS inference

We consider the latent factors $\mathbf{T}_{10}^{[i]}$, $\mathbf{T}_{11}^{[i]}$, $\mathbf{T}_{12}^{[i]}$, $\mathbf{T}_{20}^{[i]}$, $\mathbf{T}_{21}^{[i]}$, $\mathbf{T}_{22}^{[i]}$, $\mathbf{D}_{0}^{[i]}$, $\mathbf{D}_{1}^{[i]}$, $\mathbf{D}_{2}^{[i]}$ to be fixed. The objective of this step is to infer the probabilistic estimation of the state. The initial state takes the form $\mathbf{z}_0 \sim \mathcal{N}(\bar{\mathbf{z}}_0, \mathbf{P}_0)$ with $\bar{\mathbf{z}}_0 \in \mathbb{R}$ and $\mathbf{P}_0$ defined as definite symmetric positive matrix $\in \mathbb{R}^{N_z \times N_z}$. The probabilistic estimation is provided by the Kalman filter through predictive distribution :

$$p(\mathbf{z}_k|\mathbf{x}_{1:k}, \mathbf{u}_{1:k}) = \mathcal{N}(\mathbf{z}_k; \bar{\mathbf{z}}_k, \mathbf{P}_k). \tag{5.4}$$

For every k, the mean $\bar{\mathbf{z}}_k$ and the covariance $\mathbf{P}_k$ are given by the Kalman iterations:

For $k = 1, \ldots, K$:

Predict state:

$$\begin{cases} \mathbf{z}_k^- &= \mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]}\bar{\mathbf{z}}_{k-1} + \mathbf{T}_{20}^{[i]}\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}\mathbf{u}_k, \\ \mathbf{P}_k^- &= \mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]}\mathbf{P}_{k-1}(\mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]})^\top + \mathbf{Q}. \end{cases} \tag{5.5}$$

Update state:

$$\begin{cases} \mathbf{y}_k &= \mathbf{x}_k - \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{z}_k^-, \\ \mathbf{S}_k &= \mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{P}_k^-(\mathbf{D}_0^{i}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]})^\top + \mathbf{R}, \\ \mathbf{K}_k &= \mathbf{P}_k^-(\mathbf{D}_0^{[i]}\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]})^\top\mathbf{S}_k^{-1}, \\ \mathbf{z}_k &= \mathbf{z}_k^- + \mathbf{K}_k y_k, \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top. \end{cases} \tag{5.6}$$

Hereabove, $\mathbf{y}_k$ represents the measurement pre-fit residual, $\mathbf{S}_k$ represents the pre-fit covariance, $\mathbf{K}_k$ represents Kalman gain, $\bar{\mathbf{z}}_k$ represents the updated (a posteriori) state estimate, $\mathbf{P}_k$ represents the updated (a posteriori) covariance estimate. The backward recursion from the RTS smoother allow to build the smoothing distribution $p(\mathbf{z}_k|\mathbf{x}_{1:K}, \mathbf{u}_{1:K})$. For $k = K, \ldots, 1$

Backward Recursion (Bayesian Smoothing):

$$\begin{cases} \mathbf{z}_{k+1}^{-} & = \mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]}\bar{\mathbf{z}}_k + \mathbf{T}_{20}^{[i]}\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}\mathbf{u}_k, \\ \mathbf{P}_{k+1}^{-} & = \mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]}\mathbf{P}_k(\mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]})^{\top} + \mathbf{Q}, \\ \mathbf{G}_k & = \mathbf{P}_k(\mathbf{T}_{10}^{[i]}\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]})^{\top}[\mathbf{P}_{k+1}^{-}]^{-1}, \\ \mathbf{z}_k^s & = \mathbf{z}_k + \mathbf{G}_k[\mathbf{z}_{k+1}^s - \mathbf{z}_{k+1}^{-}], \\ \mathbf{P}_k^s & = \mathbf{P}_k + \mathbf{G}_k[\mathbf{P}_{k+1}^s - \mathbf{P}_{k+1}^{-}]\mathbf{G}_k^{\top}. \end{cases} \tag{5.7}$$

5.1.3.2 M-step: Operator update

This step utilizes the estimated state ($\mathbf{z}_k$) following an optimization step to increase the likelihood of the matrix parameters $\mathbf{T}_{10}$, $\mathbf{T}_{11}$, $\mathbf{T}_{12}$, $\mathbf{T}_{20}$, $\mathbf{T}_{21}$, $\mathbf{T}_{22}$, $\mathbf{D}_0$, $\mathbf{D}_1$, $\mathbf{D}_2$, using the smoothed predictive distribution obtained in the E-step.

$$\begin{aligned} &\varphi_K(\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2) \\ &\qquad \geq \mathcal{Q}\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2; \boldsymbol{\Theta}^{[i]}). \end{aligned} \tag{5.8}$$

$$p(\mathbf{z}_k|\mathbf{x}_{1:K}, \mathbf{u}_{1:k}) = \mathcal{N}(\mathbf{z}_k; \mathbf{z}_k^s, \mathbf{P}_k^s). \tag{5.9}$$

Here above, $\mathbf{\Theta}^{[i]} = \{\mathbf{\Sigma}^{[i]}, \mathbf{\Phi}^{[i]}, \mathbf{B}^{[i]}, \mathbf{C}^{[i]}, \mathbf{\Delta}^{[i]}, \mathbf{A}^{[i]}, \mathbf{F}^{[i]}, \mathbf{I}^{[i]}\}$ gathers eight quantities(variables) defined from the outputs of the E-step described in Sec. 5.1.3.1):

$$
\begin{aligned}
&\mathbf{Q}(\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2; \mathbf{\Theta}^{[i]}) = \\
&+\frac{K}{2}\text{tr}\Big(\mathbf{Q}^{-1}(\mathbf{\Sigma}^{[i]} - (\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})^\top\mathbf{C}^{[i]} - \mathbf{A}^{[i]}(\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})^\top - (\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})(\mathbf{C}^{[i]})^\top \\
&+(\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})\mathbf{\Phi}^{[i]}(\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})^\top + (\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})\mathbf{F}^{[i]}(\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})^\top - (\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})(\mathbf{A}^{[i]})^\top \\
&+(\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})(\mathbf{F}^{[i]})^\top(\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})^\top + (\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})\mathbf{I}^{[i]}(\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22})^\top)\Big) \\
&+\frac{K}{2}\text{tr}\Big(\mathbf{R}^{-1}\mathbf{\Delta}^{[i]} - \mathbf{B}^{[i]}(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2)^\top - \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2(\mathbf{B}^{[i]})^\top + \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{\Sigma}^{[i]}(\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2)^\top\Big),
\end{aligned}
\tag{5.10}
$$

with:

$$
\begin{cases}
\mathbf{\Sigma}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_k^s + \mathbf{z}_k^s(\mathbf{z}_k^s)^\top, \\
\mathbf{\Phi}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{P}_{k-1}^s + \mathbf{z}_{k-1}^s(\mathbf{z}_{k-1}^s)^\top, \\
\mathbf{B}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k(\mathbf{z}_k^s)^\top, \\
\mathbf{C}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}(\mathbf{P}_k^s\mathbf{G}_{k-1}^\top + \mathbf{z}_k^s(\mathbf{z}_{k-1}^s)^\top). \\
\mathbf{A}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_k^s\mathbf{u}_k^\top \\
\mathbf{F}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_{k-1}^s\mathbf{u}_k^\top \\
\mathbf{I}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{u}_k\mathbf{u}_k^\top \\
\mathbf{\Delta}^{[i]} &= \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k\mathbf{x}_k^\top
\end{cases}
\tag{5.11}
$$

In this step lies the main bottleneck of the deeper extension (compared to the shallow state space model of [135]). For the shallow model each of the operators was a single matrix; therefore there update step resulted in a linear inverse problem. Such is not the current case; here the variables are multi-linear in nature. Therefore we do not have the simple (analytic) updates - as was the case for the shallow model. We have to resort to the paradigm of alternating direction method of multipliers (ADMM) [139–141] for solving updating the variables from the multi-linear form. In ADMM, the idea is that, one can update

one variable assuming the others to be constant and as long as each of the variables have a closed form update, the overall optimization will reach a local minimum. Based on the ADMM approach the computations each variable under the positivity constraints on the factors $\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}^{[i+1]}$ and $\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}$ leads to:

$$(\mathbf{T}_{10})^{[i+1]} = \underset{\mathbf{T}_{10}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}, \mathbf{T}_{11}^{[i]}, \mathbf{T}_{12}^{[i]}, \mathbf{T}_{20}^{[i]}, \mathbf{T}_{21}^{[i]}, \mathbf{T}_{22}^{[i]}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{T}_{11})^{[i+1]} = \underset{\mathbf{T}_{11}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}, \mathbf{T}_{12}^{[i]}, \mathbf{T}_{20}^{[i]}, \mathbf{T}_{21}^{[i]}, \mathbf{T}_{22}^{[i]}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{T}_{12})^{[i+1]} = \underset{\mathbf{T}_{12}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}, \mathbf{T}_{20}^{[i]}, \mathbf{T}_{21}^{[i]}, \mathbf{T}_{22}^{[i]}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{T}_{20})^{[i+1]} = \underset{\mathbf{T}_{20}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}, \mathbf{T}_{21}^{[i]}, \mathbf{T}_{22}^{[i]}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{T}_{21})^{[i+1]} = \underset{\mathbf{T}_{21}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}, \mathbf{T}_{22}^{[i]}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{T}_{22})^{[i+1]} = \underset{\mathbf{T}_{22}\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}\mathbf{D}_0^{[i]}, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{D}_0)^{[i+1]} = \underset{\mathbf{D}_0\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}^{[i+1]}\mathbf{D}_0, \mathbf{D}_1^{[i]}, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{D}_1)^{[i+1]} = \underset{\mathbf{D}_1\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}^{[i]}, \mathbf{D}_0^{[i]}, \mathbf{D}_1, \mathbf{D}_2^{[i]}; \mathbf{\Theta}^{[i]})$$

$$(\mathbf{D}_2)^{[i+1]} = \underset{\mathbf{D}_2\geq 0}{\operatorname{argmax}} \mathbf{Q}(\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}^{[i+1]}\mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2; \mathbf{\Theta}^{[i]})$$

The above sub-problems can easily be rewritten as the minimization of convex quadratic functions which can be solved through several solvers. We stick to use simple alternating direction method of multipliers which is also reminiscent from the literature of deep nonnegative matrix factorization [118], and the deep ReLu neural networks models [121]. The deep neural network (DNN) based operators are regularized by imposing a positivity constraint on the entries of the estimated matrices, by simply projecting them onto the positive orthant after

each update of the M-step (similar to ReLU activation function). This is akin to deep non-negative matrix factorization [116, 142], while keeping the convergence behaviour of the EM algorithm. DNN with ReLU activation is known for its function approximation ability [122, 143]. This yields the following analytic updates:

$$
\begin{aligned}
\mathbf{T}_{10}^{[i+1]} &= \text{ReLu}\Big(\big(\mathbf{C}^{[i]}(\mathbf{T}_{12}^{[i]})^\top(\mathbf{T}_{11}^{[i]})^\top\big) - \big(\mathbf{T}_{20}^{[i]}\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}(F^{[i]})^\top(\mathbf{T}_{12}^{[i]})^\top(\mathbf{T}_{11}^{[i]})^\top\big) \\
&\quad \times(\mathbf{T}_{11}^{[i]}\mathbf{T}_{12}^{[i]}\mathbf{\Phi}^{[i]}(\mathbf{T}_{12}^{[i]})^\top(\mathbf{T}_{11}^{[i]})^\top))^\dagger\Big) \\
\mathbf{T}_{11}^{[i+1]} &= \text{ReLu}\Big(((\mathbf{T}_{10}^{[i+1]})^\top\mathbf{Q}^{-1}(\mathbf{T}_{10}^{[i+1]})^{-1})\big(((\mathbf{T}_{10}^{i+1})^\top\mathbf{Q}^{-1}\mathbf{C}^{[i]}\mathbf{T}_{12}^{[i]}\big) \\
&\quad -\big(\mathbf{T}_{10}^{[i+1]}\mathbf{Q}^{-1}\mathbf{T}_{20}^{[i]}\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}(\mathbf{F}^{[i]})^\top\mathbf{T}_{12}^{[i]}\big)\big) \times (\mathbf{T}_{12}^{[i]}\mathbf{\Phi}^{[i]}(\mathbf{T}_{12}^{[i]})^\top)^\dagger\Big) \\
\mathbf{T}_{12}^{[i+1]} &= \text{ReLu}\Big(((\mathbf{T}_{11}^{[i+1]})^\top(\mathbf{T}_{10}^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{T}_{10}^{[i+1]}\mathbf{T}_{11}^{[i+1]})^\dagger \times \big((\mathbf{T}_{11}^{[i+1]})^\top(\mathbf{T}_{10}^{[i+1]})^\top\mathbf{C}^{[i]}\mathbf{Q}^{-1}\big) \\
&\quad -\big((\mathbf{T}_{11}^{[i+1]})^\top(\mathbf{T}_{10}^{[i+1]})^\top(\mathbf{Q}^{-1}\mathbf{T}_{20}^{[i]}\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}(\mathbf{F}^{[i]})^\top)\mathbf{\Phi}^{-1}\big)\Big) \\
\mathbf{T}_{20}^{[i+1]} &= \text{ReLu}\Big(\big((\mathbf{T}_{22}^{[i]})^\top(\mathbf{T}_{21}^{[i]})^\top - \mathbf{T}_{10}^{[i+1]}\mathbf{T}_{11}^{[i+1]}\mathbf{T}_{12}^{[i+1]}\mathbf{F}^{[i]}(\mathbf{T}_{22}^{[i]})^\top(\mathbf{T}_{21}^{[i]})^\top\big) \\
&\quad \times(\mathbf{T}_{21}^{[i]}\mathbf{T}_{22}^{[i]}\mathbf{I}^{[i]}(\mathbf{T}_{21}^{[i]})^\top(\mathbf{T}_{22}^{[i]})^\top))^\dagger\Big) \\
\mathbf{T}_{21}^{[i+1]} &= \text{ReLu}\Big(\big(\mathbf{T}_{20}^{[i+1]}\mathbf{Q}^{-1}(\mathbf{T}_{20}^{[i+1]})^\top\big)^\dagger\big(\mathbf{T}_{20}^{[i+1]}\mathbf{A}^{[i+1]}\mathbf{Q}^{-1}(\mathbf{T}_{22}^{[i]})^\top \\
&\quad -\big((\mathbf{T}_{20}^{[i+1]})^\top\mathbf{Q}^{-1}(\mathbf{T}_{10}^{[i+1]})(\mathbf{T}_{11}^{[i+1]})(\mathbf{T}_{12}^{[i+1]})\mathbf{F}^{[i]}(\mathbf{T}_{22}^{[i]})^\top)\big) \\
&\quad \times(\mathbf{T}_{22}^{[i]}\mathbf{I}^{[i]}(\mathbf{T}_{22}^{[i]})^\top)^\dagger\Big) \\
\mathbf{T}_{22}^{[i+1]} &= \text{ReLu}\Big(\big((\mathbf{T}_{21}^{[i+1]})^\top(\mathbf{T}_{20}^{[i+1]})^\top\mathbf{Q}^{-1}(\mathbf{T}_{20}^{[i+1]})(\mathbf{T}_{21}^{[i+1]})\big) \times \big(\big((\mathbf{T}_{21}^{[i+1]})^\top(\mathbf{T}_{20}^{[i+1]})^\top\mathbf{Q}^{-1}\mathbf{A}^{[i]}\big) \\
&\quad -\big((\mathbf{T}_{21}^{[i+1]})^\top(\mathbf{T}_{20}^{[i+1]})^\top\mathbf{Q}^{-1} \times (\mathbf{T}_{10}^{[i+1]})^\top(\mathbf{T}_{11}^{[i+1]})^\top(\mathbf{T}_{12}^{[i+1]})^\top\mathbf{F}^{[i]}\big)\big)(\mathbf{I}^{[i]})^{-1}\Big) \\
\mathbf{D}_0^{[i+1]} &= \text{ReLu}\Big(\mathbf{B}^{[i]}(\mathbf{D}_2^{[i]})^\top(\mathbf{D}_1^{[i]})^\top(\mathbf{D}_1^{[i]}\mathbf{D}_2^{[i]}\mathbf{\Sigma}^{[i]}(\mathbf{D}_2^{[i]})^\top(\mathbf{D}_1^{[i]})^\top)^\dagger\Big) \\
\mathbf{D}_1^{[i+1]} &= \text{ReLu}\Big(((\mathbf{D}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{D}_0^{[i+1]})^\dagger((\mathbf{D}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{B}^{[i]}(\mathbf{D}_2^{[i]})^\top \\
&\quad \times(\mathbf{D}_2^{[i]}\mathbf{\Sigma}^{[i]}(\mathbf{D}_2^{[i]})^\top)^\dagger\Big) \\
\mathbf{D}_2^{[i+1]} &= \text{ReLu}\Big(((\mathbf{D}_1^{[i+1]})^\top(\mathbf{D}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{D}_0^{[i+1]}\mathbf{D}_1^{[i+1]})^\dagger(\mathbf{D}_1^{[i+1]})^\top \\
&\quad \times(\mathbf{D}_0^{[i+1]})^\top\mathbf{R}^{-1}\mathbf{B}^{[i]}(\mathbf{\Sigma}^{[i]]})^{-1}\Big)\,.
\end{aligned}
\tag{5.12}
$$

Hereabove, we use pseudo-inverse operator denoted by $(\cdot)^\dagger$. Each operator is passed over activation function ReLu which stands for, ReLu $(\cdot)$ the rectified

linear unit function, that projects each entry of its input to the positive orthant.

5.1.4 Model Summary

The Proposed algorithm is summarized in Alg.4. The algorithm infers the probabilistic estimation of the hidden state $(\mathbf{z}_k)_{1\leq k\leq K}$ jointly with the estimation of latent spaces $\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ by following the eq. (5.2). The DeCrypt model is ran for $i_{\max}$ number of iteration to achieve the stabilisation of the latent spaces.

Algorithm 4. Proposed model inference algorithm.

Inputs. *Prior parameters* $(\overline{\mathbf{z}}_0, \mathbf{P}_0)$ *; model noise covariance matrices* $\mathbf{Q}$, $\mathbf{R}$ *; set of observations* $\{\mathbf{x}_k\}_{1\leq k\leq K}$ *and control input* $(\mathbf{u}_k)_{1\leq k\leq K}$.

Initialization. *Set positive latent factors*

$\{\mathbf{T}_{10}, \mathbf{T}_{11}, \mathbf{T}_{12}, \mathbf{T}_{20}, \mathbf{T}_{21}, \mathbf{T}_{22}, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$.

Recursive step. *For* $i = 0, 1, \ldots, i_{\max}$*:*

(E step) Run the Kalman filter (5.5)-(5.6) *and RTS smoother* (5.7) *using latent factors* $\mathbf{T}_{10}^{[i]}$, $\mathbf{T}_{11}^{[i]}$, $\mathbf{T}_{12}^{[i]}$, $\mathbf{T}_{20}^{[i]}$, $\mathbf{T}_{21}^{[i]}$, $\mathbf{T}_{22}^{[i]}$, $\mathbf{D}_0^{[i]}$, $\mathbf{D}_1^{[i]}$, $\mathbf{D}_2^{[i]}$.
Calculate $\mathbf{\Sigma}^{[i]}, \mathbf{\Phi}^{[i]}, \mathbf{B}^{[i]}, \mathbf{C}^{[i]}, \mathbf{\Delta}^{[i]}, \mathbf{A}^{[i]}, \mathbf{F}^{[i]}, \mathbf{I}^{[i]}$ *using* (5.11).

(M step) Compute $\{\mathbf{T}_{10}^{[i+1]}, \mathbf{T}_{11}^{[i+1]}, \mathbf{T}_{12}^{[i+1]}, \mathbf{T}_{20}^{[i+1]}, \mathbf{T}_{21}^{[i+1]}, \mathbf{T}_{22}^{[i+1]}, \mathbf{D}_0^{[i+1]}, \mathbf{D}_1^{[i+1]}, \mathbf{D}_2^{[i+1]}\}$ *using* (5.12).

Output. *State filtering/smoothing pdfs* (5.4) *and* (5.9) *along with pointwise estimates of the latent factor from* (5.12).

5.1.4.1 Time complexity

▷ **Training.** We describe the time complexity for training DeCrypt in a given

window of length τ. The complexity can be understood by delving into Kalman-based approaches [144], which imply complexity analysis to $\mathcal{O}(\tau N_z^{2.376})$.

$\triangleright$ **Testing.** In testing phase we just perform evaluation of multi-linear equation (Equation 5.14). This concludes the complexity of $\mathcal{O}(N_x N_z^2)$ for each window. This can be further optimized to $\mathcal{O}(N_z^2)$ if performing forecast for just one feature (which looks very similar to the proposed case, ie. forecasting close price.)

5.2 Application to cryptocurrency price forecasting

This section discuss in detail how the proposed approach is applied on the very challenging application of predicting next day prices for crypto-currency.

5.2.1 Training

The major drawback of the Expected-Maximization (EM) strategy used in the proposed approach is that it requires reprocessing on *the entire sequence* to estimate the state, control input, and observation transition operators / matrices. The approach requires imposing explicit prior static assumptions on these parameters for the entire duration of the sequence. Such an assumption may not be an appropriate in practice owing to the volatility of the data; furthermore, processing the entire sequence will be computationally expensive. Owing to the volatility, the parameters should be given the freedom to learn and evolve with time. We thus propose an online implementation based on a simple windowing

strategy. Thus we are able to relax the non-volatility assumption on the entire sequence and reduce processing times.

In the said strategy, a window of size τ is slid on the entire dataset. For every time stamp k, the matrices in the multi-linear operators are estimated using the last τ observations contained in the set $\mathcal{X}_k = \{\mathbf{x}_j\}_{j=k-\tau+1}^{k}$ and $\mathcal{U}_k = \{\mathbf{u}_j\}_{j=k-\tau+1}^{k}$. The proposed EM algorithm is iteratively applied on the window to update the state and operators till convergence. Such a strategy reduces the operational cost significantly. Note that the non-volatility assumption is still there, but only on a small window - this is a reasonable assumption. The major challenge is to estimate a reasonable size of τ. A smaller size would be a better approximation for the non-volatility assumption but would lead to over-fitting on the multi-linear model. On the other hand a larger size would be less prone to over-fitting but would be computationally costly. We initialize the matrices of the multi-linear model using the warm start strategy. The matrices for the current window are initialized with the final values of the prior window. Similarly, the mean and covariance are initialized for $k - \tau + 1$ with the past information on the operators from the smoothing process.

5.2.2 Forecasting

As described in detail in Section *DeCrypt model*, we follow the sliding window strategy. Training each observed window $\mathbf{x}_k$ along with control-input $\mathbf{u}_k$, extracts latent space features and helps in updating the parameters by following Expected Maximization(EM) alternately. Once EM iteration stabilizes we have used 50

iterations for EM to converge used in Alg.4, we use the latent space features and learned matrices to estimate the close price for the next timestamp (i.e., the day indexed as $k+\tau+1$).

$$\begin{cases} \mathbf{z}_k &= \mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12}\mathbf{z}_{k-1} + \mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22}\mathbf{u}_k + \mathbf{v}_{1,k}, \\ \mathbf{x}_k &= \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_k + \mathbf{v}_{2,k}, \end{cases}$$

where input $\mathbf{u}_k$ is $(\mathbf{u}_j)_{k\le j\le k+\tau} \in \mathbb{R}^1$ which is computed using the technical indicator SMA(simple moving average)[2]. Simple moving average (SMA) calculates the average of the a fixed range of prices, usually closing price by the number of period in that range.

$$SMA = \frac{c_1 + c_2 + ... + c_n}{n} \tag{5.13}$$

where c_n= closing price of an asset for period n and n= number of total periods. The processing sequence(observed) $\mathbf{x}_k$ is $(\mathbf{x}_j)_{k\le j\le k+\tau} \in \mathbb{R}^5$ for every $k \in \{0,\ldots,K-\tau\}$, where $x_j[1]$ is the daily opening price, $x_j[2]$ is the daily adjusted close price, $x_j[3]$ is the daily high value, $x_j[4]$ is the daily low value, and $x_j[5]$ is the daily net asset volume. Running the DeCrypt model on the considered window yields the mean estimate of the five features for the immediate time stamp which can be indexed as $k+\tau+1$:

$$\widehat{\mathbf{x}}_{k+\alpha+1} = \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}^-_{k+\alpha}, \tag{5.14}$$

The associated covariance matrix is defined as $\mathbf{S}_{k+\tau}$ for immediate next time stamp indexed as $k+\tau+1$. In particular, *DeCrypt* is designed to forecast the

[2] https://www.investopedia.com/terms/s/sma.asp

whole five dimensional vector, but we focus primarily on prediction of single entry of the vector i.e., adjusted closing price of the sequence.

5.2.3 Uncertainty Quantification

The proposed approach is based on a probabilistic framework and hence can provide confidence score/uncertainty quantification associated with each point-wise estimation. The quantification provides predictive distribution of the future observation which is conditioned on previously seen control-input $(\mathbf{u}_k)$ and observed sequence $(\mathbf{x}_k)$. The probabilistic validation provides informed decision about the (un)certainty associated with model estimation while predicting the future prices of the cryptocurrencies. For each index k, the distribution of the prediction conditioned on $\widehat{\mathbf{x}}_k$ past observations and control-input $\widehat{\mathbf{u}}_k$:

$$p(\widehat{\mathbf{x}}_k|\mathbf{x}_{1:k-1}, \mathbf{u}_{1:k-1}) = \mathcal{N}\left(\widehat{\mathbf{x}}_k; \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_k^-, \mathbf{S}_k\right), \tag{5.15}$$

where the covariance is defined as $\mathbf{S}_k = \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2((\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})\mathbf{P}_{k-1}(\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12})^\top + \mathbf{Q}) + \mathbf{R}$. It is important to note that $\mathbf{z}_k^-$ and $\mathbf{P}_k$ are achieved from Kalman filter, defined in Section E-step in DeCrypt model (see [145] for more details). The main objective of our approach is to estimate the uncertainty score associated with the prediction given by model for price forecasting. In particular, the main aim is to focus on forecasting the sum of prediction i.e., estimating the first entry $\widehat{\mathbf{x}}_k$ denoted by $\widehat{\mathbf{x}}_k[0]$. The quantification about the prediction can be obtained from first row and column of $\mathbf{S}_k$, depicted by $\mathbf{S}_k[0, 0]$. We define the (un)certainty

score about an increase of the price forecasting value as :

$$\widehat{p}_k = \int_{\widehat{\mathbf{x}}_k[0]}^{+\infty} \mathcal{N}\left(y; \left[\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_k^-\right][0], \mathbf{S}_k[0,0]\right) dy \tag{5.16}$$

$$= 1 - \text{CDF}(\widehat{\mathbf{x}}_k[0] | \left[\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{z}_k^-\right][0], \mathbf{S}_k[0,0]), \tag{5.17}$$

where CDF depicts the cumulative distribution function for the multivariate predictive distribution. The equation above quantifies and estimates the probability that forecasting price value will grow in the future time stamp. After estimating $\widehat{p}_k$ for every index k, we evaluate cross entropy loss defined as :

$$\text{log-loss} = \frac{1}{K}\sum_{k=1}^{K} -\left(L_k[i]\log(\widehat{p}_k)\right), \tag{5.18}$$

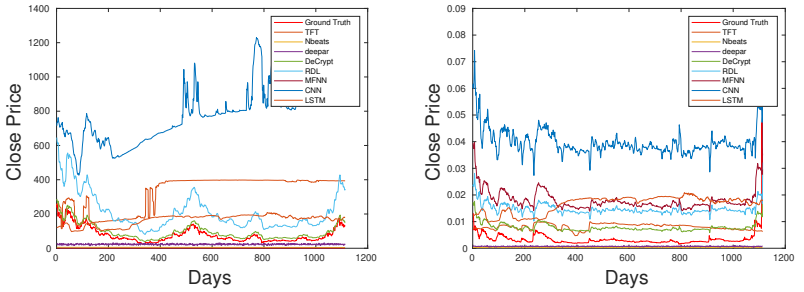
where ground-truth is depicted as $L_k \in \{0, 1\}$ at time k to increase/decrease.

5.3 Experiments and Results

This section discuss the experimental results with proposed approach and other state-of-the art methods. The section presents qualitative and quantitative analysis, hence presenting comprehensive study for modeling time series signals.

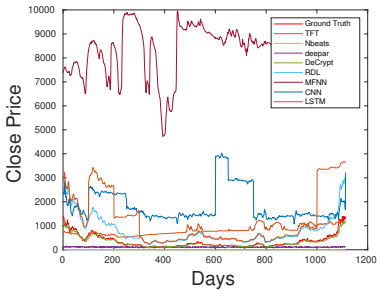
5.3.1 Dataset description

The dataset is described in Chapter1 section1.3.2.

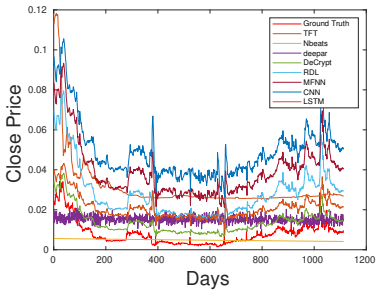


(a) Litecoin

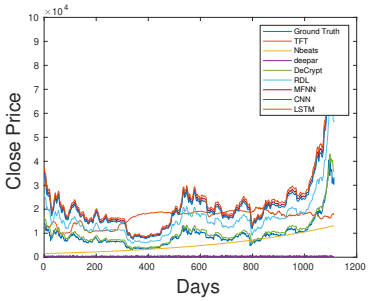
(b) Dogecoin



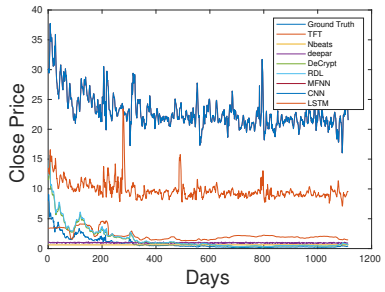
(c) Ethereum



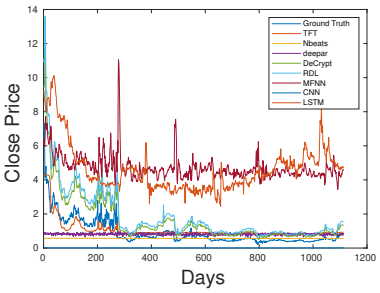
(d) Gridcoin



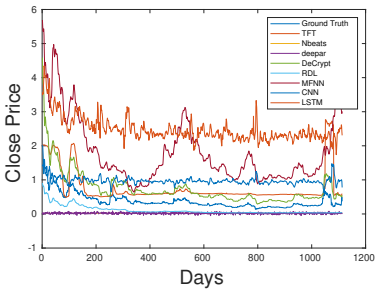
(e) Bitcoin



(f) Peercoin



(g) Namecoin



(h) Ripple

Figure 5.2: Cryptocurrency price forecasting via different algorithms (a) Litecoin, (b) Dogecoin, (c) Ethereum, (d) Gridcoin, (e)Bitcoin, (f) Peercoin, (g) Namecoin, (h) Ripple.

5.3.2 Baseline methods

- N-Beats: Nbeats is a deep neural network structure with forward and backward residual links. It consists of a deep stack of fully connected layers. It does not perform any specific feature engineering or scaling [131].
- Deep Auto regressive (DeepAR) : DeepAR functions by producing probabilistic forecast on long time series sequences [130].
- Temporal Fusion Transformers (TFT): An attention mechanism based recurrent architecture which brings together multi-horizon forecasting without having prior information on how they interact with the target [132].
- Long short term memory(LSTM) : Stacked LSTM with 2-layer architecture is used to forecast the time series sequences. The LSTM used comprises 50 cells and ReLU activation to estimate the predictions. [146].
- Convolutional neural network- Technical analysis(CNN-TA) :1-D Time series is converted to 2-D matrix with technical indicators as rows and time-units as columns. 2D CNN is used for classification and regression. [129]
- Recurrent dictionary learning (RDL): The RDL approach can be assumed to be a shallow (single layer) version of the proposed work. It can only model piece-wise linear functions. [110]
- Multi filter neural network (MFNN) : An end-to-end deep neural network comprising of recurrent neural network and convolutional neural network. [93]

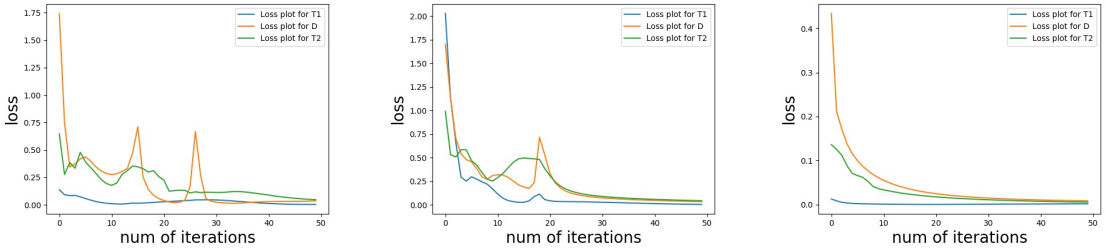
- RAO-ANN: The Rao algorithms can be categorised as the metaphor-less optimization techniques which is utilized in optimizing the parameters of ANN in forecasting crypto-currency prices [147].
- ARIMA: Autoregressive integrated moving average (ARIMA) methodology is used to forecast cryptocurrency prices. The work identifies the parameter of ARIMA model using partial auto-correlation functions (PACF) and auto correlation function (ACF). [148]

We have compared our proposed approach with above mentioned models. The ARIMA parameters are set to $(p, d, q) = (2, 1, 2)$ as it was observed to lead to the best practical performance. We modified LSTM from its original version, by removing the softmax layer and instead included a fully-connected layer to obtain a one node output. The Adam optimizer has been used with learning rate of 10^{-4}, 200 epochs and batch-size 16 is maintained to minimize the root mean square error. For methods like N-Beats, DeepAR, TFT, CNN-TA, MFNN, RDL, RAO-ANN we stick to their original implementation as described in respective papers mentioned above.

5.3.3 Performance Analysis

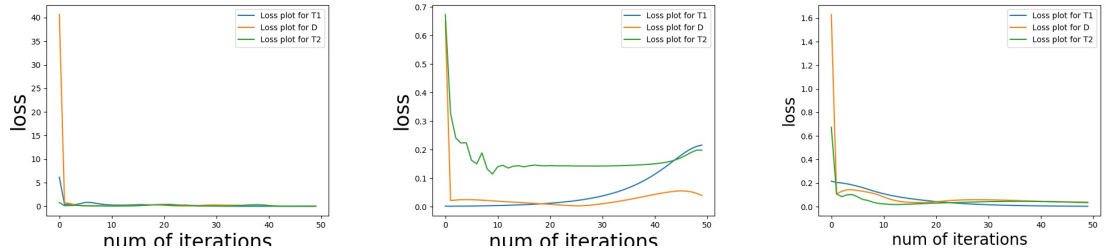
Our proposed approach is non-parametric. It only requires the specification of the window size; we found that $\tau = 50$ yields good results for all cryptocurrencies. The rest of the variables require initialization. These are given below: $\mathbf{P}_0 = \sigma_P^2 \mathbf{I}$

$\mathbf{Q} = \sigma_Q^2 \mathbf{I}$



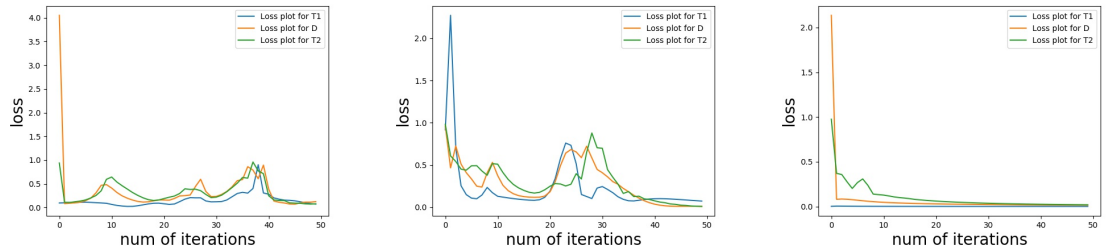
(a) 1−Layer (b) 2 Layers (c) 3 Layers

(A.) Convergence plot for Bitcoin



(a) 1 Layer (b) 2 Layers (c) 3 Layers

(B.) Convergence plot for Gridcoin



(a) 1 Layer (b) 2 Layers (c) 3 Layers

(B.) Convergence plot for Litecoin

Figure 5.3: Convergence plots for different layer architecture((a.) 1 Layer, (b.) 2 Layers, (c.) 3 Layer) for (A.) Bitcoin, (B.) Gridcoin, (C.) Litecoin

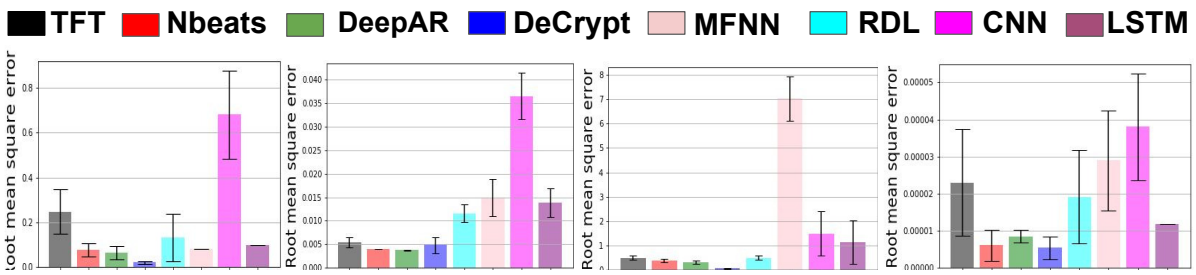


Figure 5.4: Error bar plot comparing RMSE for DeCrypt against other baseline methods for (a) Litecoin, (b) Dogecoin, (c) Ethereum, (d) Gridcoin.

$$\mathbf{R} = \sigma_R^2 \mathbf{I}$$

$(\sigma_Q, \sigma_R, \sigma_P) = (10^{-5}, 10^{-1}, 10^{-1})$, $\mathbf{I}$: identity matrix

$\overline{z}_0 = \mathbf{0}$

N_z= 5,N_y= 1, N_x=5,

During training, the entries of the DNN matrices / operators are initialized at time 0 using a uniform distribution on $[0, 10^{-1}]$. During the test phase, these matrices are fixed, and only the Kalman/RTS inference is run. All presented scores are averaged over 10 trials, and computed only during the test phase. More specifically, we will distinguish in our experiments:

DeCrypt (1 layer): $\mathbf{T}_{11} = \mathbf{T}_{12} = \mathbf{I}$ is fixed and $\{\mathbf{T}_{10}\}$ is estimated; $\mathbf{T}_{21} = \mathbf{T}_{22} = \mathbf{I}$ is fixed and $\{\mathbf{T}_{20}\}$ is estimated; and $\mathbf{D}_1 = \mathbf{D}_2 = \mathbf{I}$ fixed and $\{\mathbf{D}_0\}$ is estimated;

DeCrypt (2 layers): $\mathbf{T}_{12} = \mathbf{I}$ is fixed and $\{\mathbf{T}_{10}\,\mathbf{T}_{11}\}$ is estimated; $\mathbf{T}_{22} = \mathbf{I}$ is fixed and $\{\mathbf{T}_{20}, \mathbf{T}_{21}\}$ is estimated; and $\mathbf{D}_2 = \mathbf{I}$ fixed and $\{\mathbf{D}_0\,\mathbf{D}_1\}$ is estimated;

DeCrypt (3 layers): $\{\mathbf{T}_{10}\,\mathbf{T}_{11}\,\mathbf{T}_{12}\}$, $\{\mathbf{T}_{20}\,\mathbf{T}_{21}\,\mathbf{T}_{22}\}$ is estimated $\{\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2\}$ are estimated.

It is important to note that ignoring the positivity constraints on DeCrypt (1 layer) would identify with the previous work [135]. For benchmarking we have used the commons metrics for regression; they are **Root mean square error** (RMSE), **Mean Absolute Percentage Error** (MAPE) and **Symmetric Mean Absolute Percentage Error** (SMAPE). All the said metrics are based on the error between the actual and predicted prices and a lower value implies better result. We have

also computed the **Pearson correlation coefficient** (r) between the predicted and actual prices; for this metric a higher value implies better result.

We are showing these results for a given window size $\tau = 50$ and number of layers. If we continue to increase the window size, the results start to deteriorate, this is likely due to over-fitting. Increasing the window size does not help either; larger window size fails to capture the volatility of the data and hence the performance falls.

5.3.4 Result Analysis Discussions

In this section we focus on the performance analysis of the proposed approach. The proposed approach (DeCrypt) has been compared with various numerical methods like N-Beats, DeepAR, TFT, CNN-TA, MFNN, RDL, RAO-ANN, ARIMA, RAO-ANN.

5.3.4.1 Influence of window size and depth

Table 5.1: Comparative performance of the proposed approach against different window size for 1-layer architecture. Lower value(↓) is considered the better.

Window size(τ)	r	RMSE↓	MAPE(%)↓	SMAPE(%)↓
10	0.38	0.51	72.1	69.3
15	0.37	0.50	73.2	70.4
20	0.42	0.47	69.6	65.2
25	0.47	0.45	65.4	61.2
30	0.53	0.43	61.3	57.3
35	0.57	0.46	58.5	53.4
40	0.63	0.39	51.8	52.3
45	0.67	0.32	43.6	41.5
50	0.73	0.21	35.7	32.3
55	0.73	0.22	36.2	33.2
60	0.71	0.20	33.2	31.7

The proposed solution is non-parametric. The only design parameters that

Table 5.2: Comparative performance of the proposed approach against different window size for 2-layer architecture. Lower value(↓) is considered the better.

Window size(τ)	r	RMSE↓	MAPE(%)↓	SMAPE(%)↓
10	0.42	0.59	74.1	71.2
15	0.45	0.54	73.5	70.7
20	0.49	0.43	71.8	70.2
25	0.53	0.41	68.5	65.6
30	0.57	0.39	65.1	64.3
35	0.59	0.33	58.6	57.8
40	0.65	0.27	51.5	50.8
45	0.69	0.21	47.5	39.7
50	0.76	0.17	31.2	28.7
55	0.77	0.17	32.3	29.6
60	0.75	0.15	31.6	27.8

Table 5.3: Comparative performance of the proposed approach against different window size for 3-layer architecture. Lower value(↓) is considered the better.

Window size(τ)	r	RMSE↓	MAPE(%)↓	SMAPE(%)↓
10	0.43	0.59	0.76	0.72
15	0.47	0.56	0.71	0.71
20	0.48	0.51	0.67	0.65
25	0.53	0.47	0.61	0.62
30	0.59	0.42	0.59	0.57
35	0.64	0.35	0.54	0.51
40	0.69	0.27	0.43	0.42
45	0.73	0.21	0.39	0.33
50	0.81	0.12	29.1	21.2
55	0.82	0.14	29.3	20.3
60	0.81	0.13	28.4	20.4

need to be fixed are the window size and depth. Therefore, it is very important to choose the optimal window size which finds a good balance between the model complexity (depth) and accuracy. To better understand, we present Table 5.1, Table 5.2, Table 5.3 which represent the window size analysis in DeCrypt (1 layer), DeCrypt (2 layers), DeCrypt (3 layers) architecture respectively. The above mentioned tables provides comprehensive analysis on the performance of the model on varying window sizes. A comprehensive study has been compiled which offers analysis of various metrics like Pearson correlation (r), RMSE (Root Mean Square Error), MAE (Mean Absolute Error), and SMAPE (Symmetric mean absolute percentage error) for different window sizes τ. After analysing

the results we conclude that model performance keeps on improving as window size increases till a stabilization point $\tau = 50$. Ideally one would expect that the results would improve as the window size increases; especially for deeper versions. This is because larger window size means more data and hence better generalisation ability. But note that it is a dynamical model. The tacit assumption here is that in the window the size the underlying dynamical function does not change. While this is true for shorter windows, this does not hold for larger ones as the non-stationarity comes into play. This is the reason we find a trade-off between window size and accuracy. We further set this value of window size in upcoming experiments.

5.3.4.2 Comparison with state-of-the-art methods

Table 5.4: Comparative performance of the proposed approach against baseline methods. Lower value(↓) is considered the better.

Model	r	RMSE↓	MAPE(%)↓	SMAPE(%)↓
LSTM	0.33	0.71	83.2	64.3
CNN-TA	0.29	0.68	91.2	70.8
ARIMA	0.29	0.68	91.2	70.8
Rao-ANN	0.29	0.68	91.2	70.8
MFNN	0.31	0.21	70.28	68.2
N-Beats	0.38	0.27	41.2	38.72
DeepAR	0.30	0.39	46.8	41.47
TFT	0.52	0.24	48.95	40.62
RDL	0.69	0.20	48.7	35.68
ARIMA	0.58	0.49	56.4.	46.24
Rao-ANN	0.42	0.59	68.7	64.38
DeCrypt (1 layer)	0.73	0.21	35.7	32.33
DeCrypt (2 layers)	0.76	0.17	31.2	28.7
DeCrypt (3 layers)	**0.81**	**0.12**	**29.14**	**21.2**

The overall performance of the model vis-a-vis the state-of-the-art is shown in Table5.4. The Table presents the average results for ten cryptocurrencies;

owing to limitations in space we are not able to show individual results. One can see that the proposed method outperforms the baseline by a considerable margin. Fig 5.2 shows the forecast performance of DeCrypt with other baseline methods for visual evaluation and Fig 5.3 shows the convergence plots for model parameters for different layer of architecture for DeCrypt where $\mathbf{T}_1$ is the multi-linear state operator achieved from product of three positive-valued linear factors $\mathbf{T}_{10}\mathbf{T}_{11}\mathbf{T}_{12}$, $\mathbf{T}_2$ is the is the multi-linear control operator achieved from product of three positive-valued linear factors $\mathbf{T}_{20}\mathbf{T}_{21}\mathbf{T}_{22}$ and $\mathbf{D}$ is the multi-linear observation operator achieved from product of three positive-valued linear factors $\mathbf{D}_0\mathbf{D}_1\mathbf{D}_2$. It can be clearly seen that the *DeCrypt* with its three layers architecture outperforms the other baseline approaches significantly. This is mainly because of the deeper network's capacity to better model non-linearity compared to the shallower ones. The model achieves a 0.17 points drop in RMSE, 19.56% drop in MAPE, and 14.48% drop in SMAPE; it gains 0.12 in Pearson's correlation r compared to the best performing benchmarks. We have plotted the error-bars for four different cryptocurrencies (Litecoin, Dogecoin, Ethereum, Gridcoin)in Fig.5.4. From these plots the reader can verify that not only is the proposed method more accurate (least mean error) but is also the most robust (least deviation). In Table 5.6 we present the comparison of performance of proposed approach with state-of-the-art method through statistical test (T-test) with confidence interval of 0.95. We can observe that the T-test values for proposed approach DeCrypt (3 layers) is very small as compared to the other methods, hence we can conclude that more similarity exists between the actual

closing prices and predicted closing prices when compared for different Crypto-currencies. Due to space constraints we have provided Avg. Score for t-test for all the ten cryptocurrency for each method in Table 5.6. From the results we conclude that average T-test score is very low for Decrypt (3 layers) method when compared with other state-of-the-art method, hence we conclude that Decrypt (3 layers) performance is very close to ground truth. In contrast when we see individual crypto-currency analysis from table we can see that TFT outperforms all the methods in Gridcoin and DeepAR outperforms Dogecoin. To understand

Method	Train Time cost (h.)	Test Time cost (min.)
DeCrypt (3 layers)	2.21h	22 min
DeCrypt (2 layers)	2.32h	22.4 min
DeCrypt (1 layer)	1.48h	18.8 min
ARIMA	2.31h	36 min
LSTM	5 days	41 min
DeepAR	2.45h	20 min
TFT	2.25h	27 min
Nbeats	3.12h	25 min
CNN-TA	4.57h	40 min
MFNN	4.12h	37 min
RDL	1.69h	35 min
Rao-ANN	4.35h	25 min

Table 5.5: Averaged time over 50 random runs for processing the dataset (train(hrs) and test(min)), for the proposed approach and its competitors.

Method	Bitcoin	Gridcoin	Dogecoin	Litecoin	Avg. Score
DeCrypt (3 layers)	0.31	0.73	0.82	0.57	0.68
DeCrypt (2 layers)	0.56	0.85	0.91	0.69	0.72
DeCrypt (1 layer)	0.58	0.98	0.99	0.72	0.78
ARIMA	0.90	0.87	0.95	0.93	0.83
LSTM	0.59	1.75	0.97	0.82	0.94
DeepAR	0.54	0.79	0.81	0.55	0.71
TFT	0.40	0.70	1.12	0.71	0.73
Nbeats	0.56	0.68	0.99	0.61	0.84
CNN-TA	0.87	1.53	2.11	1.19	1.13
MFNN	0.82	1.30	1.21	0.74	1.37
RDL	0.58	0.95	0.94	0.68	0.84
Rao-ANN	0.54	0.70	1.02	1.13	0.94

Table 5.6: Comparison of T-test score for the proposed approach with state-of-the-art method for (a) Bitcoin, (b) Gridcoin, (c) Dogecoin, (d) Litecoin, (e)Avg. Score for all the ten crypto-currencies.

the comparison in performance between the proposed method and state-of-the-art methods, we present Table 5.5 which depicts the computational time for forecasting the next day closing price of ten cryptocurrencies. We provide a comprehensive analysis by distinguishing the time required to train and test the methods (on their training and testing time frame using the walk-forward method described in [110, Section 4.2.1]. We conclude that the highest computational time was consumed by LSTM approach. Among the other methods, DeCrypt (1 layer) and RDL method is the fastest while the computation time of DeCrypt (3 layers) is very much comparable to DeepAR and TFT. However, note that the existing algorithms are optimized to take advantage of GPU, the proposed approach does not, it runs only on the CPU. It may be possible to improve the performance in the future through parallelization.

5.3.4.3 Uncertainty quantification

Table 5.7: (Un)certainty quantification (log-loss) and Cryptocurrency Volatility Index (CVI) evaluated using DeCrypt (3 layers) and Nbeats

Cryptocurrency	DeCrypt (1 layer)	DeCrypt (2 layer)	DeCrypt (3 layers)	CVI (DeCrypt*))	CVI (Nbeats)
Bitcoin	0.79	**0.67**	0.86	**0.31**	0.48
Dogecoin	**4.32**	4.89	4.75	**2.62**	2.71
Namecoin	4.34	3.79	**3.65**	1.61	1.81
Litecoin	1.21	1.10	**0.92**	**0.35**	0.49
Gridcoin	1.81	1.56	**1.45**	0.59	0.63
Peercoin	1.67	1.43	**1.23**	0.50	0.59
Ripple	1.24	1.11	**0.97**	**0.42**	0.53
NXT	1.32	1.10	**0.91**	**0.32**	0.42
Ethereum	1.56	**1.41**	1.43	**0.38**	0.63
Binance coin	1.34	1.21	**0.84**	**0.33**	0.72

The advantage of DeCrypt over other baseline approaches is the estimation of (un)certainty quantification associated with each prediction. For cryptocurrencies

Table 5.8: Cryptocurrency Volatility Index (CVI) evaluated using state-of-the-art method predictions

Cryptocurrency	LSTM	CNN-TA	ARIMA	Rao-ANN	MFNN	DeepAR	TFT	RDL
Bitcoin	0.57	0.53	0.46	0.59	0.72	0.52	0.49	0.38
Dogecoin	2.87	3.27	2.35	3.43	3.87	2.72	2.83	2.57
Namecoin	1.93	2.12	1.88	1.83	1.96	**1.58**	1.68	1.73
Litecoin	1.31	0.56	0.48	0.51	0.67	0.38	0.46	0.41
Gridcoin	0.99	0.87	0.67	0.74	0.96	0.68	0.75	**0.53**
Peercoin	0.81	0.78	**0.48**	0.69	0.93	0.64	0.71	0.58
Ripple	0.78	0.83	0.64	0.73	0.88	0.61	0.68	0.54
NXT 0.57	0.63	0.46	0.53	0.64	0.58	0.54	0.59	0.47
Ethereum	0.98	0.78	0.43	0.58	0.49	0.52	0.61	0.45
Binance coin	0.53	0.64	0.41	0.49	0.51	0.45	0.54	0.39

this measure will be directly proportional to the volatility index. As discussed in section *Uncertainty Quantification*, it is easy to evaluate (un)certainty of prediction of an increase/decrease of price forecast by calculating the log-loss penalization as explained in eq. 5.18. To validate the proposed method results on (un)certainty quantification the work also evaluated cryptocurrency volatility index for each cryptocurrency. Cryptocurrecy Volatility Index (CVI) [149, 150]can be defined as a measure of market's expectation of volatility over the near trading terms for a particular asset. Volatility is often described as the "rate and magnitude of changes in prices" and in finance often referred to as risk. Volatility is sometimes associated with the uncertainty of risk related to the amount of changes in security's value. This can be further described as if the security's value can potentially be spread out over a larger range of values, it indicates that the price of the security can change dramatically over a short time period in either direction which is flagged as higher volatility. On the other hand, A lower volatility means that a security's value does not fluctuate dramatically,

and tends to be more steady. The mathematical formula to calculate CVI [150]:

$$\text{CVI} = \sqrt{365} * \sqrt{\frac{1}{N}\sum_{N=1}^{N}((Closeprice - PriceatN)^2))}, \tag{5.19}$$

Table 5.7 depicts the calculated log-loss values for each cryptocurrency vis-a-vis their cryptocurrency volatility index (CVI)[3].Table 5.7 represents the log loss score for all the layer architecture for DeCrypt and CVI score for DeCrypt (3 layers) and Nbeats. Table 5.8 represents the CVI scores from state-of-the-art method. A smaller value of the loss should be associated with lower volatility and vice versa. It can be clearly seen that log-loss associated with the Bitcoin, Litecoin, Peercoin, Ripple, NXT, Binance coin is less than one, meaning prices associated with these cryptocurrencies are less volatile. In contrast, Dogecoin, which is highly volatile and has a history of spiked values after a tweet by a major influencer, is more difficult to assess and has a log-loss score of 4.75. Thus one can see how the proposed algorithm can quantify uncertainty and how this measure is proportionate to the oracle volatility. This is by far the most important result in this work. This result shows how the proposed approach may be used for practical trading where both the point estimate as well as the uncertainty about the estimate is required for making decisions.

Owing to limitations in space, unfortunately not able to show the convergence of the proposed algorithm or the run-times of different techniques. Although the work have used 50 iterations and its empirically checked that the proposed algorithm converges in about 20-25 iterations. The convergence is monotonic.

[3] https://github.com/dc-aichara/PriceIndices

In terms of speed the proposed method is about 2-4 times faster than LSTM, CNN-TA and MFNN, and is about an order of magnitude faster than TFT, Nbeats and DeepAR. Of the existing methods, only RDL is comparable to the proposed method in terms of speed.

5.3.5 Discussion

Consistently beating in modeling Time series signals has been challenging for a long time. The proposed three-layer architecture method performed better than the current state-of-the-art method. The proposed method is based on the deep state space and feedback strategy. As can be seen from Fig. 5.2 the proposed model can predict the sudden spikes in the data compared to other state-of-the-art methods. This is mainly because the proposed method is based on SSM, which uses probabilistic predictive distribution to estimate the future state of the price trajectories. The technique also embeds deep non-negative factors to learn the model parameters. Deep factors helps in updating the model parameters and state space of unseen signals continuously with time, in contrast to machine learning models, which use a huge amount of data to learn approximations. We observed that when the time series signals grow, these models suffer from vanishing gradient and exploding gradient problems, which hampers learning model parameters. Due to this, model approximations are not learned efficiently and cannot capture sudden spikes (highs and lows in prices). These models also suffer from over-fitting. The proposed method avoids over-fitting as we move ahead in the sliding window protocol, and previously updated model parameters

are used as initialized values for the next window parameters. We have also presented a comprehensive analysis of empirical and statistical performance. When Table 5.4 and Table 5.6 are analyzed, it is observed that the proposed method performance is outstanding in 3 out of 4 crypto-currency presented, and its average score for ten crypto-currency is smallest as compared to other state-of-the-art methods hence we can conclude that more similarity exists between the actual closing prices and predicted closing prices when compared for different Crypto-currencies. The proposed method can be applied to various other prediction applications where its challenging to model unseen volatile data, such as short-term load monitoring, sales and revenue prediction, and predicting hate intensity for social media content.

Chapter 6

Conclusion

This thesis addressed several bottlenecks (discussed the chapters) in the area of dynamically modeling time-series signals. We have tried to propose solutions to overcome these bottlenecks to make the process more practical and scalable.

6.1 Summary of Contribution

The thesis contributions are outlined below:

- We proposed new modeling and inferential tool for dynamical processing of time series. The approach is called recurrent dictionary learning (RDL). The proposed model reads as a linear Gaussian Markovian state-space model involving two linear operators, the state evolution and the observation matrices that we assumed unknown. These two unknown operators (that can be seen interpreted as dictionaries) and the sequence of hidden states are jointly learnt via an expectation-maximization algorithm. Experimental

results show that our proposed method excels over state-of-the-art stock analysis models such as CNN-TA, MFNN, and LSTM.

- Next, we proposed sequential transform learning (STL). The proposed work is a linear Gaussian Markovian state-space model involving state evolution, observation matrices, and an exogenous control input. The resultant formulation resembles loosely based on transform learning. Furthermore, the formulation is made supervised by the label consistency cost. The approach differs from RDL due to presence of an exogenous input, which helps to establish more informed prior for state-space evolution. Benchmarking with the state-of-the-art has shown that our method excels over the rest.

- Further, we proposed Deep recurrent dictionary learning (DRDL). The proposed work is developed to cater to bottlenecks experienced in recurrent dictionary learning approach. The work overcomes the limitations of RDL and also dives into multi-linear Gaussian state space. In this work, we combine the benefits of both approaches(signal processing and neural network) by introducing a multi-linear Gaussian SSM whose state and evolution operators can be learnt from the data. We propose factorized forms for the state and evolution operators to cope with possible non-linearity in the observed data and the hidden state. Numerical experiments on a problem of stock market data inference shows its superiority among several state-of-the-art dynamic modeling tools.

- Finally, we proposed Deep sequential transform learning (DSTL). The pro-

posed method is a deep network to model multi-linear Gaussian state space in the presence of an exogenous input inherited from sequential transform learning. The work modeled non-linearity using a deep factor model. The method is developed to overcome the limitations of the Sequential transform learning model and explore the multi-linear Gaussian state-space model in the presence of exogenous input, which differentiates it from the DRDL approach. Our proposed DSTL, when applied to predicting cryptocurrency prices, improves over state-of-the-art deep learning models by a considerable margin.

6.2 Future Work

In this section, we discuss the further advancements of the thesis work.

The approaches discussed in chapter2 to chapter4 inherits advantages from sophisticated modeling techniques while quantifying the uncertainty in the estimates. We have applied it for the processing of challenging financial time series involved in stock forecasting and stock trading tasks. The results show that the proposed method outperforms the state-of-the-art techniques.Given these promising results, we plan as future work to delve deeper into the area of financial forecasting, including the application of our technique in forecasting derivatives.

The study dicussed in chapter5 proposes a novel approach to forecast the prices of cryptocurrencies; this is halfway to the goal. The final objective is to take trading positions (BUY / SELL / HOLD). For that, we would need to define

thresholds. This is a matured area in traditional financial markets [151] where game theory is mainly used in arriving at the decision boundaries. Currently, a combination of game theory and social network analysis is used for arriving at such decision boundaries [152]. In future, we would like to see if such cues from stock trading can be used for maximising returns in cryptocurrency trading.

References

[1] J. D. Cryer and K.-S. Chan, *Time series analysis: with applications in R*. Springer, 2008, vol. 2.

[2] R. H. Shumway and D. S. Stoffer, "An approach to time series smoothing and forecasting using the EM algorithm," *Journal of Time Series Analysis*, vol. 3, no. 4, pp. 253–264, 1982.

[3] D. Shah, H. Isah, and F. Zulkernine, "Stock market analysis: A review and taxonomy of prediction techniques," *International Journal of Financial Studies*, vol. 7, no. 2, p. 26, 2019.

[4] E. Fama, "Random walks in stock market prices," *Financial Analysts Journal*, vol. 51, no. 1, pp. 75–80, 1995.

[5] B. Malkiel and E. Fama, "Efficient capital markets: A review of theory and empirical work," *The journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970.

[6] Y. Kwon and B. Moon, "A hybrid neurogenetic approach for stock forecasting," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 851–864, 2007.

[7] G. Atsalakis and K. Valavanis, "Surveying stock market forecasting techniques–part ii: Soft computing methods," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5932–5941, 2009.

[8] P. Pai and C. Lin, "A hybrid arima and support vector machines model in stock price forecasting," *Omega*, vol. 33, no. 6, pp. 497–505, 2005.

[9] G. Atsalakis and K. Valavanis, "Surveying stock market forecasting techniques-part i: Conventional methods," *Journal of Computational Optimization in Economics and Finance*, vol. 2, no. 1, pp. 45–92, 2010.

[10] A. A. Ariyo, A. Adewumi, and C. Ayo, "Stock price prediction using the ARIMA model," in *Proceedings of the 16th International Conference on Computer Modelling and Simulation*. IEEE, 2014, pp. 106–112.

[11] P. Mondal, L. Shit, and S. Goswami, "Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices," *International Journal of Computer Science, Engineering and Applications*, vol. 4, no. 2, p. 13, 2014.

[12] J. Jarrett and E. Kyper, "ARIMA modeling with intervention to forecast and analyze chinese stock prices," *International Journal of Engineering Business Management*, vol. 3, no. 3, pp. 53–58, 2011.

[13] B. Devi, D. Sundar, and P. Alli, "An effective time series analysis for stock trend prediction using ARIMA model for nifty midcap-50," *International Journal of Data Mining & Knowledge Management Process*, vol. 3, no. 1, p. 65, 2013.

[14] A. Petricua, S. Stancu, and A. Tindeche, "Limitation of arima models in financial and monetary economics." *Theoretical & Applied Economics*, vol. 23, no. 4, 2016.

[15] S. Makridakis and M. Hibon, "Arma models and the box–jenkins methodology," *Journal of Forecasting*, vol. 16, no. 3, pp. 147–163, 1997.

[16] T. M. O'Donovan, *Short term forecasting: An introduction to the Box-Jenkins approach.* JOHN WILEY & SONS, 1983.

[17] S. Chadsuthi, C. Modchang, Y. Lenbury, S. Iamsirithaworn, and W. Triampo, "Modeling seasonal leptospirosis transmission and its association with rainfall and temperature in thailand using time–series and arimax analyses," *Asian Pacific Journal of Tropical Medicine*, vol. 5, no. 7, pp. 539–546, 2012.

[18] S. Sarkka, *Bayesian Filtering and Smoothing*, 3rd ed., C. U. Press, Ed., 2013.

[19] R. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[20] J. P. Nóbrega and A. L. O., "A sequential learning method with Kalman filter and extreme learning machine for regression and time series forecasting," *Neurocomputing*, vol. 337, pp. 235–250, 2019.

[21] N. Saini and A. Mittal, "Forecasting volatility in indian stock market using state space models," *Journal of Statistical and Econometric Methods*, vol. 3, no. 1, pp. 115–136, 2014.

[22] Y. Zeng and S. Wu, *State-space models: Applications in economics and finance*. Springer, 2013, vol. 1.

[23] E. Zivot, J. Wang, and S. Koopman, "State space modelling in macroeconomics and finance using ssfpack in s+ finmetrics," *State Space and Unobserved Component Models*, pp. 284–335, 2004.

[24] L. Ljung, "Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 1, pp. 36–50, 1979.

[25] C. Slim, "Neuro-fuzzy network based on extended kalman filtering for financial time series," *World Academy of Science, Engineering and Technology*, vol. 22, pp. 134–139, 2006.

[26] E. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, 2000, pp. 153–158.

[27] S. Yang and H. Li, "Application of ekf and ukf in target tracking problem," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 1. IEEE, 2016, pp. 116–120.

[28] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Transactions on Signal Processing*, vol. 50, no. 3, pp. 736–746, 2002.

[29] P. Djuric, J. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. Bugallo, and J. Miguez, "Particle filtering," *IEEE Signal Processing Magazine*, vol. 20, no. 5, pp. 19–38, 2003.

[30] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[31] R. Casarin, C. Trecroci *et al.*, *Business cycle and stock market volatility: A particle filter approach*. Università degli studi, Dipartimento di scienze economiche, 2006.

[32] A. Javaheri, D. Lautier, and A. Galli, "Filtering in finance," *Wilmott*, vol. 3, pp. 67–83, 2003.

[33] J. Smith and A. Santos, "Second-order filter distribution approximations for financial time series with extreme outliers," *Journal of Business & Economic Statistics*, vol. 24, no. 3, pp. 329–337, 2006.

[34] V. Elvira, L. Martino, M. F. Bugallo, and P. Djurić, "In search for improved auxiliary particle filters," in *Signal Processing Conference (EUSIPCO), 2018 Proceedings of the 26th European*. IEEE, 2018, pp. 1–5.

[35] V. Elvira, L. Martino, M. F. Bugallo, and P. M. Djuric, "Elucidating the auxiliary particle filter via multiple importance sampling [lecture notes]," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 145–152, 2019.

[36] V. Elvira, J. Míguez, and P. Djurić, "Adapting the number of particles in sequential monte carlo methods through an online scheme for convergence

assessment," *IEEE Transactions on Signal Processing*, vol. 65, no. 7, pp. 1781–1794, 2017.

[37] ——, "New results on particle filters with adaptive number of particles," *arXiv preprint arXiv:1911.01383*, 2017.

[38] A. Harvey, *Forecasting, structural time series models and the Kalman filter*, C. U. Press, Ed., 1990.

[39] M. Eichler, "Graphical modelling of multivariate time series," *Probability Theory and Related Fields*, vol. 153, no. 1, pp. 233–268, Jun. 2012. [Online]. Available: https://doi.org/10.1007/s00440-011-0345-8

[40] F. R. Bach and M. I. Jordan, "Learning graphical models for stationary time series," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2189–2199, Aug. 2004.

[41] D. Barber and A. T. Cemgil, "Graphical models for time-series," *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 18–28, Nov 2010.

[42] E. Chouzenoux and V. Elvira, "GraphEM: EM algorithm for blind Kalman filtering under graphical sparsity constraints," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, 2020, pp. 1–5.

[43] M. Abe and H. Nakayama, "Deep learning for forecasting stock returns in the cross-section," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2018, pp. 273–284.

[44] R. Singh and S. Srivastava, "Stock prediction using deep learning," *Multimedia Tools and Applications*, vol. 76, no. 18, pp. 18 569–18 584, 2017.

[45] D. Lien Minh, A. Sadeghi-Niaraki, H. D. Huy, K. Min, and H. Moon, "Deep learning approach for short-term stock trends prediction based on two-stream gated recurrent unit network," *IEEE Access*, vol. 6, pp. 55 392–55 404, 2018.

[46] Y. Xu and S. Cohen, "Stock movement prediction from tweets and historical prices," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2018, pp. 1970–1979.

[47] Z. Hu, W. Liu, J. Bian, X. Liu, and T. Liu, "Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction," in *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. ACM, 2018, pp. 261–269.

[48] G. Zhang, *Neural Networks in Business Forecasting*, ser. IGI Global. Journal of Intelligent Systems, 2004, vol. 6.

[49] H. Kim and K. Shin, "A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets," *Applied Soft Computing*, vol. 7, no. 2, pp. 569–576, 2007.

[50] M. R. Hassan, "A combination of hidden Markov model and fuzzy model for stock market forecasting," *Neurocomputing*, vol. 72, no. 16-18, pp. 3439–3446, 2009.

[51] X. Zhong and D. Enke, "A comprehensive cluster and classification mining procedure for daily stock market return forecasting," *Neurocomputing*, vol. 267, pp. 152–168, 2017.

[52] M. Hiransha, E. Gopalakrishnan, V. K. Menon, and K. Soman, "NSE stock market prediction using deep-learning models," *Procedia Computer Science*, vol. 132, pp. 1351–1362, 2018.

[53] M. Kumar and M. Thenmozhi, "Forecasting stock index returns using ARIMA-SVM, ARIMA-ANN, and ARIMA-random forest hybrid models," *International Journal of Banking, Accounting and Finance*, vol. 5, no. 3, pp. 284–308, 2014.

[54] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

[55] R. Akita, A. Yoshihara, T. Matsubara, and K. Uehara, "Deep learning for stock prediction using numerical and textual information," in *Proceedings of the 15th International Conference on Computer and Information Science (ICIS 2016)*. IEEE, 2016, pp. 1–6.

[56] E. Chong, C. Han, and F. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017.

[57] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[58] T. Gao and Y. Chai, "Improving stock closing price prediction using recurrent neural network and technical indicators," *Neural Computation*, vol. 30, no. 10, pp. 2833–2854, 2018.

[59] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, pp. 1–12, 2018.

[60] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, p. 318–362.

[62] D. Nelson, A. Pereira, and R. de Oliveira, "Stock market's price movement prediction with LSTM neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2017)*. IEEE, 2017, pp. 1419–1426.

[63] A. Samarawickrama and T. Fernando, "A recurrent neural network approach in predicting daily stock prices an application to the Sri Lankan

stock market," in *Proceedings of the IEEE International Conference on Industrial and Information Systems (ICIIS 2017)*, 2017, pp. 1–6.

[64] Z. Shen, Y. Zhang, J. Lu, J. Xu, and G. Xiao, "A novel time series forecasting model with deep learning," *Neurocomputing*, vol. 396, pp. 302–313, 2020.

[65] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in *Proceedings of the 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC 2016)*. IEEE, 2016, pp. 324–328.

[66] Y. Li, Z. Zhu, D. Kong, H. Han, and Y. Zhao, "Ea-lstm: Evolutionary attention-based lstm for time series prediction," *Knowledge-Based Systems*, vol. 181, p. 104785, 2019.

[67] K. Althelaya, E. El-Alfy, and S. Mohammed, "Evaluation of bidirectional LSTM for short-and long-term stock market prediction," in *Proceedings of the 9th international conference on information and communication systems (ICICS 2018)*. IEEE, 2018, pp. 151–156.

[68] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 114–119, 2019.

[69] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-d convolutional neural networks," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 3, pp. 664–675, 2015.

[70] K. Kashiparekh, J. Narwariya, P. Malhotra, L. Vig, and G. Shroff, "Convtimenet: A pre-trained deep convolutional neural network for time series classification," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2019)*. IEEE, 2019, pp. 1–8.

[71] D. Smirnov and E. Nguifo, "Time series classification with recurrent neural networks," *Advanced Analytics and Learning on Temporal Data*, p. 8, 2018.

[72] S. Griffith-Jones, M. Segoviano, and S. Spratt, "Basel II and developing countries: diversification and portfolio effects," LSE Financial Markets Group, Tech. Rep. 34, 2003, discussion paper 437.

[73] V. Kumar and D. Shah, "Expanding the role of marketing: from customer equity to market capitalization," *Journal of Marketing*, vol. 73, no. 6, pp. 119–136, 2009.

[74] I. Tosic and P. Frossard, "Dictionary learning: What is the right representation for my signal?" *IEEE Signal Processing Magazine*, vol. 28, pp. 27–38, 2011.

[75] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding." *Journal of Machine Learning Research*, vol. 11, no. 1, 2010.

[76] X. Li, H. Shen, L. Zhang, H. Zhang, Q. Yuan, and G. Yang, "Recovering quantitative remote sensing products contaminated by thick clouds and

shadows using multitemporal dictionary learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 11, pp. 7086–7098, 2014.

[77] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 4, pp. 791–804, 2011.

[78] Y. Xu, Z. Li, J. Yang, and D. Zhang, "A survey of dictionary learning algorithms for face recognition," *IEEE access*, vol. 5, pp. 8502–8514, 2017.

[79] Y. Xu, Z. Li, B. Zhang, J. Yang, and J. You, "Sample diversity, representation effectiveness and robust dictionary learning for face recognition," *Information Sciences*, vol. 375, pp. 171–182, 2017.

[80] Z. Li, Z. Lai, Y. Xu, J. Yang, and D. Zhang, "A locality-constrained and label embedding dictionary learning algorithm for image classification," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 2, pp. 278–293, 2015.

[81] X. Zhang and F. Ding, "Adaptive parameter estimation for a general dynamical system with unknown states," *International Journal of Robust and Nonlinear Control*, vol. 30, no. 4, pp. 1351–1372, 2020.

[82] D. Luengo, L. Martino, M. Bugallo, V. Elvira, and S. Särkkä, "A survey of monte carlo methods for parameter estimation," *EURASIP Journal on Advances in Signal Processing*, vol. 2020, no. 1, pp. 1–62, 2020.

[83] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, no. 3.

[84] N. Thacker and A. Lacey, "Tutorial: The kalman filter," *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, p. 61, 1998.

[85] E. Benhamou, "Kalman filter demystified: from intuition to probabilistic graphical model to real case in financial markets," *arXiv preprint arXiv:1811.11618*, 2018.

[86] M. P. Deisenroth and H. Ohlsson, "A probabilistic perspective on gaussian filtering and smoothing," *arXiv preprint arXiv:1006.2165*, 2010.

[87] P. Gupta, J. Maggu, A. Majumdar, E. Chouzenoux, and G. Chierchia, "Deconfuse: a deep convolutional transform-based unsupervised fusion framework," *EURASIP Journal on Advances in Signal Processing*, no. 1, pp. 1–32, 2020.

[88] M. Pereyra, P. Schniter, E. Chouzenoux, J.-C. Pesquet, J. Tourneret, A. Hero, and S. McLaughlin, "A survey of stochastic simulation and optimization methods in signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 224–241, 2016.

[89] C. F. J. Wu, "On the convergence properties of the EM algorithm," *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, 1983.

[90] E. Chouzenoux, A. Jezierska, J. Pesquet, and H. Talbot, "A majorize-minimize subspace approach for l2-l0 image regularization," *SIAM Journal on Imaging Science*, vol. 6, no. 1, pp. 563–591, 2013.

[91] E. Chouzenoux and J. Pesquet, "A stochastic majorize-minimize subspace algorithm for online penalized least squares estimation," *IEEE Transactions on Signal Processing*, vol. 65, no. 18, pp. 4770–4783, 2017.

[92] O. Sezer and A. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018.

[93] W. Long, Z. Lu, and L. Cui, "Deep learning-based feature engineering for stock price movement prediction," *Knowledge-Based Systems*, vol. 164, pp. 163–173, 2019.

[94] J. W. Hall, "Adaptive selection of us stocks with neural nets," *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets. New York: Wiley*, pp. 45–65, 1994.

[95] A. Timmermann, "Elusive return predictability," *International Journal of Forecasting*, vol. 24, no. 1, pp. 1–18, 2008.

[96] A. W. Lo, "Reconciling efficient markets with behavioral finance: the adaptive markets hypothesis," *Journal of investment consulting*, vol. 7, no. 2, pp. 21–44, 2005.

[97] R. Lawrence, "Using neural networks to forecast stock market prices," Tech. Rep., 1997, https://people.ok.ubc.ca/rlawrenc/research/Papers/nn.pdf.

[98] X. Zhong and D. Enke, "Forecasting daily stock market return using dimensionality reduction," *Expert Systems with Applications*, vol. 67, pp. 126–139, 2017.

[99] Y. Abu-Mostafa and A. Atiya, "Introduction to financial forecasting," *Applied Intelligence*, vol. 6, no. 3, pp. 205–213, 1996.

[100] K. F. Bannör and M. Scherer, "Model risk and uncertainty—illustrated with examples from mathematical finance," in *Risk-A Multidisciplinary Introduction*. Springer, 2014, pp. 279–306.

[101] C. Gollier, *The economics of risk and uncertainty*. Edward Elgar Publishing Limited, 2018.

[102] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms," *IEEE Transactions on Signal Processing*, vol. 61, no. 5, pp. 1072–1086, 2012.

[103] J. Maggu, H. K. Aggarwal, and A. Majumdar, "Label-consistent transform learning for hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, 2019.

[104] S. Liang and R. Srikant, "Why deep neural networks for function approximation?" *arXiv preprint arXiv:1610.04161*, 2016.

[105] D. Elbrächter, D. Perekrestenko, P. Grohs, and H. Bölcskei, "Deep neural network approximation theory," *arXiv preprint arXiv:1901.02220*, 2019.

[106] P. Cheridito, A. Jentzen, and F. Rossmannek, "Efficient approximation of high-dimensional functions with neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[107] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[108] Q. Yang and X. Wu, "10 challenging problems in data mining research," *International Journal of Information Technology & Decision Making*, vol. 5, no. 04, pp. 597–604, 2006.

[109] E. F. Fama, "Efficient capital markets a review of theory and empirical work," *The Fama Portfolio*, pp. 76–121, 2021.

[110] S. Sharma, V. Elvira, E. Chouzenoux, and A. Majumdar, "Recurrent dictionary learning for state-space models with an application in stock forecasting," vol. 450. Elsevier, 2021, pp. 1–13.

[111] S. Sharma, A. Majumdar, V. Elvira, and E. Chouzenoux, "Blind kalman filtering for short-term load forecasting," *IEEE Transactions on Power Systems*, vol. 35, no. 6, pp. 4916–4919, 2020.

[112] E. Chouzenoux and V. Elvira, "Graphem: Em algorithm for blind kalman filtering under graphical sparsity constraints," in *ICASSP 4 May 2020-*

8 May 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona. IEEE, 2020, pp. 5840–5844.

[113] P. De Handschutter, N. Gillis, and X. Siebert, "A survey on deep matrix factorizations," *Computer Science Review*, vol. 42, p. 100423, 2021.

[114] A. Cichocki, R. Zdunek, A. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley-Blackwell, 2009.

[115] L. Yu, C. Liu, and Z.-K. Zhang, "Multi-linear interactive matrix factorization," *Knowledge-Based Systems*, vol. 85, pp. 307–315, 2015.

[116] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. W. Schuller, "A deep matrix factorization method for learning attribute representations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 3, pp. 417–429, 2016.

[117] H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems." in *IJCAI*, vol. 17. Melbourne, Australia, 2017, pp. 3203–3209.

[118] Z. Chen, S. Jin, R. Liu, and J. Zhang, "A deep non-negative matrix factorization model for big data representation learning," *Frontiers in Neurorobotics*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbot.2021.701194

[119] J. Flenner and B. Hunter, "A deep non-negative matrix factorization neural network," *Semantic Scholar*, 2017, https://www1.cmc.edu/pages/faculty/BHunter/papers/deep-negative-matrix.pdf.

[120] B. Liu and Y. Liang, "Optimal function approximation with relu neural networks," *Neurocomputing*, vol. 435, pp. 216–227, 2021.

[121] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, "Nonlinear approximation and (deep) relu networks," *Constructive Approximation*, vol. 55, no. 1, pp. 127–172, 2022.

[122] M. Chen, H. Jiang, W. Liao, and T. Zhao, "Efficient approximation of deep relu networks for functions on low dimensional manifolds," *Advances in neural information processing systems*, vol. 32, pp. 8174–8184, 2019.

[123] S. Tariyal, A. Majumdar, R. Singh, and M. Vatsa, "Deep dictionary learning," *IEEE Access*, vol. 4, pp. 10 096–10 109, 2016.

[124] S. Mahdizadehaghdam, A. Panahi, H. Krim, and L. Dai, "Deep dictionary learning: A parametric network approach," *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 4790–4802, 2019.

[125] C. Andrieu, A. Doucet, and R. Holenstein, "Particle markov chain monte carlo methods," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 72, no. 3, pp. 269–342, 2010.

[126] N. Chopin, P. E. Jacob, and O. Papaspiliopoulos, "SMC2: an efficient algorithm for sequential analysis of state space models," *Journal of the*

Royal Statistical Society: Series B (Statistical Methodology), vol. 75, no. 3, pp. 397–426, 2013.

[127] D. Crisan and J. Miguez, "Nested particle filters for online parameter estimation in discrete-time state-space markov models," *Bernoulli*, vol. 24, no. 4A, pp. 3039–3086, 2018.

[128] M. W. Jacobson and J. A. Fessler, "An expanded theoretical treatment of iteration-dependent majorize-minimize algorithms," *IEEE Transactions on Image Processing*, vol. 16, no. 10, pp. 2411–2422, 2007.

[129] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018.

[130] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[131] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-beats: Neural basis expansion analysis for interpretable time series forecasting," *arXiv preprint arXiv:1905.10437*, 2019.

[132] B. Lim, S. Ö. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *CoRR*, vol. abs/1912.09363, 2019. [Online]. Available: http://arxiv.org/abs/1912.09363

[133] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[134] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 12, pp. 2303–2308, 2014.

[135] S. Sharma and A. Majumdar, "Sequential transform learning," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 5, pp. 1–18, 2021.

[136] R. Molla, "When elon musk tweets, crypto prices move," https://www.vox.com/recode/2021/5/18/22441831/elon-musk-bitcoin-dogecoin-crypto-prices-tesla, 2021, accessed: 2021-06-14.

[137] S. Soni, "Crypto investors lost $748 billion in last seven days as bitcoin, ethereum, dogecoin, others declined," https://www.financialexpress.com/market/crypto-investors-lost-748-billion-in-last/seven-days-as-bitcoin-ethereum-dogecoin-others-declined, 2021, accessed: 2021-05-23.

[138] V. Digalakis, J. R. Rohlicek, and M. Ostendorf, "Ml estimation of a stochastic linear system with the em algorithm and its application to speech recognition," *IEEE Transactions on speech and audio processing*, vol. 1, no. 4, pp. 431–442, 1993.

[139] Y. Wang, W. Yin, and J. Zeng, "Global convergence of admm in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, vol. 78, no. 1, pp. 29–63, 2019.

[140] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A general analysis of the convergence of admm," in *International Conference on Machine Learning*. PMLR, 2015, pp. 343–352.

[141] T. Lin, S. Ma, and S. Zhang, "On the global linear convergence of the admm with multiblock variables," *SIAM Journal on Optimization*, vol. 25, no. 3, pp. 1478–1497, 2015.

[142] J. Mei, Y. De Castro, Y. Goude, J.-M. Azaïs, and G. Hébrail, "Nonnegative matrix factorization with side information for time series recovery and prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 493–506, 2019.

[143] D. Yarotsky, "Optimal approximation of continuous functions by very deep relu networks," in *Conference on Learning Theory*. PMLR, 2018, pp. 639–649.

[144] C. Montella, "The kalman filter and related algorithms: A literature review," *Research Gate*, 2011.

[145] V. Elvira, J. Míguez, and P. M. Djurić, "Adapting the number of particles in sequential monte carlo methods through an online scheme for convergence assessment," *IEEE Transactions on Signal Processing*, vol. 65, no. 7, pp. 1781–1794, 2017.

[146] S. Elsworth and S. Güttel, "Time series forecasting using lstm networks: A symbolic approach," *arXiv preprint arXiv:2003.05672*, 2020.

[147] S. K. Nayak, S. C. Nayak, and S. Das, "Modeling and forecasting cryptocurrency closing prices with rao algorithm-based artificial neural networks: A machine learning approach," *FinTech*, vol. 1, no. 1, pp. 47–62, 2021.

[148] N. Abu Bakar and S. Rosbi, "Autoregressive integrated moving average (arima) model for forecasting cryptocurrency exchange rate in high volatility environment: A new insight of bitcoin transaction," *International Journal of Advanced Engineering Research and Science*, vol. 4, no. 11, pp. 130–137, 2017.

[149] A. Kim, S. Trimborn, and W. K. Härdle, "Vcrix—a volatility index for crypto-currencies," *International Review of Financial Analysis*, vol. 78, p. 101915, 2021.

[150] F. Woebbeking, "Cryptocurrency volatility markets," *Digital Finance*, pp. 1–26, 2021.

[151] R. B. Shelton, *Gaming the market: Applying game theory to create winning trading strategies*. John Wiley & Sons, 1997, vol. 69.

[152] X. Molinero and F. Riquelme, "Influence decision models: from cooperative game theory to social network analysis," *Computer Science Review*, vol. 39, p. 100343, 2021.