END-TO-END HYBRID ARCHITECTURES FOR EFFECTIVE SEQUENTIAL DATA PREDICTION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Mustafa Enes Aydın

August 2023

END-TO-END HYBRID ARCHITECTURES FOR EFFECTIVE SEQUENTIAL DATA PREDICTION

By Mustafa Enes Aydın

August 2023

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Sinan Gezici

Umut Orguner

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

END-TO-END HYBRID ARCHITECTURES FOR EFFECTIVE SEQUENTIAL DATA PREDICTION

Mustafa Enes Aydın
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
August 2023

We investigate nonlinear prediction in an online setting and introduce two hybrid models that effectively mitigate, via end-to-end architectures, the need for hand-designed features and manual model selection issues of conventional nonlinear prediction/regression methods. Particularly, we first use an enhanced recurrent neural network (LSTM) to extract features from sequential signals, while preserving the state information, i.e., the history, and soft gradient boosted decision trees (sGBDT) to produce the final output. The connection is in an end-to-end fashion and we jointly optimize the whole architecture using stochastic gradient descent. Secondly, we again use recursive structures (LSTM) for automatic feature extraction out of raw data but accompany it with a traditional linear time series model (SARIMAX) to deal with the intricacies of the sequential data, e.g., seasonality. The unification of the models is again in a joint manner; it is through a single state space and we optimize the entire architecture using particle filtering. The proposed frameworks are generic so that one can use other recurrent architectures, e.g., GRUs, and differentiable machine learning algorithms as well as time series models that have state space representations in lieu of the specific models presented. We demonstrate the learning behavior of the models on synthetic data and the significant performance improvements over the conventional methods and the disjoint counterparts over various real life datasets, with which we also show the generic nature of the frameworks. Furthermore, we openly share the source code of the proposed methods to facilitate further research.

Keywords: Online learning, prediction, time series, end-to-end learning, long short-term memory (LSTM), soft gradient boosting decision tree (sGBDT), seasonal auto-regressive integrated moving average with exogenous regressors (SARIMAX).

ÖZET

ETKİLİ ARDIŞIK VERİ TAHMİNİ İÇİN UÇTAN UCA MELEZ MİMARİLER

Mustafa Enes Aydın
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Ağustos 2023

Çevrimiçi öğrenim ile doğrusal olmayan tahminlemeyi çalıştığımız bu tezde, iki adet etkili melez öğrenme metodu sunmaktayız. Bu metotlar, uçtan uca mimariler olup etkili bir şekilde geleneksel algoritmaların zaaflarından olan ihtisasa ve elle ayarlamaya dayalı öznitelik ve model seçimini ortadan kaldırmaktadır. Ayrıntılı olarak, ilk modelimizde gelişmiş bir yinelemeli yapay sinir ağı (Uzun-Kısa-Soluklu Bellek) kullanarak sıralı ham veriden öznitelik özümlemesini otomatik olarak gerçekleştirirken, yumuşak (türevlenebilir) gradyan destekli karar ağaçlarıyla nihai tahmini etkili bir şekilde sağlamaktayız. Bu iki taban modelin birbirine bağlanması uçtan uca olup tüm mimari olasılıksal gradyan inişi (OGİ) ile ortaklaşa eğitilmektedir. İkinci modelimizde ise yine yinelemeli yapıları (UKSB) kullanarak ham veriden öznitelik çıkarımını özedimli hale getirirken, yanına zaman serileri için özel olarak tasarlanmış geleneksel bir model olan harici değişkenli ve tümlevli mevsimsel otoregresif hareketli ortalamalar modelini koyarak ardıl verilere has mevsimsellik gibi çetrefilliklere daha etkili bir şekilde hitap etmekteyiz. Bu iki modelin çalışması yine ayrışık olmayıp tüm mimariyi tek bir durum uzayında toplayıp parçacık süzme (PS) tekniği ile eğitimini gerçekleştirmekteyiz. Sunduğumuz her iki melez mimari de jenerik olup istenirse UKSB yerine, örneğin, geçitli yinelemeli ünite (GYÜ) gibi modeller kullanılabileceği gibi, diğer bileşenler için de herhangi bir türevlenebilir makine öğrenmesi modeli ve durum uzayı temsiline sahip bir geleneksel zaman serisi modeli de kullanılabilmektedir. Ortaya koyduğumuz mimarilerin öğrenme davranışlarını yapay veri setleri ile gösterirken, hem diğer geleneksel metotlara hem de ayrışık hallerine kıyasla elde ettikleri kaydadeğer performans artışlarını da çeşitli gerçek hayat verileri ile sunuyoruz; bu verilerle aynı zamanda modellerin jenerikliğini de göstermekteyiz. Ayriyeten, ortaya konulan modellerin kaynak kodlarını da açık bir şekilde paylaşmaktayız.

Anahtar sözcükler: Çevrimiçi öğrenme, tahminleme, zaman serileri, uçtan uca öğrenme, uzun-kısa-soluklu bellek, yumuşak (türevlenebilir) gradyan destekli karar ağaçları, harici değişkenli ve tümlevli mevsimsel otoregresif hareketli ortalamalar.

Acknowledgement

I would like to thank my supervisor Prof. Süleyman Serdar Kozat, for their expertise was precious in both theoretical and practical aspects of the master's studies. I would like to thank Prof. Sinan Gezici and Prof. Umut Orguner for being in the thesis committee and their valuable time. I would like to thank Arda Fazla for the valuable discussions. I would like to express my gratitude to my mother Gülçin, my father Sezai and my brother Esad for their support and encouragement.

Contents

1 Introduction **1**

2 Soft gradient boosted decision tree (sGBDT) aided sequential prediction with RNNs **6**

2.1 Related Work 7

2.2 Material and methods 8

2.2.1 Problem Statement 8

2.2.2 The proposed model 12

2.3 Experiments 20

2.3.1 Architecture Verification 21

2.3.2 Real Life Datasets 25

2.3.3 M4 Competition Datasets 28

2.3.4 Financial Datasets 32

3 A Traditional Model Meets RNNs: A Joint State Space Architecture for SARIMAX and LSTM **36**

3.1 Related Work 37

3.2 Preliminaries 39

3.2.1 Problem Statement 39

3.3 The proposed model 42

3.3.1 State Space Formulations 43

3.3.2 Estimation with Particle Filtering 46

3.4 Experiments 49

3.4.1 Real Life Datasets 50

3.4.2 M4 Competition Datasets 52

3.4.3 Genericness Study 53

4 Conclusion **55**

List of Figures

2.1 A hard binary decision tree. Pink nodes are the internal nodes and green nodes are the leaf nodes. D_i's represent the decision rules applied at each internal node to form a binary decision. For an example input $\boldsymbol{h}$, the set of decisions lead to the second leaf node where the path is colored blue. Therefore, $\boldsymbol{h}$ is associated with the value ϕ_2 that is assigned to the second leaf node. 11

2.2 A soft binary decision tree. Pink nodes are the internal nodes and green nodes are the leaf nodes, as in Fig. 2.1. Unlike the hard decision tree, however, the decision rule at each internal node is not hard and the input $\boldsymbol{h}$ is routed to both left *and* right child, as indicated with blue edges. The direction is accompanied by a Bernoulli random variable parameterized by $\boldsymbol{w}_j$ and b_j. Therefore, *all* of the leaf nodes has a contribution in the final prediction of the tree, as expressed in (2.4). We also note that the learnable parameters $\boldsymbol{\phi}_j$ in each leaf node are vector valued in general in contrast to hard decision trees. 14

2.3 The proposed LSTM-SGBDT architecture. The recurrent network in the front end is tasked to extract a feature vector, $\boldsymbol{h}_{\text{LSTM}}$, from the raw sequential data. This extracted feature vector is a summary (i.e., pooling) of the last layer's hidden states of all time steps. $\boldsymbol{h}_{\text{LSTM}}$ is then fed to the soft GBDT in the back end where the weak learners are soft decision trees. Boosted trees act as a supervised regressor; however, the need for hand-designed features are mitigated. The total loss is backpropagated through the whole architecture, whose direction is shown by the dashed lines, achieving an end-to-end optimization. 17

2.4 Learning curve comparison with a disjoint architecture. The joint architecture not only mitigates training separate models but also achieves closer to optimal performance compared to the disjoint architecture. 22

2.5 Integrity verification results of the hybrid model for different configurations. 23

2.6 Naive, LSTM and LSTM-SGBDT predictions over the forecast horizon of an M4 hourly series. 30

2.7 Time accumulated errors of standalone and hybrid models for the Alcoa stock price dataset for the period 2000-2020. 35

2.8 Time accumulated errors of standalone and hybrid models for the Hong Kong exchange rate against U.S. dollars. 35

3.1 Time accumulated errors of standalone and hybrid models for the Alcoa stock price dataset for the period 2000-2020. 53

List of Tables

2.1 Mean MAPE metrics and timings of three models for the ablation study over 255 hourly time series from the M4 dataset. 25

2.2 Cumulative errors of the models on Bank, Elevators, Kinematics and Pumadyn datasets. 27

2.3 Mean MAPE performances of the models across M4 datasets. . . 29

2.4 Mean sMAPE (%) metrics of the selected 12 models for the recursive estimation experiment over 414 hourly time series from the M4 dataset. 32

3.1 Cumulative errors of the models on Bank, Elevators, Kinematics and Pumadyn datasets 51

3.2 Mean MAPE performances of the models across M4 datasets . . . 52

Chapter 1

Introduction

We study nonlinear prediction in a temporal setting where we receive a data sequence related to a target signal and estimate the signal's next samples. This online learning problem is extensively studied in the signal processing and machine learning literatures since it has many applications such as in electricity demand [1], medical records [2], weather conditions [3] and retail sales [4]. Commonly, this problem is studied as two disjoint sub-problems where features are extracted, usually by a domain expert, and then a decision algorithm is trained over these selected features using possibly different feature selection methods. As shown in our simulations, however, this disjoint optimization can provide less than adequate results for different applications since the selected features may not be the best features to be used by the selected algorithm. To remedy this important problem, we introduce two architectures that perform these two tasks jointly and optimize both the features and the algorithm in an end-to-end manner in order to minimize the loss of interest. In particular, in both models, we use recursive structures to extract features from sequential data, while preserving the state information, i.e., the history. Two models differ in the regression of the final output, for which we first use boosted decision trees, which are shown to be very effective in several different real life applications [5,6], and secondly, a traditional auto-regressive and moving average (ARMA) model, which is specifically tailored for time series prediction. Both of these combinations are then jointly optimized

to minimize the final loss in an end-to-end manner.

The current practice for sequential data prediction revolves around three main approaches: 'traditional' models such as ARMA and ETS (exponential smoothing); neural networks such as RNNs and LSTMs; gradient-boosted decision trees such as XGBoost and LightGBM. Due to the 'no free lunch' theorem, none of these modellings are capable of giving satisfactory performance to a given arbitrary sequential data prediction problem. The well-known traditional models (such as autoregressive integrated moving average and exponential smoothing) for regression are robust to overfitting, have generally fewer parameters to estimate, easier to interpret due to the intuitive nature of the underlying model and amenable to automated procedures for hyperparameter selection [7]. However, these models have very strong assumptions about the data such as stationarity and linearity, which prevent them from capturing nonlinear patterns that real life data tend to possess [8]. Machine learning-based models, on the other hand, are data-driven and free of the strong statistical assumptions. Recurrent neural networks (RNN) and in particular long short-term memory neural networks (LSTM) are one of such models that excel at representing the data, especially the sequential data, in a hierarchical manner via nonlinear modules stacked on top of each other [9]. Thanks to the feedback connections to preserve the history in memory [10], they are widely used in sequential learning tasks. Nevertheless, a fully connected layer usually employed in the final layer of these hidden layers often hinders their regression ability (e.g., in [11], authors use an attention layer for more accurate judgments). An alternative to these deep learning models is the tree-based models that learn hierarchical relations in the data by splitting the input space and then fitting different models to each split [9]. Such a data-driven splitting and model fitting are shown to be very effective in real life applications [12, 13]. Among such models, gradient-boosted decision trees (GBDT) [14] are promising models that work via sequentially combining weak learners, i.e., decision trees. Specialized GBDTs, such as XGBoost [15] and LightGBM [16], demonstrated excellent performance at various time series prediction problems [17]. Although they were not designed to process sequential data, GBDTs can be used by incorporating temporal side information, such as lagged

values of the desired signal, as components of the feature vectors. However, this need for domain-specific feature engineering and independent design of the algorithm from the selected features hinder their full potential in sequential learning tasks since both processes are time consuming in most applications and require domain expertise [18, 19].

In this thesis, we address the alleviation of the trade-offs associated with the contemporary base models in online time series prediction. To this end, following the well-known ensembling methodology [20, 21], we introduce two 2-component hybrid models that are both fueled with joint optimization.

In Chapter 2, we combine an RNN with a GBDT where the end-to-end learning occurs via backpropagation with SGD [22]; however, since a GBDT is composed of *hard* decision trees at its core, it is not suitable for an SGD-based learning. Therefore, we introduce a *soft* decision tree model, on top of which a soft GBDT (sGBDT) model is based. This not only allows for a continuous gradient flow but also makes multi-valued, i.e., vector output time series regression possible as a byproduct, which the hard GBDT models cannot perform. This hybrid model aims to mitigate the disjoint nature of feature extraction and supervised regression by assigning each task to a separate model while enjoying end-to-end optimization. The main contributions of Chapter 2 are as follows.

1. We introduce a hybrid architecture composed of an LSTM and a soft GBDT for sequential data prediction that can be trained in an end-to-end fashion with stochastic gradient descent. We also provide the backward pass equations that can be used in backpropagation.

2. To the best of our knowledge, this model is the first in the literature that is armed with joint optimization for a sequential feature extractor in the front end and a supervised regressor in the back end.

3. The soft GBDT regressor in the back end not only learns a feature mapping but also learns the optimal partitioning of the feature space.

4. The proposed architecture is generic enough that the feature extractor part

can readily be replaced with, for example, an RNN variant (e.g., gated recurrent unit (GRU) [23]) or a temporal convolutional network (TCN) [24].

5. Through an extensive set of experiments, we show the efficacy of the proposed model over real life datasets. We also empirically verify the integrity of our model with synthetic datasets.

6. We publicly share our code for both model design and experiment reproducibility[1].

In Chapter 3, we introduce an RNN & SARIMAX (seasonal auto-regressive integrated moving average with exogenous regressors) model, which is trained with particle filtering [25] after putting them into a joint state space. Thanks to our novel state space formulation for the SARIMAX model, a single-pass solution becomes possible unlike current approaches (which depend on an iterative maximum likelihood estimation (MLE) procedure that necessitates many passes over the data), and consequently, this also opens the door for the concatenation of the two models in a single state space formulation. This hybrid architecture allows for utilizing a time-series specific model to boost the performance of a neural network in time series prediction. The main contributions of Chapter 3 are as follows.

1. We introduce a hybrid architecture composed of an LSTM network and a SARIMAX model for sequential data prediction that is trainable in a joint manner with particle filtering. To the best of our knowledge, this model is the first in the literature that is armed with joint optimization for a nonlinear sequential feature extractor and a linear traditional time series regressor.

2. We introduce a novel state space formulation for SARIMAX that allows for a single-pass solution for its parameter estimation, which eases online learning and concatenation of state spaces with other models.

[1] https://github.com/mustafaaydn/lstm-sgbdt

3. The proposed architecture is generic enough that the nonlinear part can be replaced with, for example, another RNN variant (e.g., gated recurrent unit (GRU) [23]) or a temporal convolutional network (TCN) [24]. Similarly, the linear part can also be another time series regression model (e.g., ETS [26]) as long as it has a state space formulation.

4. Through an extensive set of experiments, we show the efficacy of the proposed model over real life datasets. We also experiment with a GRU network for the nonlinear component to demonstrate the generic nature of the architecture.

5. We publicly share our code for both model design and experiment reproducibility[2].

Notation: All vectors are real column vectors and presented by boldface lowercase letters. Matrices are denoted by boldface uppercase letters. x_k and $x_{t,k}$ denotes the k^{th} element of the vectors $\boldsymbol{x}$ and $\boldsymbol{x}_t$, respectively. $\boldsymbol{x}^T$ represents the ordinary transpose of $\boldsymbol{x}$. $\boldsymbol{X}_{i,j}$ represents the entry at the i^{th} row and the j^{th} column of the matrix $\boldsymbol{X}$.

[2]https://github.com/mustafaaydn/lstm-sx

Chapter 2

Soft gradient boosted decision tree (sGBDT) aided sequential prediction with RNNs

In this chapter, we effectively combine LSTM (an enhanced RNN) and GBDT architectures for nonlinear sequential data prediction. Our motivation is to use an LSTM network as a feature extractor from the sequential data in the front end, mitigating the need for hand-designed features for the GBDT, and use the GBDT to perform supervised regression on these learned features while enjoying joint optimization. In particular, we connect the two models in an end-to-end fashion and jointly optimize the whole architecture using stochastic gradient descent in an online manner. For differentiability, and to be able to learn not only the feature mapping but also the optimal partition of the feature space, we use a *soft* gradient boosting decision tree (sGBDT) [27] that employs soft decision trees [28, 29] as the weak learners. We emphasize that our framework is generic so that one can use other deep learning architectures for feature extraction (such as RNNs and GRUs). In fact, in Section 2.3.4, we employ a GRU network in place of LSTM in our model and show the generic nature via experiments.

2.1 Related Work

Combination of decision trees with neural networks has attracted wide interest in the machine learning literature. In [30], authors propose a neural decision forest where they connect soft decision trees to the final layer of a CNN architecture. This combination provided the state-of-the-art performance in the ImageNet dataset [31]. Our work differs from this method since we use boosting with fixed-depth trees instead of bagging and we jointly optimize the whole architecture whereas they employ an alternating optimization procedure. In [29], Frosst and Hinton first employ a neural network and then using its predictions along with the true targets, train a soft decision tree. Their main goal is to get a better understanding of the decisions a neural network makes by distilling its knowledge into a decision tree. They, however, focus on classification, use two separate training phases by design, and favor explainability over model performance. The adaptive neural trees proposed in [9] uses a soft decision tree with "split data, deepen transform and keep" methodology where neural networks are the transformers in the edges of the tree. The tree is thereby grown step-wise and overall architecture necessitates both a "growth" phase and a "refinement" phase (where parameters are actually updated). Our architecture, however, does not embed neural networks inside the trees but places it in a sequential manner, does not grow trees but uses fixed depth weak learners and performs the optimization via a single backpropagation pass. References [32, 33] also take similar approaches to the aforementioned models but none of the proposed models are designed to process sequential data, which renders them unsuitable for time series forecasting tasks. Perhaps the closest architecture to ours is [34] where authors aim a forecasting task with a hybrid model of an LSTM network and XGBoost. Nevertheless, they employ a disjoint architecture in that they have a three stage training: twice for XGBoost and once for LSTM, which not only increases the computational time but also retains it from enjoying the benefits of an end-to-end optimization [35, 36]. Adaptive feature extraction-based solutions for images have been studied in [37, 38], where the authors develop adaptive and secure image steganographic algorithms based on input channel correlations and feature

extraction, respectively. In [39], authors propose a two-stream convolutional neural network (CNN) to detect image operator chains. Their end-to-end model is parallel in the sense that the input is fed to two different convolutional network architectures, outputs of which are merged together at the end to improve performance. Our framework is more flexible to perform sequential data regression as they focus on image classification and introduce a specific CNN architecture, however.

Unlike previous studies, our model is armed with joint optimization of a *sequential* feature extractor in the front end and a supervised regressor in the back end. The sequential nature of the data is addressed by the recurrent network in the front, which acts as an automatic feature extractor for the following gradient-boosted supervised regressor. Therefore, a single hybrid architecture composed of a recurrent network and gradient boosted soft trees enjoying end-to-end optimization is proposed to effectively address the negative effects of separate feature selection and decision making processes for sequential data regression.

2.2 Material and methods

2.2.1 Problem Statement

We study the nonlinear prediction of sequential data. We observe a sequence $\{y_t\}$ possibly along with a side information sequence $\{\boldsymbol{s}_t\}$. At each time t, given the past information $\{y_k\}$, $\{\boldsymbol{s}_k\}$ for $k \leq t$, we produce the predictions $\hat{\boldsymbol{d}}_H = [\,\hat{y}_{t+1}, \ldots, \hat{y}_{t+H}]^T$ where H is the number of steps ahead to predict. Hence, in this purely online setting, our goal is to find the relationship

$$\hat{\boldsymbol{d}}_H = f\big([\ldots, y_{t-1}, y_t], [\ldots, \boldsymbol{s}_{t-1}, \boldsymbol{s}_t]\big),$$

where f is a nonlinear function, which models $\boldsymbol{d}_H = [\,y_{t+1}, \ldots, y_{t+H}]^T$. We note that f can either produce H estimates at once or do it in a recursive fashion (roll out) by using its own predictions. In either case, we suffer the loss over the

prediction horizon given by

$$L = \frac{1}{H} \sum_{i=1}^{H} l(d_{H,i}, \hat{d}_{H,i}),$$

where l, for example, can be the squared error loss. As an example, in weather nowcasting, temperature (y_t) for the next six hours ($H = 6$) are predicted using the hourly measurements of the past. Side information could include the wind and humidity levels ($\boldsymbol{s}_t \in \mathbb{R}^2$). The model updates itself whenever new measurements of the next hour become available, i.e., works in an online manner.

We use RNNs for processing the sequential data. An RNN is described by the recursive equation [10]

$$\boldsymbol{h}_t = u(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}),$$

where $\boldsymbol{x}_t$ is the input vector and $\boldsymbol{h}_t$ is the state vector at time instant t. $\boldsymbol{W}_{ih}$ and $\boldsymbol{W}_{hh}$ are the input-to-hidden and hidden-to-hidden weight matrices (augmented to include biases), respectively; u is a nonlinear function that applies element wise. As an example, for the prediction task, one can use $\boldsymbol{s}_t$ directly as the input vector, i.e., $\boldsymbol{x}_t = \boldsymbol{s}_t$, or some combination of the past information $\{y_t, \ldots, y_{t-r+1}\}$ with a window size r along with $\boldsymbol{s}_t$ to construct the input vector $\boldsymbol{x}_t$. The final output of the RNN is produced by passing $\boldsymbol{h}_t$ through a sigmoid or a linear model:

$$\hat{\boldsymbol{d}}_H = \boldsymbol{W}\boldsymbol{h}_t,$$

where $\boldsymbol{W}$ is $H \times m$ and $\boldsymbol{h}_t \in \mathbb{R}^m$. Here, we consider $\boldsymbol{h}_t$ as the "extracted feature vector" from the sequential data by the RNN.

To effectively deal with the temporal dependency in sequential data, we employ a specific kind of RNN, namely the LSTM neural networks [40] in the front end as a feature extractor. Here we use the variant with the forget gates [41]. The

forward pass equations in one cell are described as

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_f) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned} \tag{2.1}$$

where $\boldsymbol{c}_t$ is the state vector, $\boldsymbol{x}_t$ is the input to the cell, $\boldsymbol{h}_{t-1}$ is the hidden state from the previous cell and $\{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\}$ denotes the vertical concatenation of $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. $\boldsymbol{f}_t, \boldsymbol{i}_t, \tilde{\boldsymbol{c}}_t, \boldsymbol{o}_t$ are the forget gate, input gate, block input and output gate vectors, respectively, whose stacked weight matrices are $\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_c$ and $\boldsymbol{W}_o$. The corresponding biases are represented by $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c$ and $\boldsymbol{b}_o$. σ denotes the logistic sigmoid function, i.e, $\sigma(x) = 1/(1 + \exp(-x))$ and tanh is the hyperbolic tangent function, i.e, $\tanh(x) = 2\sigma(2x) - 1$; both are nonlinear activation functions that apply point-wise. $\odot$ is the Hadamard (element-wise) product operator. The constant error flow through the cell state in backpropagation allows LSTMs to prevent vanishing gradients, and nonlinear gated interactions described by (2.1) control the information flow effectively to capture long term dependencies [22].

Instead of the linear model, one can use decision trees to process the extracted features, i.e., $\boldsymbol{h}_t$ produced by the RNN or LSTM. A (hard) binary decision tree forms an axis-aligned partition of the input space [42]. The tree consists of two types of nodes: internal nodes and leaf nodes, as depicted in Fig. 2.1. The input vector enters from the root node and is subjected to a decision rule at each internal node, routing it either to the left child or the right child. This path ends when the input ends up in a leaf node where an output value is produced, through which the input space is partitioned. Thereby, a regression tree with L leaf nodes is recursively constructed to split the input space into L regions, R_ℓ, where $\ell = 1, 2, \ldots, L$. This construction is usually done with a greedy approach [42]. The forward pass of an input vector $\boldsymbol{h}$ is given by

$$g(\boldsymbol{h}) = \sum_{\ell=1}^{L} \phi_\ell \mathbb{1}(\boldsymbol{h} \in R_\ell), \tag{2.2}$$

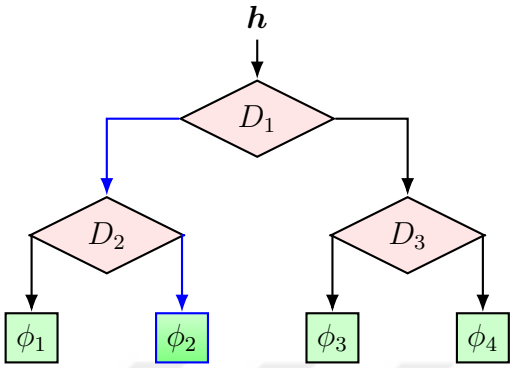

Figure 2.1: A hard binary decision tree. Pink nodes are the internal nodes and green nodes are the leaf nodes. D_i's represent the decision rules applied at each internal node to form a binary decision. For an example input $\boldsymbol{h}$, the set of decisions lead to the second leaf node where the path is colored blue. Therefore, $\boldsymbol{h}$ is associated with the value ϕ_2 that is assigned to the second leaf node.

where ϕ_ℓ is the scalar quantity that ℓ^{th} leaf node associates with the input, $\mathbb{1}$ is the indicator function and g represents the tree. By (2.2), the summation has only one nonzero term due to the *hard* decisions that route the input vector in a binary fashion, i.e., only one of the leaf nodes contributes to the final output.

A standalone tree, however, usually suffers from overfitting [42] and a way to overcome this is to use many trees in a sequential manner, i.e., gradient boosting decision trees (GBDT) [14]. A GBDT works by employing individual trees as "weak" learners and fitting one after another to the residuals of the previous tree with the same greedy approach for constructing individual trees. The forward pass is the weighted sum of the predictions of each individual tree. For $M+1$ trees, the overall output for the input $\boldsymbol{h}$ is given by

$$\text{GBM}(\boldsymbol{h}) = g^{(0)}(\boldsymbol{h}) + \sum_{j=1}^{M} \nu g^{(j)}(\boldsymbol{h}), \tag{2.3}$$

where $g^{(j)}$ for $j = 0, \ldots, M$ represents the forward pass of the j^{th} tree and ν is the shrinkage parameter that weights the predictions of individual trees (except for the first one in the chain) to provide regularization [14]. The very first tree, i.e., $g^{(0)}$, predicts a constant value regardless of the input, e.g., the mean value of the targets when the loss criterion is the mean squared error.

Connecting a recurrent neural network and a decision tree based architecture in an end-to-end fashion and training with gradient based methods is not possible when conventional decision trees are used as they inherently lack differentiability due to hard decisions at each node. Therefore, for the joint optimization, we introduce a model composed of an RNN in the front end and a *soft* GBDT in the back end, which are jointly optimized in an end-to-end manner. We next introduce this architecture for nonlinear prediction.

2.2.2 The proposed model

We next introduce a model which jointly optimizes the feature selection and model building in a fully online manner. To this end, we use an LSTM network in the front end as a feature extractor from the sequential data and a soft GBDT (sGBDT) in the back end as a supervised regressor. Given $\{y_k\}$ and $\{\boldsymbol{s}_k\}$ for $k \leq t$ in the forward pass, a multi layer LSTM produces hidden state vectors by applying (2.1) in each of its cells. Any differentiable pooling method can be used on the hidden state vectors of the last layer, e.g, last pooling where only the hidden state of the rightmost LSTM cell (when unrolled) is taken. This pooled quantity, $\boldsymbol{h}_{\text{LSTM}}$, is the filtered sequential raw data that represents the feature extraction or learning part via LSTM. This extracted feature vector $\boldsymbol{h}_{\text{LSTM}}$ is then fed into the sGBDT as the input to produce $\hat{\boldsymbol{d}}_H$. We now describe the soft decision tree based GBDT used in the model.

2.2.2.1 sGBDT

An sGBDT is a gradient boosting decision tree [14] where weak learners are soft decision trees [28, 29]. In this section, we first study the soft decision tree variant we use in our model and then present the gradient boosting machinery that combines them.

2.2.2.1.1 Soft Decision Tree (sDT) The widely used *hard* binary decision trees recursively route a sample from the root node to a single leaf node and result in (usually) axis-aligned partitions of the input space. The decision rules applied at each node redirect the sample either to the left or the right child. Soft decision trees differ in that they redirect a sample to *all* leaf nodes of the tree with certain probabilities attached. These *soft* decision rules yield a differentiable architecture.

Formally, we have a binary tree with an ordered set of internal nodes $\mathcal{N}$ and leaf nodes $\mathcal{L}$. At the m^{th} internal node $N_m \in \mathcal{N}$, the sample is subjected to a probabilistic routing controlled by a Bernoulli random variable with parameter p_m where "success" corresponds to routing to the left child. A convenient way to obtain a probability is to use the sigmoid function $\sigma(z) = 1/(1+\exp(-z))$. Thereby we attach $\boldsymbol{w}_m$ and b_m to each N_m as learnable parameters such that $p_m = \sigma(\boldsymbol{w}_m^T \boldsymbol{h} + b_m)$ is the probability of sample $\boldsymbol{h}$ going to the left child and $1 - p_m$ is the probability of it going to the right child. With this scheme, we define a "path probability" for each leaf node $\ell \in \mathcal{L}$ as the multiplication of the probabilities of the internal nodes that led to it. As shown in Fig. 2.2, *every* leaf node has a probability attached to it for a given sample. These path probabilities correspond to each leaf nodes' contribution in the overall prediction given by the tree, which is

$$\begin{aligned} g(\boldsymbol{h}) &= \sum_{\ell \in \mathcal{L}} \text{pp}_\ell \, \boldsymbol{\phi}_\ell \\ &= \sum_{\ell \in \mathcal{L}} \big(\prod_{i \in \text{A}(\ell)} p_i \big) \, \boldsymbol{\phi}_\ell \\ &= \sum_{\ell \in \mathcal{L}} \big(\prod_{i \in \text{A}(\ell)} \sigma(\boldsymbol{w}_i^T \boldsymbol{h} + b_i) \big) \, \boldsymbol{\phi}_\ell, \end{aligned} \tag{2.4}$$

where pp_ℓ is the path probability of leaf node ℓ, $\boldsymbol{\phi}_\ell$ is the predicted value produced at ℓ which is a learnable parameter (and vector valued in general) and $\text{A}(\ell)$ denotes the ascendant nodes of the leaf node ℓ, including the root node.

The soft decisions formed through p_m's not only allow for a fully differentiable tree but also result in smooth decision boundaries that effectively reduce the number of nodes necessary to construct the tree in comparison to hard trees [28].

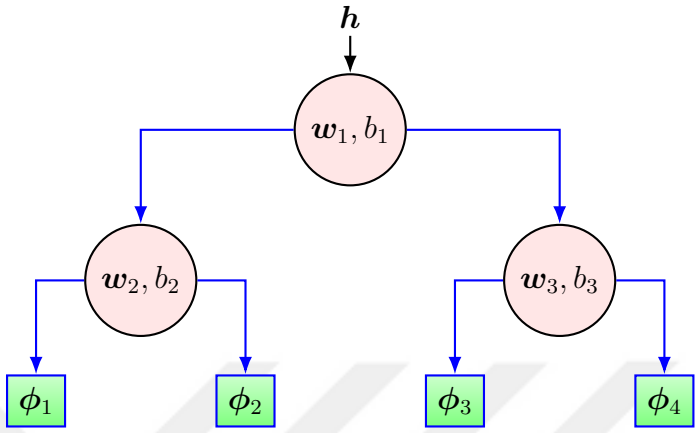

Figure 2.2: A soft binary decision tree. Pink nodes are the internal nodes and green nodes are the leaf nodes, as in Fig. 2.1. Unlike the hard decision tree, however, the decision rule at each internal node is not hard and the input $\boldsymbol{h}$ is routed to both left *and* right child, as indicated with blue edges. The direction is accompanied by a Bernoulli random variable parameterized by $\boldsymbol{w}_j$ and b_j. Therefore, *all* of the leaf nodes has a contribution in the final prediction of the tree, as expressed in (2.4). We also note that the learnable parameters $\boldsymbol{\phi}_j$ in each leaf node are vector valued in general in contrast to hard decision trees.

Furthermore, $\boldsymbol{\phi}_{\boldsymbol{\ell}}$ being a vector valued quantity in general makes multi output regression possible.

2.2.2.1.2 Gradient boosting of sDTs We adapt the gradient boosting machinery as originally proposed in [14]. The weak learners are fixed depth soft decision trees. Hence, all of the trees have a predetermined depth which saves computational time. In this regard, our sGBDT resembles AdaBoost where decision stumps are employed as weak learners [43]. Note that our implementation can be extended to growing trees in a straightforward manner. As depicted in Fig. 2.3, each tree (except for the very first one) is fitted to the residuals of the previous tree in the boosting chain. The first tree, g_0, does not have any predecessor and is tasked to produce a constant output regardless of the input, which can be, for example, the mean of the target values of the training set. The forward pass equation of the sGBDT follows (2.3). The first tree does not possess any learnable parameter and rather acts as a starting point in the boosting chain. The parameters of other trees, which include $\boldsymbol{w}_m, b_m$ for internal nodes and $\boldsymbol{\phi}_\ell$

for leaf nodes, are learned through backpropagation with gradient descent. The backward pass equations are given in the next section.

An sGBDT has three hyperparameters to tune: the number of trees, the depth of each tree and the shrinkage rate, which controls how much an individual tree contributes to the forwarded output. The number of trees and the shrinkage rate ideally move in an inversely proportional way, i.e., as the number of trees increases, one needs to reduce the shrinkage rate. The shrinkage rate not only acts as a learning rate parameter but also provides (inverse) regularization [14]. Therefore smaller values of shrinkage rate reduce overfitting; however, this leads to usage of large number of trees, increasing the computational cost. As the depth of each weak learner increases, not only the computational complexity increases exponentially but also the risk of overfitting arises. On the other hand, stumps might easily underfit a given dataset. To remedy these trade-offs, we employ validation procedures to choose these hyperparameters for a given task. As an example, we perform a k-fold cross validation over the training split of the data and determine the "best" configuration of hyperparameters.

2.2.2.2 The end-to-end model

Here, we describe how a given sequential data is fed forward and the corresponding backward pass equations of the joint optimization. Given an L-layer LSTM network, the input is first windowed and then fed into the first layer of LSTM, denoted by $\text{LSTM}^{(1)}$. The window size T determines how many time steps the model should look back for the prediction, i.e., the number of steps an LSTM cell should be unrolled in each layer. Each cell applies (2.1) sequentially where $\boldsymbol{h}_0 = \boldsymbol{c}_0 = \mathbf{0}$, i.e., initial cell gets $\mathbf{0}$ vector for both the hidden state and cell state. Hidden states at all time steps ($\boldsymbol{h}_t^{(j)}$ for the j^{th} layer) are recorded and fed into the next layer of LSTM as the inputs as shown in the left side of Fig. 2.3. This process repeats until it reaches the last layer of LSTM whereby a pooling method, P, is applied to the hidden states of each cell $\boldsymbol{h}_t^{(L)}$ for $t = 1, \ldots, T$. P can be any differentiable operation; here we present three options of *last*, *mean*

and *max* pooling as

$$\begin{aligned} P_{last}(\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_T^{(L)}) &= \boldsymbol{h}_T^{(L)} \\ P_{mean}(\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_T^{(L)}) &= \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{h}_t^{(L)} \\ P_{max}(\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_T^{(L)})_i &= \max_t h_{t,i}^{(L)}. \end{aligned} \tag{2.5}$$

The pooled LSTM output, $\boldsymbol{h}_{\text{LSTM}} = P(\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_T^{(L)})$, carries the features extracted from the raw sequential data and becomes the input vector for the sGBDT, as shown in Fig. 2.3. Then, as per the gradient boosting machinery, $\boldsymbol{h}_{\text{LSTM}}$ is fed into $M+1$ soft trees, last M of which give an output as described in (2.4), whereas the first one outputs a constant value irrespective of the input, as explained in Section 2.2.2.1.2. Nevertheless, we still call this first component in the chain a *tree*. The output of the sGBDT is a weighted sum of individual tree outputs. Combining (2.3) and (2.4), the overall output of the sGBDT is given by

$$\begin{aligned} \text{sGBDT}(\boldsymbol{h}) &= \sum_{j=0}^{M} \nu^{(j)} g^{(j)}(\boldsymbol{h}) \\ &= g^{(0)}(\boldsymbol{h}) + \nu \sum_{j=1}^{M} \sum_{\ell \in \mathcal{L}^{(j)}} \prod_{i \in \text{A}(\ell^{(j)})} p_i^{(j)} \boldsymbol{\phi}_\ell^{(j)}, \end{aligned} \tag{2.6}$$

where

$$\nu^{(j)} = \begin{cases} 1, & \text{if } j = 0 \\ \nu, & \text{otherwise,} \end{cases}$$

$g^{(j)}$ represents the forward pass of the j^{th} tree, $\mathcal{L}^{(j)}$ is the set of leaf nodes in the j^{th} tree and $\text{A}(\ell^{(j)})$ is the set of ascendant nodes of leaf node ℓ of the j^{th} tree. We note that the superscript (j) indicates that the quantity belongs to the j^{th} tree. This completes the forward pass of the data.

We fit this architecture in an end-to-end manner using backpropagation with gradient descent. For the backward pass equations, without loss of generality, we focus on the case where values produced at leaf nodes of the trees, $\phi_\ell^{(j)}$, are scalar quantities; and we use the mean squared error for the loss function. Further, for brevity, we assume a 1-layer LSTM and let $\tilde{\boldsymbol{h}}_{\text{LSTM}}$ be the augmented version

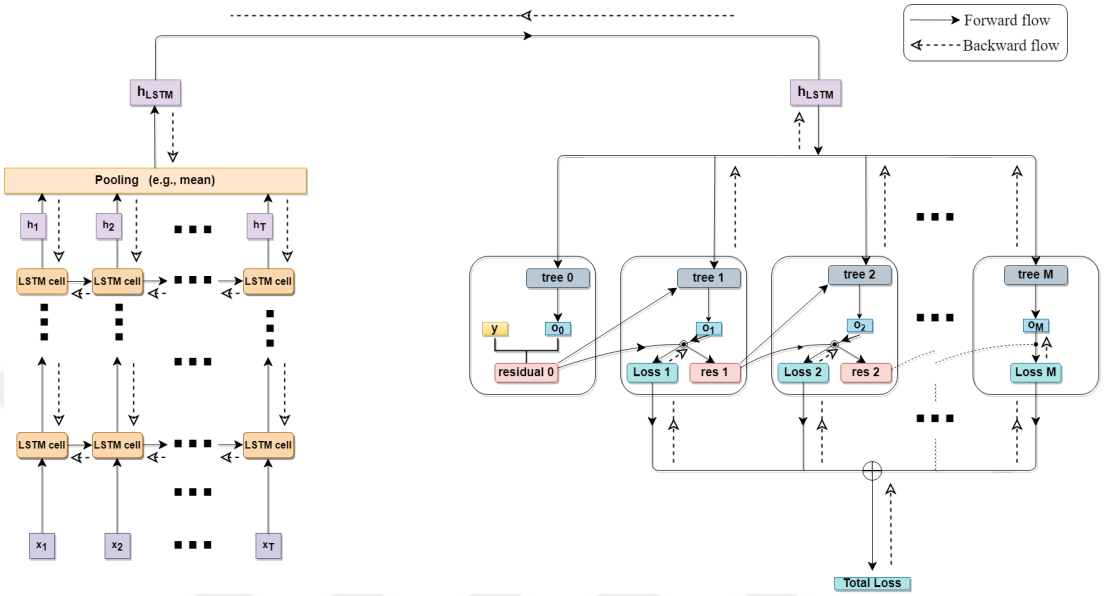

Figure 2.3: The proposed LSTM-SGBDT architecture. The recurrent network in the front end is tasked to extract a feature vector, $\boldsymbol{h}_{\text{LSTM}}$, from the raw sequential data. This extracted feature vector is a summary (i.e., pooling) of the last layer's hidden states of all time steps. $\boldsymbol{h}_{\text{LSTM}}$ is then fed to the soft GBDT in the back end where the weak learners are soft decision trees. Boosted trees act as a supervised regressor; however, the need for hand-designed features are mitigated. The total loss is backpropagated through the whole architecture, whose direction is shown by the dashed lines, achieving an end-to-end optimization.

of the pooled LSTM output $\boldsymbol{h}_{\text{LSTM}}$ (i.e., it has a 1 prepended to it so that $\boldsymbol{w}_m$ and b_m described in Section 2.2.2.1.1 are collapsed into one vector, $\boldsymbol{w}_m$). Let $\boldsymbol{x}_{t'} \in \mathbb{R}^{m+1}$ with $t' = 1, \ldots, T$ be the input vector (possibly augmented with side information $\boldsymbol{s}_{t'} \in \mathbb{R}^m$; e.g., $\boldsymbol{x}_{t'} = [y_{t+1-t'}, \boldsymbol{s}_{t'}^T]^T$) fed into t'^{th} LSTM cell when unrolled and $y_{true} \coloneqq y_{t+1}$ be the ground truth, which is a real number. The total loss E obtained from the soft trees, as shown in Fig. 2.3, can be written as

$$E = \text{Total Loss} = \sum_{j=1}^{M} \text{Loss}^{(j)} = \sum_{j=1}^{M} (r^{(j)} - \nu o^{(j)})^2,$$

where $o^{(j)}$ is the output of the j^{th} tree as in (2.4), ν is the shrinkage rate parameter that helps for regularization [14] and $r^{(j)}$ is the residual that becomes the

supervised signal the j^{th} tree is fitted to, i.e.,

$$o^{(j)} = \sum_{\ell^{(j)}} \mathrm{pp}_{\ell}^{(j)} \phi_{\ell}^{(j)}$$

$$r^{(j)} = y_{true} - \nu \sum_{i=0}^{j-1} o^{(i)}.$$

We now present the gradient of the total loss with respect to the learnable parameters. For the sGBDT, we have $\phi_{\ell}^{(j)}$ for each leaf node $\ell^{(j)}$ and $\boldsymbol{w}_{\boldsymbol{m}}^{(j)}$ for each internal node $m^{(j)}$ in the j^{th} tree. The gradients are given as

$$\frac{\partial E}{\partial \phi_{\ell}^{(j)}} = \mathrm{pp}_{\ell}^{(j)} \sum_{k=j}^{M} 2(\nu o^{(k)} - r^{(k)}) \tag{2.7}$$

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{w}_m^{(j)}} = \Bigg(\sum_{\ell^{(j)} \in \mathrm{D}(\mathrm{m}^{(j)})} \phi_{\ell}^{(j)} \frac{\mathrm{pp}_{\ell}^{(j)} \mathrm{p}_m^{(j)} (1 - \mathrm{p}_m^{(j)})}{\mathrm{p}_m^{(j)} - \mathbb{1}(\ell^{(j)} \in \mathrm{LD}(m^{(j)}))} \\ \tilde{\boldsymbol{h}}_{\mathrm{LSTM}} \sum_{k=j}^{M} 2(\nu o^{(k)} - r^{(k)}) \Bigg), \end{aligned} \tag{2.8}$$

where M is the number of weak learners (soft decision trees), $\mathrm{D}(m)$ is the set of descendants of the internal node m, $\mathrm{LD}(m)$ is the set of *left* descendants of the internal node m, i.e., it consists of the nodes in the sub-tree rooted at node m which are reached from m by first selecting the left branch of m. (2.7) and (2.8) give the necessary components for the update equations of the learnable parameters of the sGBDT, namely, $\boldsymbol{w}_m^{(j)}$ and $\phi_{\ell}^{(j)}$ for each tree and node.

For the joint optimization, we also derive the gradient of the loss with respect to $\tilde{\boldsymbol{h}}_{\mathrm{LSTM}}$, which is given as

$$\begin{aligned} \frac{\partial E}{\partial \tilde{\boldsymbol{h}}_{\mathrm{LSTM}}} = \\ \sum_{j=1}^{M} \Bigg(\sum_{\ell^{(j)}} \phi_{\ell}^{(j)} \mathrm{pp}_{\ell}^{(j)} \\ \Bigg(\sum_{n^{(j)} \in \mathrm{A}(\ell^{(j)})} (\mathbb{1}(n^{(j)} \in \mathrm{RA}(\ell^{(j)})) - p_n^{(j)}) \mathbf{w}_n^{(j)} \Bigg) \\ \sum_{k=j}^{M} 2(\nu o^{(k)} - r^{(k)}) \Bigg), \end{aligned} \tag{2.9}$$

where $\mathrm{A}(\ell)$ is the set of ascendants of the leaf node ℓ, $\mathrm{RA}(\ell)$ is the set of *right* ascendants of the leaf node ℓ, i.e., it consists of the nodes in the tree from which by first selecting the right branch, there is a path that leads to ℓ. With (2.9), the backpropagation path until the pooled output of LSTM is completed.

We assume last pooling, i.e., $P = P_{last}$ of (2.5) for the derivation such that the error propagates through the hidden state of the last cell. The gradients of the total loss with respect to LSTM parameters are given by

$$\frac{\partial E}{\partial \boldsymbol{W}_\star} = \sum_{t=1}^{T} \left(\frac{\partial E}{\partial \boldsymbol{h}_t} \odot \delta\bigstar_t \right) \tilde{\boldsymbol{x}}_t^T \tag{2.10}$$

$$\frac{\partial E}{\partial \boldsymbol{b}_\star} = \sum_{t=1}^{T} \frac{\partial E}{\partial \boldsymbol{h}_t} \odot \delta\bigstar_t, \tag{2.11}$$

where $\star$ denotes the set $\{f, i, c, o\}$,

$$\frac{\partial E}{\partial \boldsymbol{h}_{t-1}^T} = \frac{\partial E}{\partial \boldsymbol{h}_t} (\Lambda(\delta \boldsymbol{o}_t)\tilde{\boldsymbol{W}}_o + \Lambda(\delta \boldsymbol{f}_t)\tilde{\boldsymbol{W}}_f + \Lambda(\delta \boldsymbol{i}_t)\tilde{\boldsymbol{W}}_i + \Lambda(\delta \boldsymbol{c}_t)\tilde{\boldsymbol{W}}_c)^T,$$

starting from the hidden state of the rightmost cell (i.e, $t = T$) with $\partial E/\partial \boldsymbol{h}_T = \partial E/\partial \tilde{\boldsymbol{h}}_{\mathrm{LSTM}}$ given by (2.9) and $\tilde{\boldsymbol{W}}_\star$ denotes the square matrix inside the stacked $\boldsymbol{W}_\star$ that multiplies $\boldsymbol{h}_{t-1}$ in (2.1). $\Lambda(\cdot)$ results in a square, diagonal matrix where the diagonal entries are given by its vector operand. Finally, $\delta\bigstar_t$ are given by

$$\begin{aligned}
\delta \boldsymbol{f}_t &= \boldsymbol{o}_t \odot \tanh'(\boldsymbol{c}_t) \odot \boldsymbol{c}_{t-1} \odot \sigma'(\boldsymbol{f}_t) \\
\delta \boldsymbol{i}_t &= \boldsymbol{o}_t \odot \tanh'(\boldsymbol{c}_t) \odot \tilde{\boldsymbol{c}}_t \odot \sigma'(\boldsymbol{i}_t) \\
\delta \boldsymbol{c}_t &= \boldsymbol{o}_t \odot \tanh'(\boldsymbol{c}_t) \odot \boldsymbol{i}_t \odot (1 - \tilde{\boldsymbol{c}}_t^2) \\
\delta \boldsymbol{o}_t &= \tanh(\boldsymbol{c}_t) \odot \sigma'(\boldsymbol{o}_t).
\end{aligned}$$

Equations (2.7)–(2.11) give the required backward pass equations of the whole architecture. The corresponding stochastic gradient update equations are

$$\Delta \phi_\ell^{(j)} = -\eta \frac{\partial E}{\partial \phi_\ell^{(j)}} \qquad \Delta \boldsymbol{w}_m^{(j)} = -\eta \frac{\partial E}{\partial \boldsymbol{w}_m^{(j)}} \tag{2.12}$$

$$\Delta \boldsymbol{W}_\star = -\eta \frac{\partial E}{\partial \boldsymbol{W}_\star} \qquad \Delta \boldsymbol{b}_\star = -\eta \frac{\partial E}{\partial \boldsymbol{b}_\star}, \tag{2.13}$$

where Δ represents the change in the variable between two successive iterations and η is the learning rate hyperparameter.

Remark. Given a conventional LSTM network described by (2.1), for an input vector of dimension m and hidden state dimension q, there are four matrix-vector multiplications for the input, four matrix-vector multiplications for the recurrent state, and three vector-vector multiplications between the gates; these correspond to $4q^2 + 4qm + 3q$ multiplication operations in total. On the other hand, a soft tree of depth d, which is fed with the resultant state vector of the LSTM in the front end (of dimension q) has 2^d- internal nodes and therefore results in $q(2^d - 1)$ multiplications for the probability calculations and $2^d - 1$ multiplications for the path probabilities, leading to $(q+1)(2^d-1)$ multiplications in total per tree. Hence, the total number of multiplications an sGBDT with M trees brings is $M(2^d-1)(q+1)$. Although the depth seems to increase complexity exponentially, since it is constant per tree and does not exceed 3 in applications, it can be treated as a constant along with M. Overall, the multiplicational complexity of the hybrid model remains in the order of q^2 in terms of the hidden state dimension of the LSTM network, which is the same as using an LSTM alone.

2.3 Experiments

In this section, we illustrate the performance of the proposed architecture under different scenarios. In the first part, we demonstrate the learning advantages of the proposed model with respect to disjoint structures through synthetic datasets. We also perform an ablation study to exhibit the importances of each component of the model. We then consider the regression performance of the model over well-known real life datasets such as bank [44], elevators [45], kinematics [46] and pumadyn [45] datasets in an online setting. We then perform simulations over the hourly, daily and yearly subsets of the M4 competition [47] dataset, aiming one-step ahead forecasting. We lastly consider two financial datasets, Alcoa stock price [48] and Hong Kong exchange rate data against USD [49] where we replace the LSTM part of our model with a GRU network to show the generic nature of our framework. In each experiment, for all models, the only preprocessing step we applied to the datasets is standardization. In offline settings (e.g., with the

M4 dataset), we first extract the mean and the standard deviation of the features across the training split of the data, and scale both the training and testing data with those same statistics, i.e., for each feature, we subtract the mean and divide by the standard deviation. In online settings (e.g., in financial datasets), we apply standardization cumulatively, i.e., we store a running mean and running standard deviation obtained from the available data for scaling and update them as new data points arise. We use standardization to prevent possible scaling differences among the raw features, to which the gradient descent is highly sensitive and it could lead to performance degradation in learning [50]. We emphasize that we do not extract any additional features from the datasets; the LSTM network in the front end is tasked to automatically extract the features from the raw data. We use the past values of the target signal to feed into the LSTM network, and use the side information features whenever dataset provides them, e.g., in the real life datasets of Section 3.2. All the experiments were conducted using a computer with i7-6700HQ processor, 2.6 GHz CPU and 12 GB RAM.

2.3.1 Architecture Verification

In this section, we show the advantages of our end-to-end optimization framework with respect to the classical disjoint framework, i.e., where one first extracts features and then optimizes the regressor. For controlled experiments, we first work with artificially generated data that are generated through an LSTM network. The inputs are from i.i.d. standard Gaussian distribution and the targets, y_t, are formed as the output of a fixed LSTM network with a given seed. The "disjoint" model consists of an LSTM for feature extraction and a hard GBDT for regressing over those features. We note that the mechanism, as stated in [34], requires fitting the models independently. On the other hand, our "joint" model has an LSTM in the front end and a soft GBDT in the back end, while the optimization is now end-to-end. We generate 1,000 artificial samples with a fixed seed and spare 20% of the data to test set. Both models are trained for 100 epochs. At the end of each epoch, we subject both models to test data and record their prediction performance in RMSE. The results are seen in Fig. 2.4. We observe that

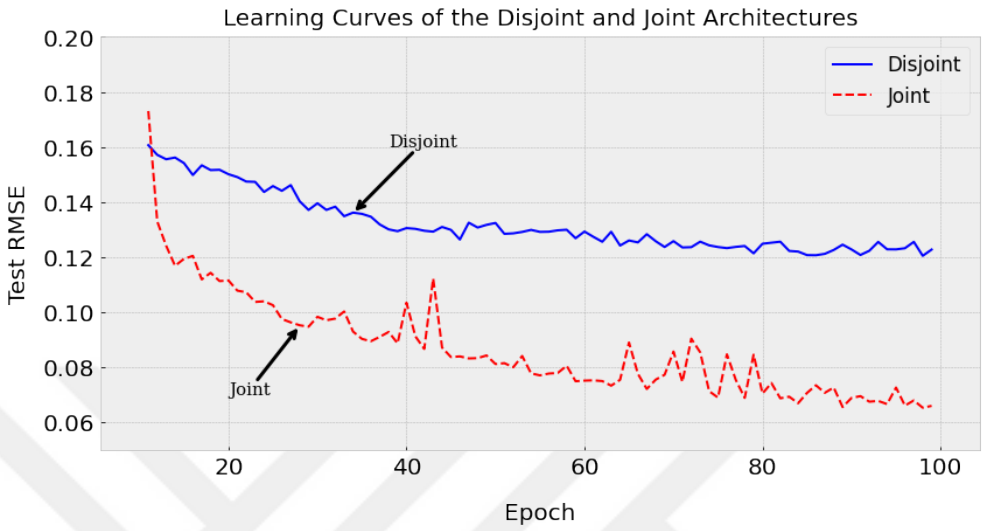

Figure 2.4: Learning curve comparison with a disjoint architecture. The joint architecture not only mitigates training separate models but also achieves closer to optimal performance compared to the disjoint architecture.

the disjoint model struggles to reach an optimal solution and stabilizes its performance after around 40 epochs while the joint architecture has steadily-decreasing error performance on the test set. A number of peaks in the curve of the joint model is possibly caused by the noninformative samples in those timestamps, for which the recovery is made shortly unlike the disjoint model having a plateau. This shows the advantage of employing end-to-end optimization and verifies the working characteristics of our algorithm.

We further test the integrity of our end-to-end model using datasets that are produced using both LSTM and GBDT. Through this, we check the learning mechanism of the model in various controlled scenarios. First, we generate a random sequential input from an i.i.d. standard Gaussian distribution and feed it o an LSTM network whose weights are fixed with a random seed. Its output is then fed to a fixed soft GBDT to get the ground truths, i.e, y_t. The architecture is then subjected to *replicate* these ground truth values as close as possible given the same random inputs. This tests whether the model is able to tune its parameters in an end-to-end manner to fit the given synthetic dataset. Secondly, we feed the above randomly generated data directly into a soft GBDT and get the ground truths as its output. We then train the model with this input-output

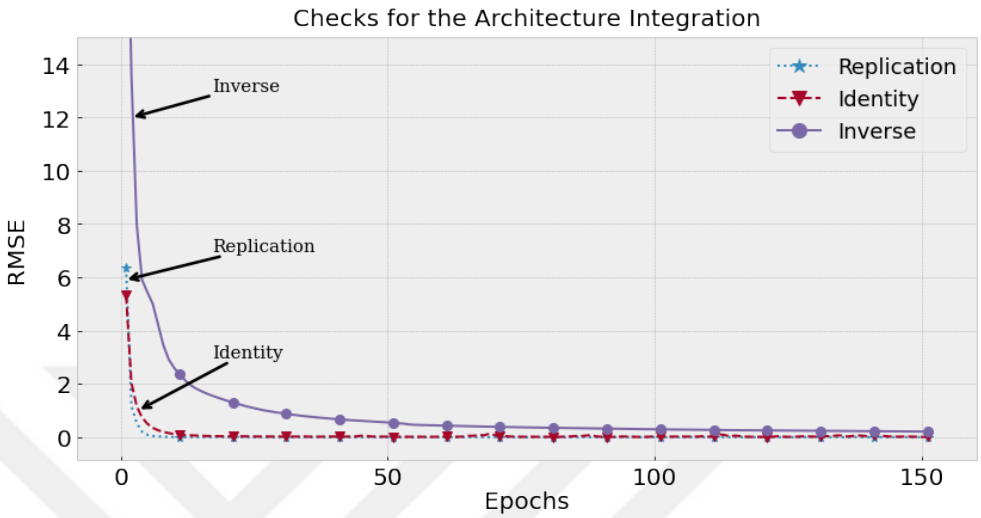

Figure 2.5: Integrity verification results of the hybrid model for different configurations.

pair, expecting that the LSTM part in the front end learns the *identity* mapping. Lastly, we again use a soft GBDT to generate the same input-output pair, but train the overall model with the inputs $\boldsymbol{AX}$ where $\boldsymbol{X}$ is the randomly generated design matrix and $\boldsymbol{A}$ is a random constant matrix of suitable shape. This configuration aims for LSTM to learn an *inverse* mapping. We generate 1,000 samples in all three configurations.

In Fig. 2.5, we observe that in all configurations, the network reaches to near zero root mean squared error (RMSE) between its predictions and the ground truths. Convergence is very fast in *replication* and *identity* tasks suggesting that the model is capable of learning from the individual parts and that the joint optimization readily tunes the parameters of the LSTM network. In the *inverse* task, the convergence is rather slow since the input was subjected to a random projection. Still, the LSTM in the front end learns an inverse mapping, verifying the integrity of the end-to-end model.

2.3.1.1 Ablation Study

In order to show the importance of both feature extractor and supervised regressor parts of the proposed model, we perform an ablation study. To this end, we train three model configurations. First one is the fully functional LSTM-SGBDT model, where the feed-forward and backward update operations are as described in Section 2.2.2. Second model is an LSTM-SGBDT where the feed-forward mechanism remains intact but the feature extractor LSTM part is "frozen", i.e., the corresponding learned parameters are *not* updated through backpropagation. The parameters of the SGBDT part, however, are still updated through backpropagation. Similarly, the third model freezes its supervised regressor SGBDT part. We use 255 randomly chosen hourly time series from the M4 competition dataset [47] and perform one-step ahead forecasting with all three models. We report the mean absolute percentage error (MAPE) over the 48 hours of forecast horizon as it is scale-independent and defined as

$$\text{MAPE}(\boldsymbol{d}_H, \hat{\boldsymbol{d}}_H) = \frac{1}{H} \sum_{i=1}^{H} \left| \frac{d_{H,i} - \hat{d}_{H,i}}{d_{H,i}} \right|,$$

where H is the prediction horizon (e.g., 48 hours), $\boldsymbol{d}_H$ and $\hat{\boldsymbol{d}}_H$ denote the vector of true and predicted values over the horizon, respectively. For a fair comparison, we use the same number of epochs in each model's training phase. Other hyperparameters are validated with 3-fold cross validation over the training split of each series. We also report the timing of each model; we note that the duration of validation in timings is not included, i.e., only the elapsed times over training and prediction of the "best" model is reported for each three models. In Table 2.1, we observe that the full model is more than three times better than either of the frozen models in mean MAPE performance over 255 time series, which shows that both the feature extractor LSTM part and the supervised regressor part SGBDT are crucial parts of the proposed model. In addition, the timing of the fully functional model is, as expected, more than the frozen models; however, the difference is tolerable as it is around 3 to 6 seconds over 255 series, making it a favorable choice.

	Mean MAPE	Mean Time (s)
Model I (full)	**0.1837**	15.42
Model II (frozen LSTM)	0.6120	**9.83**
Model III (frozen SGBDT)	0.5696	12.33

Table 2.1: Mean MAPE metrics and timings of three models for the ablation study over 255 hourly time series from the M4 dataset.

2.3.2 Real Life Datasets

In this section, we evaluate the performance of the proposed model in an online learning setting. To this end, we consider four real life datasets.

- Bank dataset [44] is a simulation of queues in a series of banks using customers with varying level of complexity of tasks and patience. We aim to predict the fraction of customers that have left the queue without getting their task done. There are 8,192 samples and each sample has 32 features, i.e., $\boldsymbol{x}_t \in \mathbb{R}^{32}$.

- Elevators dataset [45] is obtained from the experiments of controlling an F16 aircraft; the aim is to predict a numeric variable in range $[0.012, 0.078]$ related to an action on the elevator of the plane, given 18 features. The dataset has 16,599 samples.

- Kinematics dataset [46] consists of 8,192 samples that are a result of a realistic simulation of the dynamics of an 8-link all-revolute robot arm. We aim to predict the distance of the end-effector from a given target using 8 features.

- Pumadyn dataset [45], similar to Kinematics dataset, contains a realistic simulation of a Puma 560 robot arm. The goal is to predict the angular acceleration of one of the robot arm's links. There are 8,192 samples and 32 features.

We compare our model against six different models: the Naive model, autoregressive integrated moving average model with exogenous regressors (ARIMAX),

multilayer perceptron (MLP), light gradient boosting machine (LightGBM), the conventional LSTM architecture and the disjoint model composed of an LSTM and a hard tree-based gradient boosting machine as described in Section 2.3.1. The Naive model predicts $\hat{y}_t = y_{t-1}$ for all time steps, i.e., it carries the last measurement as its prediction for the next time step. ARIMAX model, as mentioned in Section 1, is a statistical model fitting an infinite impulse response (IIR) filter to the (possibly differenced) data with side information. We tune the number of AR and MA components using a stepwise algorithm outlined in [51]. The MLP we use in our experiments approximate a nonlinear autoregressive process with exogenous regressors (ARX), i.e., the values of the input neurons are the lagged values of the target signal as well as any side information the data provides. We use a one-layer shallow network for which we tune number of hidden neurons, learning rate and the activation function in the hidden layer (ReLU [52] is used for the output neuron). LightGBM is, as mentioned in Section 1, a hard tree-based gradient boosting machine, which, with well-designed features, achieves effective sequential data prediction. We tune the number of trees, learning rate, bagging fraction, maximum number of leaves and regularization constants for LightGBM. For the LSTM model, we tune the number of layers, number of hidden units in each layer and the learning rate. For our architecture, we also search for the pooling method P, number of soft trees $M + 1$, depth of each tree and the shrinkage rate ν in boosting. Among these six models, however, only the Naive model, MLP and LSTM are amenable to online learning; for the other 3 models we proceed as follows. For ARIMAX, we use the underlying Kalman filter to perform training; for LightGBM and the disjoint model, we would ideally perform re-training over the accumulated data at each time step; however, since this would already become time consuming, we introduce a queue of fixed size, say 50, and when it is fully filled, we append the new data to the historical data to refit the model; the queue is then emptied.

To assess the performance of online regression, we measure time accumulated mean squared errors of models, which is computed as the cumulative sum of MSE's normalized by the data length at a given time t. In Table 2.2, we report the cumulative error at the last time step for all models. We do not apply any

preprocessing to the data for the Naive, ARIMAX and LightGBM models, while applying the standardization described in Section 2.3 to other four models.

We observe that our joint model performs better on almost all datasets (except for Kinematics) over the last cumulative error metric against other models. In particular, in the Elevators dataset where the sample size is the largest, the difference in normalized cumulative error compared to other models is the greatest. This suggests that the baseline models are unable to learn from new data in long term as well as the joint architecture, featuring the role of the embedded soft GBDT as the final regressor. Similarly, in the Pumadyn dataset where the number of features is the largest, there is an order of magnitude difference in performances (except for MLP; the performance is still as twice as better). This verifies the importance of mitigating hand-designed features and letting LSTM in the front end to extract necessary information from raw features to feed to the regressor in the end. In addition, our end-to-end model consistently achieves much better performance than the disjoint architecture over all datasets, featuring the importance of joint optimization. In all datasets, Naive model has the highest errors as expected, and LightGBM model performs worse than MLP on all datasets; this is mainly because LightGBM is not tailored for online learning and required fully refitting, while MLP is amenable to do online updates using, for example, SGD. MLP performs slightly better than our model in Kinematics dataset; since MLP we used approximates a nonlinear autoregressive process, this implies that there might be high partial autocorrelation patterns in the particular dataset which MLP specifically focused on. Overall, the hybrid model achieves

	Bank	Elevators	Kinematics	Pumadyn
Naive	0.0307	0.0714	0.1305	0.0218
ARIMAX	0.0183	0.0465	0.0897	0.0107
MLP	0.0165	0.0043	**0.0766**	0.0019
LightGBM	0.0203	0.0234	0.0991	0.0143
LSTM	0.0190	0.0119	0.0884	0.0094
Disjoint	0.0236	0.0650	0.1102	0.0160
LSTM-SGBDT	**0.0151**	$\mathbf{2.671 \times 10^{-5}}$	0.0787	**0.0009**

Table 2.2: Cumulative errors of the models on Bank, Elevators, Kinematics and Pumadyn datasets.

better than both its disjoint counterpart and various standalone models including LSTM over real life datasets, exhibiting its effectiveness in an online setting.

2.3.3 M4 Competition Datasets

The M4 competition [47] provided 100,000 time series of varying length with yearly, quarterly, monthly, weekly, daily and hourly frequencies. In our experiments, we chose yearly, daily and hourly datasets. The yearly dataset consists of 23,000 very short series with the average length being 31.32 years. Therefore, inclusion of this dataset aims to assess the performance of the models under sparse data conditions. Next, the daily dataset has 4,227 very long series where the average length is 2357.38 days. This dataset helps assess how effective a model is in capturing long-term trends and time shifts. Lastly, the hourly dataset consists of 414 series. The reason for including this subset is the dominant seasonal component. We experiment over all of the hourly series and randomly selected 500 of daily and yearly series each. The prediction horizon for hourly, daily and yearly datasets are 48, 14 and 6 time steps, respectively. We aim for one-step ahead predictions in an offline setting; we train and validate the models over the training split of the data, and then predict the test split of the data one step at a time *without* refitting/updating the models with the arriving of new data.

We compare our model (LSTM-SGBDT) with six models presented in Section 2.3.2; except we use seasonal ARIMAX (SARIMAX) instead of ARIMAX in the hourly dataset to account for the seasonality (we report the model's name as SARIMAX in Table 2.3 for brevity, though no seasonal component was fitted for yearly and daily datasets). For each dataset, we separate a hold-out subseries from the end of each time series as long as the desired prediction horizon. For example, a series in the hourly dataset has its last 48 samples spared for validation of the hyperparameters. The searched hyperparameters of models are as described in Section 2.3.2. We use MAPE to evaluate models' performance across series. The preprocessing setup is the same as in the experiments of Section 2.3.2.

The mean MAPE scores of the models across each datasets are seen in Table

	Hourly	Daily	Yearly
Naive	0.50915	0.09608	0.67152
SARIMAX	0.15549	0.05425	0.20032
MLP	0.19380	0.02940	0.22447
LightGBM	0.19468	**0.01602**	0.38881
LSTM	0.20282	0.06448	0.18032
Disjoint	0.37580	0.07860	0.56895
LSTM-SGBDT	**0.12290**	0.01658	**0.15852**

Table 2.3: Mean MAPE performances of the models across M4 datasets.

2.3. The LSTM-SGBDT model achieves better performance than other models in hourly and yearly datasets; in the daily dataset, LightGBM has a slightly better error metric. In the yearly dataset, which has short-length series, LSTM-SGBDT model struggles less than other models against the sparse data. This indicates that soft GBDT part helps reduce overfitting of the data hungry LSTM network and acts as a regularizer. LSTM-SGBDT also outperforms the vanilla LSTM in the daily dataset with a wider margin of 74.3%. Given that the daily dataset consists of very long series, LSTM-SGBDT is able to model long-term dependencies better than LSTM. Lastly, our model achieves better performance across the whole M4 hourly dataset than all baseline models, and in particular 39.4% better than LSTM. Notably, the closest model turned out to be SARIMAX with seasonality period 24, which outperformed other machine learning-based models which are not directly modelled for seasonality unlike a part of SARIMAX. Thereby, this suggests that our model is better at detecting seasonality patterns, which are dominantly seen in this particular dataset. We note that no preprocessing specific to seasonality extraction has been made. Similar to the real life datasets, the Naive model has the highest error metric in all datasets, and the disjoint model performs consistently worse than our joint model. LightGBM and MLP performs comparably better than our baseline models, though they have higher errors in hourly and yearly datasets where seasonality and sparsity were emphasized, respectively. Overall, the hybrid model achieves better than both its disjoint counterpart and various standalone models including LSTM on average over three subset of M4 competition datasets of varying characteristics.

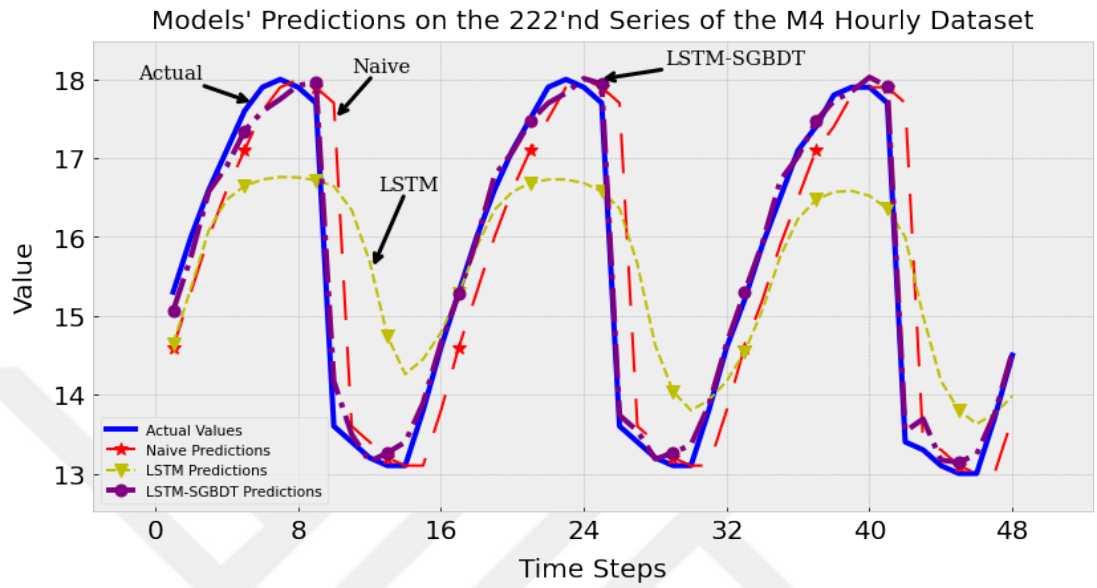

Figure 2.6: Naive, LSTM and LSTM-SGBDT predictions over the forecast horizon of an M4 hourly series.

2.3.3.1 Case Study on a Hourly Time Series

As a case study to show the effectiveness of our proposed model visually, we look at the testing part of an example series (222nd) from the hourly dataset of length 48 along with the predictions of LSTM, LSTM-SGBDT and the naive model. The plot of predictions are given in Fig. 2.6. The corresponding MAPE metrics of the naive, LSTM and LSTM-SGBDT models are 0.0417, 0.0723 and 0.0096, respectively. We observe that the vanilla LSTM model failed to capture the amplitude of the seasonal pattern. It also tends to predict the last observed value (i.e., the same as the naive predictions) for many time steps, which LSTM-based models are known to suffer from [53]. These lead to LSTM being a poorer predictor than the naive model in this specific time series. Our model, on the other hand, successfully captures the seasonal pattern and closely follows the actual values.

2.3.3.2 Recursive Prediction over the Hourly Dataset

In this section, we perform recursive prediction with our model. To this end, we use the entire hourly dataset that M4 competition provided [47]. There are in total 414 time series in this dataset, of which 245 are of length 1008 and the remaining 169 are of length 748. The competition demanded a 48-step prediction without revealing the true values as the prediction arrives, which distinguishes this task from the one-step ahead prediction we performed in the previous section. We use the last 48 samples of each series as a hold-out set, over which we validate our model. We note that the validation was performed for each series separately. We extend one-step ahead forecasting to this scenario as follows. We first predict the first sample of the test set. Then, treating this prediction as if it was the true value, we predict the second sample of the test set without refitting. This procedure recursively continues until we produce all of the 48 predictions. As for the model configuration, we use 1-layer LSTM network in the front end and use last pooling in its last layer. The tuned hyperparameters are number of hidden neurons for the LSTM network; number of trees, depth of each tree and shrinkage rate for the sGBDT. We also tune the number of epochs and the learning rate of the gradient-based updates. We use the same preprocessing procedure as described in the previous section. For the performance metric, to comply with the M4 competition results, we use symmetric mean absolute percentage error (sMAPE), which is given as

$$\text{sMAPE}(\boldsymbol{d}_H, \hat{\boldsymbol{d}}_H) = \frac{2}{H} \sum_{i=1}^{H} \frac{|d_{H,i} - \hat{d}_{H,i}|}{|d_{H,i}| + |\hat{d}_{H,i}|},$$

where H is the prediction horizon (e.g, 48 hours), $\boldsymbol{d}_H$ and $\hat{\boldsymbol{d}}_H$ denote the vector of true and predicted values over the horizon, respectively.

There are 29 sMAPE results for the hourly dataset reported in the M4 competition [47], some of which are benchmark results, e.g., simple exponential smoothing. For brevity, we do not include the results that were ranked after (and including) the best benchmark result, which was the multi-layer perceptron benchmark. Therefore, 12 results including ours are reported in Table 2.4 in sorted order.

We note that the contestants had come up with models that were tailored for the dataset, performed extensive feature engineering, and also tuned data preprocessing strategies during the competition. Our model, on the other hand, employs no special feature engineering and only applies standardization as the preprocessing step (as done in all of the experiments in Section 3). We only used the past values of the target signal for each series, and due to its generic nature, the LSTM network on the front end of the model is tasked to generate relevant features from this raw sequential data and feed it to the supervised regressor sGBDT on the back end. We observe that our model still achieves a comparable performance among contestants when using the recursive estimation strategy over this dataset.

2.3.4 Financial Datasets

In this section, we show the generic nature of our framework. To this end, we replace the LSTM network in the front end of the model with a gated recurrent unit (GRU) network [23]. A GRU cell, similar to an LSTM cell, is endowed with gated routing mechanisms in order to avoid the vanishing gradient problem [54]. In particular, the forward pass equations in one cell are described as

$$\begin{aligned} \boldsymbol{z}_t &= \sigma(\boldsymbol{W}_z \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_z) \\ \boldsymbol{r}_t &= \sigma(\boldsymbol{W}_r \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_r) \\ \tilde{\boldsymbol{h}}_t &= \tanh(\boldsymbol{W}_h \{\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_h) \\ \boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t, \end{aligned} \tag{2.14}$$

Doornik et al.	Smyl	Pawlikowski et al.	Pedregal et al.
8.913	9.328	9.611	9.765
Jaganathan & Prakash	Montero-Manso et al.	Darin & Stellwagen	LSTM-SGBDT
9.934	11.506	11.683	11.871
Waheeb	Roubinchtein	Petropoulos & Svetunkov	Shaub
12.047	12.871	13.167	13.466

Table 2.4: Mean sMAPE (%) metrics of the selected 12 models for the recursive estimation experiment over 414 hourly time series from the M4 dataset.

where $\boldsymbol{h}_t$ is the hidden state vector, $\boldsymbol{x}_t$ is the input to the cell, $\boldsymbol{h}_{t-1}$ is the hidden state from the previous cell and $\{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\}$ denotes the vertical concatenation of $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. $\boldsymbol{z}_t$ and $\boldsymbol{r}_t$ are the update and reset gate vectors, respectively, and $\tilde{\boldsymbol{h}}_t$ is the candidate state vector, whose corresponding stacked weight matrices are $\boldsymbol{W}_z, \boldsymbol{W}_r$ and $\boldsymbol{W}_h$. The corresponding biases are represented by $\boldsymbol{b}_z, \boldsymbol{b}_r$ and $\boldsymbol{b}_h$. σ denotes the logistic sigmoid function and tanh is the hyperbolic tangent function; both apply point-wise. To implement a GRU-SGBDT, we directly replace the LSTM equations with GRU equations; this shows the flexibility of the framework for the sequential feature extraction.

We consider four models in this section: LSTM, GRU, LSTM-SGBDT and GRU-SGBDT. We evaluate the performances of the models under two financial scenarios in an online setting. We first consider the Alcoa stock price dataset [48], which contains the daily stock price values between the years 2000 and 2020. We aim to predict the future prices values by considering the past prices. We only apply standardization to the price values in a cumulative manner. For standardization, we update the stored mean and standard deviation as more data are revealed. We hold out 10% of the data from the beginning and use it for the hyperparameter validation. We choose a window size of 5 so that the last five trading days' stock prices are used by the models to predict the today's price value. Tuned hyperparameters are the same as those in M4 datasets. Fig. 2.7 illustrates the performance of the models as the data length varies. We observe that hybrid models consistently achieve lower accumulated errors than the standalone recurrent models, showing the benefits of the proposed framework. GRU network starts off with a huge error margin but closes the gap after 3000 data points, although it does not surpass LSTM overall. LSTM-SGBDT performs closer to GRU-SGBDT yet better in each time step, though the gap is essentially closed when all data are revealed. In addition, LSTM and GRU models are rather slower to react to the abrupt change in the stock prices around length of 2,000 which corresponds to the global financial crisis in the late 2008.

Apart from the Alcoa stock price dataset, we also experiment with the Hong Kong exchange rate dataset [49], which has the daily rate for Hong Kong dollars

against US dollars between the dates 2005/04/01 and 2007/03/30. Our goal is to predict the future exchange rates by using the data of the previous five days. We use the same setup for the hyperparameter configuration as in the Alcoa dataset. The time accumulated errors for both models are presented in Fig. 2.8. We observe that while all models follow a similar error pattern, our hybrid models again consistently achieve a lower cumulative error than the standalone models for almost all time steps. The cumulative errors of the two hybrid models follow each other quite closely, although GRU-SGBDT has a slightly better final accumulated error than LSTM-SGBDT. We also notice that the gap between the errors of standalone models and the hybrid models becomes wider as more data are revealed.

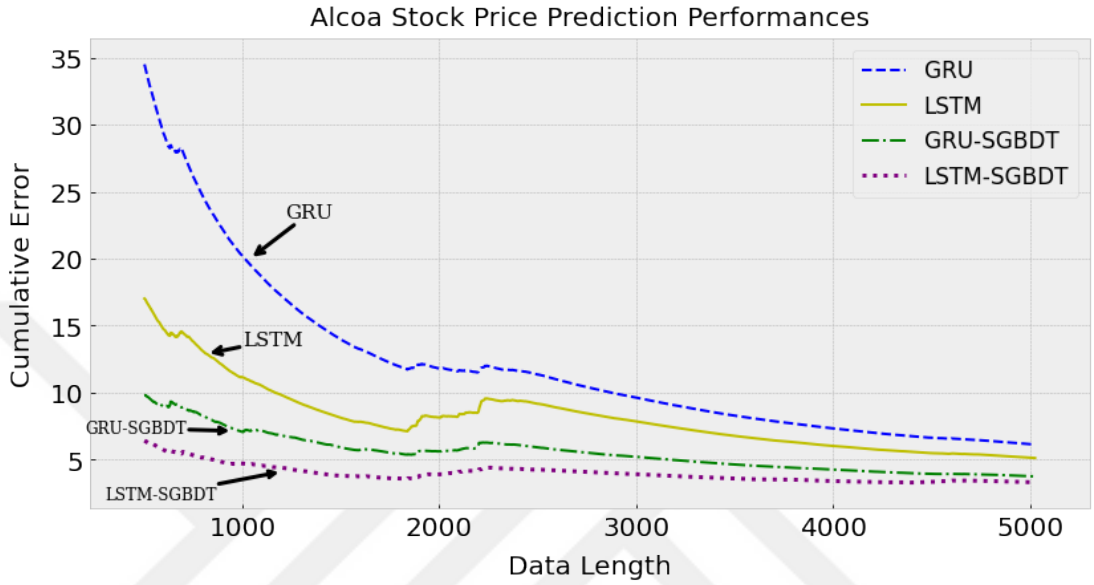

Figure 2.7: Time accumulated errors of standalone and hybrid models for the Alcoa stock price dataset for the period 2000-2020.

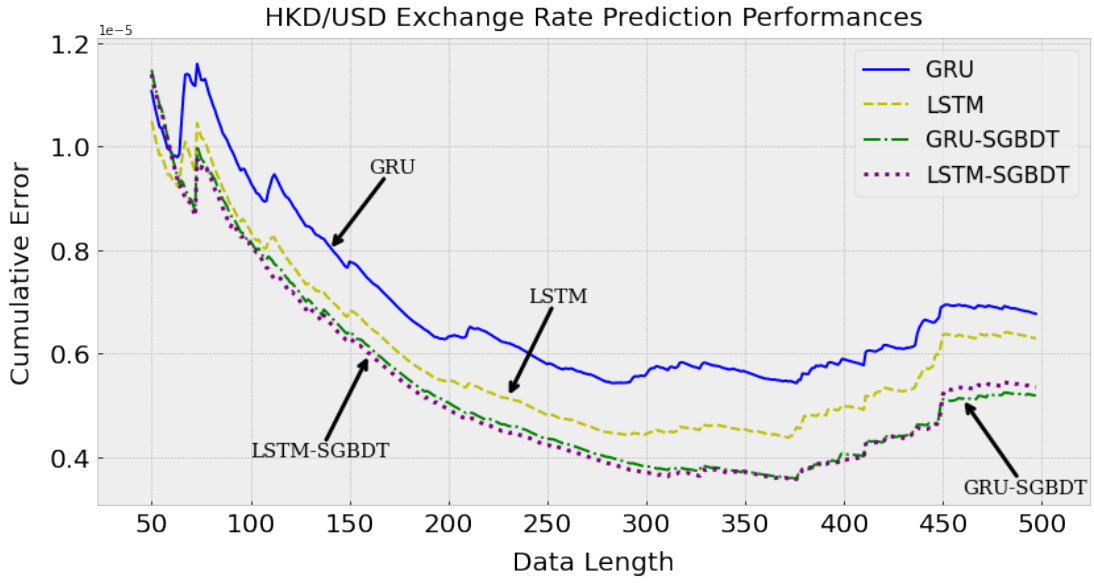

Figure 2.8: Time accumulated errors of standalone and hybrid models for the Hong Kong exchange rate against U.S. dollars.

Chapter 3

A Traditional Model Meets RNNs: A Joint State Space Architecture for SARIMAX and LSTM

Here, we propose to unify a time series-specific model and a recurrent neural network to achieve a "separation of duties" in prediction, i.e., the RNN is tasked for automatic feature extraction and revealing the nonlinear relations while a time series-specific model, e.g., SARIMAX, is employed for trend & seasonality detection in a linear manner. Moreover, the exogenous variables, i.e., side information that help in prediction, undergo numerous nonlinear transformations in an RNN structure to finally affect the ultimate prediction of the sequential data, while in a less complex SARIMAX model, there is a direct linear link between side information and the desired signal. Therefore, we aim to combine an RNN with a SARIMAX model side by side to bring their prominent features seamlessly for effective prediction while enjoying joint optimization of the parameters of the entire model. In particular, we connect the two models side-by-side by putting them into a joint nonlinear state space formulation and jointly optimizing the whole architecture using particle filtering. In order to achieve online learning, we

introduce a novel state space formulation for the SARIMAX model that allows for a single-pass solution for its parameter estimation, which also paves the way for the concatenation of state spaces. We highlight that our framework is generic, i.e., one can use other recurrent network architectures for the nonlinear part (such as simple RNNs and GRUs) and traditional time series models (such as ETS) for the linear part as long as there exists a state space representation for them.

3.1 Related Work

Neural networks have been used for time series prediction for decades now [55, 56]. With the introduction of powerful recurrent neural networks [40], focus shifted towards RNN-based solutions for time series tasks. Even though they are not particularly designed for the task, their sequential data processing power easily allowed them to be popular in the area [57, 58]. However, despite their ability to work with raw features, neural networks needed a lot of domain-specific preprocessings and feature engineering for sequential data prediction tasks [59, 60]; otherwise they are known to converge to a naive predictor, i.e., a simple model carrying the last observation as the prediction [53,61]. On the other hand, traditional time series models, such as ARMA and ETS, date back even more than neural networks. They have generally fewer parameters to estimate, easier to interpret due to the intuitive nature of the underlying model, are robust to overfitting and amenable to automated procedures for hyperparameter selection [7]. Since they are tailored specifically for time series regression, they are still employed in contemporary forecasting repertoires [7]. Nevertheless, they have highly strong assumptions about the data such as stationarity and linearity, which prevent them from capturing nonlinear patterns that real life data tend to possess [8]. Therefore, both the (nonlinear) complex recurrent networks and the (linear) traditional models have their drawbacks when it comes to time series prediction.

To overcome the individual weaknesses of each model families, there have been several works on combining an RNN with an ARMA model. In [62], authors

successively combine an ARMA model with a recurrent neural network after preprocessing with wavelet transformations for solar power prediction. However, the combination is done in a *disjoint* manner, i.e., only after an ARMA model is fit on the data, is the RNN employed on the residuals. This disjoint approach not only increases the computational time but also retains it from enjoying the benefits of a joint optimization of an integrated model [35, 36]. Moreover, they focus on solar power prediction, which renders the overall model domain specific. In [63], authors alter the recurrent relation of a vanilla RNN for its hidden state via an inspiration from ARMA models by incorporating the effects of lagged input for the state transition. As their experiments show, however, this does not cause a dramatic increase in performance compared to the vanilla RNN structure, which is possibly because the side information sequence, which does not directly affect the final output in their framework, can be highly auto-correlated and recent lagged values do not carry as much information. Furthermore, they focus on sequence classification tasks and do not address time series prediction. In [64], inspiration is this time taken from RNN models towards an ARMA-based model. Authors subject AR and MA components, i.e., y_{t-i} and ε_{t-j} to sigmoidal nonlinearities borrowed from RNNs. Nonetheless, this does not bring the full capabilities of an enhanced RNN, e.g., the gating mechanisms for efficient information flow is missing. Overall, the contemporary literature addresses the unification of models either in a disjoint manner or via borrowing ideas from one model family to another to implement partial integration. Unlike previous studies, our model is geared with joint optimization of a nonlinear and recurrent sequential feature extractor (e.g., LSTM) and a linear time-series specific model (e.g., SARIMAX) for online learning, where both models are unified through a state space formulation thanks to our novel state space structure for ARMA family models.

3.2 Preliminaries

3.2.1 Problem Statement

We study the online regression of sequential data. We observe a sequence $\{y_t\}$ along with a side information sequence $\{\boldsymbol{s}_t\}$. At each time t, given the past information $\{y_k\}$, $\{\boldsymbol{s}_k\}$ for $k \leq t$, we produce the prediction $\hat{y}_{t+1}$. Therefore, in this purely online setting, our goal is to find the relationship

$$\hat{y}_{t+1} = f\big([\dots, y_{t-1}, y_t], [\dots, \boldsymbol{s}_{t-1}, \boldsymbol{s}_t]\big),$$

where f is an unknown function, which models $\hat{y}_{t+1}$. In our framework, $\hat{y}_{t+1}$ is formed as a sum of two components: prediction from the linear and the nonlinear state space models, i.e.,

$$\hat{y}_{t+1} = \hat{y}^{(n)}_{t+1} + \hat{y}^{(l)}_{t+1},$$

where $\hat{y}^{(n)}_{t+1}$ and $\hat{y}^{(l)}_{t+1}$ denote the nonlinear and linear parts, e.g., coming from an LSTM network and a SARIMAX model, respectively. We note that even though they are separate entities, the framework encapsulates them both in a unified state space representation and optimizes the underlying models jointly. Once y_{t+1} is revealed, we suffer the loss

$$L_{t+1} = \ell(y_{t+1}, \hat{y}_{t+1}),$$

where ℓ can be, for example, squared error loss, i.e., $\ell(y_{t+1}, \hat{y}_{t+1}) = (y_{t+1} - \hat{y}_{t+1})^2$. As an example, in weather nowcasting, temperature (y_t) for the next hour is predicted using the hourly measurements of the past. Side information could include the wind and humidity levels ($\boldsymbol{s}_t$). The model adapts itself whenever new observations of the next hour become available, i.e., works in an online manner.

We use RNNs as the nonlinear part of the framework. An RNN is described by the recursive equation [10]

$$\boldsymbol{h}_t = u(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}),$$

where $\boldsymbol{x}_t$ is the input vector and $\boldsymbol{h}_t$ is the state vector at time instant t. $\boldsymbol{W}_{ih}$ and $\boldsymbol{W}_{hh}$ are the input-to-hidden and hidden-to-hidden weight matrices (augmented

to include biases), respectively; u is a nonlinear function that applies element wise. As an example, for the prediction task, one can use $\boldsymbol{s}_t$ directly as the input vector, i.e., $\boldsymbol{x}_t = \boldsymbol{s}_t$, or some combination of the past information $\{y_t, \ldots, y_{t-r+1}\}$ with a window size r along with $\boldsymbol{s}_t$ to construct the input vector $\boldsymbol{x}_t$. The final output of the RNN can be produced by passing $\boldsymbol{h}_t$ through a dense layer on top:

$$\hat{y}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{h}_t, \tag{3.1}$$

where $\boldsymbol{w}_t, \boldsymbol{h}_t \in \mathbb{R}^m$.

To effectively deal with the temporal dependency in sequential data, we use a specific kind of RNN, the LSTM neural networks [40]. Here we use the variant with the forget gates and without peephole connections [41]. The forward pass equations in one cell are described as

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_f) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned} \tag{3.2}$$

where $\boldsymbol{c}_t$ is the state vector, $\boldsymbol{x}_t$ is the input to the cell, $\boldsymbol{h}_{t-1}$ is the hidden state from the previous cell and $\{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\}$ denotes the vertical concatenation of $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. $\boldsymbol{f}_t, \boldsymbol{i}_t, \tilde{\boldsymbol{c}}_t, \boldsymbol{o}_t$ are the forget gate, input gate, block input and output gate vectors, respectively, whose stacked weight matrices are $\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_c$ and $\boldsymbol{W}_o$. The corresponding biases are represented by $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c$ and $\boldsymbol{b}_o$. σ denotes the logistic sigmoid function, i.e, $\sigma(x) = 1/(1 + \exp(-x))$ and tanh is the hyperbolic tangent function, i.e, $\tanh(x) = 2\sigma(2x) - 1$; both are nonlinear activation functions that apply point-wise. $\odot$ is the Hadamard (element-wise) product operator. By allowing a constant error flow through the cell state in backpropagation, LSTMs greatly prevent vanishing gradients, and nonlinear gated relations described by (2.1) manage the information flow effectively to handle long term dependencies [22].

As for the linear part of the framework, we employ the seasonal auto-regressive

integrated moving average with exogenous regressors (SARIMAX) model [65,66]. Tailored specifically for time series modelling, SARIMAX builds on the conventional ARMA architecture [67] via considering the difference(s) of the target signal where seasonal effects and side information are also incorporated. The ARMA model of order (p, q) on y_t corresponds to

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t, \tag{3.3}$$

where $\{\varepsilon_t\}$ is a white Gaussian noise sequence, and ϕ_i and θ_j are the parameters to estimate for the auto-regressive and the moving average parts, respectively. To simplify the expression, we use the "lag notation", with which (3.3) becomes

$$\phi_p(B)\, y_t = \theta_q(B)\, \varepsilon_t,$$

where the lag polynomials are defined as $\phi_p(B) := (1 - \phi_1 B - \ldots - \phi_p B^p)$ and $\theta_q(B) := (1 + \theta_1 B + \ldots + \theta_q B^q)$ where B is the backshift (lag) operator defined as $Bx_t := x_{t-1}$ for any sequence x_t such that the recurrence $B^k x_t = B(B^{k-1} x_t)$ holds for all integer $k >= 2$. The SARIMAX model of order (p, d, q) and seasonal order (P, D, Q) with seasonality m is then expressed as

$$\begin{aligned} \tilde{y}_t &= \Delta_d(B)\, \Delta_D(B^m)\, y_t \\ \phi_p(B)\, \Phi_P(B^m)\, \tilde{y}_t &= \theta_q(B)\, \Theta_Q(B^m)\, \varepsilon_t + \boldsymbol{\beta}_t^T \boldsymbol{x}_t, \end{aligned} \tag{3.4}$$

where $\Delta_h := (1 - B - \ldots - B^h)$ is the difference operator, $\{\tilde{y}_t\}$ is the differenced sequence, $\boldsymbol{x}_t$ is the side information vector, i.e., exogenous context-based data to help in prediction and $\boldsymbol{\beta}_t$ is the coefficient vector associated with it.

In the following section, we first present state space models for both LSTM and SARIMAX architectures, for the latter of which we introduce a novel representation that requires only a single pass over the data for parameter estimation. We then introduce a joint state space model to accommodate both LSTM and SARIMAX models in a unified manner to ease optimization of the end-to-end model. Building on top of that, we will propose a generic particle filter-based solution for the sequential prediction problem at hand.

3.3 The proposed model

We next introduce a hybrid model which aims to boost the performance of RNNs in time series prediction using a time series-specific model as auxiliary while enjoying joint optimization.

Recurrent neural networks are generally trained with backpropagation through stochastic gradient descent (SGD) [22]. Time series specific models such as SARIMAX (or ETS), on the other hand, are usually trained with Bayesian estimation methods with iterative optimization, e.g., maximum likelihood estimation (MLE) where no prior information on parameters is assumed, after they are put in a state space. A gradient-based solution is rendered ineffective for these type of models mainly because they include "unobserved" components, i.e., the current and past errors (shocks) made by the model, for which gradient calculations are not possible as they are unknown. Even though there exist methods to approximate these current and lagged errors, e.g., Hannah & Rissanen's [68] technique of replacing the MA(q), i.e., the error related parts of a SARIMAX model with a sufficiently large AR(p) model, these are not suitable for online prediction as they require continuous re-fitting in, e.g., a sliding window manner to keep track of the unobserved components. Therefore, to connect an RNN and a SARIMAX model, we propose a nonlinear joint state space model which can be efficiently trained with, e.g., particle filtering. Nevertheless, the contemporary state space formulations for SARIMAX modeling, e.g., Harvey's [69] or Hamilton's [70] do not put the parameters to be estimated directly into the state vector but instead mix it with the unobserved components for a more compact state representation. This, however, renders single-pass solutions, e.g., with a Kalman filter [71], unsuitable for parameter estimation since the matrices in the formulation of the state spaces cannot be built out of unknown components. Then, instead of a single pass estimation, an iterative optimization based on MLE is used, which requires many passes over the time series that is not possible for online prediction. Therefore, we introduce a novel state space formulation for SARIMAX that puts the parameters into the state vector and excludes the unobserved components. This not only allows for a single-pass solution suitable for online prediction but also makes

it possible to concatenate state space representations with that of an RNN for a joint optimization.

3.3.1 State Space Formulations

Here, we first present the state space formulations of the base models. Then, we join them for the hybrid architecture and propose a particle filter-based solution for the prediction problem.

3.3.1.1 LSTM

As presented in Eqs. (2.1), an LSTM network has two inherent state variables: cell state $\boldsymbol{c}_t$ and hidden state $\boldsymbol{h}_t$. Their (nonlinear) evolution will be captured by the state transition equations as

$$\boldsymbol{c}_t = \gamma\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}\right) + \boldsymbol{u}_t \tag{3.5}$$

$$\boldsymbol{h}_t = \tau\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_t\right) + \boldsymbol{v}_t, \tag{3.6}$$

where γ and τ represent the nonlinear relations depicted in Eqs. (3.2), and $\boldsymbol{u}_t$ and $\boldsymbol{v}_t$ are the noise components. We not only estimate the internal states of an LSTM cell but also the parameters, i.e., weights and biases in Eqs. (3.1) and (3.2); for brevity, we collect all of these coefficients in a large parameter vector $\boldsymbol{\theta}_t \in \mathbb{R}^{n_\theta}$ which includes ravelled $\boldsymbol{W}_*$ matrices and $\boldsymbol{b}_*$ vectors for $* = \{f, i, c, o\}$ and the final regressor vector $\boldsymbol{w}_t$. Assuming a state size of k and input dimension of l, i.e., $\boldsymbol{h}_t \in \mathbb{R}^k$ and $\boldsymbol{x}_t \in \mathbb{R}^l$, we have $n_\theta = 4\left(k(k+l)+k\right) + k$. In order to put these parameters into the state equation, we will assume a time-varying system where the parameters will be allowed to evolve with random noise in state transition. This will also allow modeling a non-stationary process in general as a byproduct. Given this, and letting $\boldsymbol{s}_{LSTM}$ be the encompassing state vector covering the internal state as well as the weights & biases of the network, we

have the overall state transition equation for an LSTM cell as

$$\boldsymbol{s}_{t,LSTM} = \begin{bmatrix} \boldsymbol{c}_t \\ \boldsymbol{h}_t \\ \boldsymbol{\theta}_t \end{bmatrix} = \begin{bmatrix} \gamma\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}\right) \\ \tau\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_t\right) \\ \boldsymbol{\theta}_{t-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}_t \\ \boldsymbol{v}_t \\ \boldsymbol{\epsilon}_t \end{bmatrix}, \tag{3.7}$$

where $\boldsymbol{\epsilon}_t$ is the noise vector driving the parameters of LSTM. Lastly, the measurement equation is given as

$$y_{t,LSTM} = \boldsymbol{w}_t^T \boldsymbol{h}_t + \varepsilon_t, \tag{3.8}$$

where ε_t is a scalar noise, which is assumed to be independent of $\boldsymbol{u}_t$, $\boldsymbol{v}_t$ and $\boldsymbol{\epsilon}_t$ of the state transition. We note that this is the only assumption we make about the noise quantities, e.g., no Gaussian white noise assumption is made.

3.3.1.2 SARIMAX

We first start with the state space formulation of an ARMA(p, q) model, where p and q are the order of auto-regressive and moving average parts, respectively, and then extend the representation to a SARIMAX$(p, d, q)(P, D, Q, m)$ model where d and D are the order of ordinary and seasonal differences to the sequence, P and Q are the seasonal counterparts of p and q, and m represents the period of seasonality of the sequence, if any, e.g., 24 for hourly data. Unlike the current state space approaches for the ARMA model, we (only) put parameters to be estimated into the state vector. From Eq. (3.3), we introduce time variability to the AR and MA parameters as

$$y_t = \sum_{i=1}^{p} \phi_{t,i}\, y_{t-i} + \sum_{j=1}^{q} \theta_{t,j}\, \varepsilon_{t-j} + \varepsilon_t, \tag{3.9}$$

i.e., instead of static ϕ_i and θ_j parameters, we now have evolving coefficients to estimate, which is in compliance with online learning. Let $\boldsymbol{\Phi}_t = [\phi_{t,1}, \phi_{t,2}, \ldots, \phi_{t,p}]^T$ and $\boldsymbol{\Theta}_t = [\theta_{t,1}, \theta_{t,2}, \ldots, \theta_{t,q}]^T$. Then, for the state vector $\boldsymbol{s}_{t,ARMA} := [\boldsymbol{\Phi}_t^T, \boldsymbol{\Theta}_t^T]^T \in \mathbb{R}^{p+q}$, we introduce the state transition and the measurement equation as

$$\boldsymbol{s}_{t,ARMA} = \boldsymbol{s}_{t-1,ARMA} + \boldsymbol{e}_t \tag{3.10}$$

$$y_{t,ARMA} = \boldsymbol{r}_t^T \boldsymbol{s}_{t,ARMA} + \varepsilon_t, \tag{3.11}$$

where $\boldsymbol{e}_t$ is the noise vector driving the evolution of the parameters, and $\boldsymbol{r}_t := [y_{t-1,ARMA}, \dots y_{t-p,ARMA}, \varepsilon_{t-1}, \dots \varepsilon_{t-q}]^T$. We note that all components of the vector $\boldsymbol{r}_t$ are known as it comprises only of past data, i.e., at time t, for $i > 0$, all $y_{t-i,ARMA}$ are observed and ε_{t-i} are unknown yet estimated as $\hat{\varepsilon}_{t-i} = y_{t-i,ARMA} - \hat{y}_{t-i,ARMA}$. Now we extend this representation for a SARIMAX$(p,d,q)(P,D,Q,m)$ model described in Eqs. (3.4), which can be straightforwardly done via augmenting $\boldsymbol{s}_{t,ARMA}$ and $\boldsymbol{r}_t$ vectors in Eqs. (3.10) with the seasonal and exogenous coefficients. Then, we have

$$
\begin{aligned}
\tilde{\boldsymbol{\Phi}}_t &= [\phi_{t,1}, \dots, \phi_{t,p}, \phi_{t,m}, \dots, \phi_{t,mP}]^T \\
\tilde{\boldsymbol{\Theta}}_t &= [\theta_{t,1}, \dots, \theta_{t,q}, \theta_{t,m}, \dots, \theta_{t,mQ}]^T \\
\boldsymbol{s}_{t,SX} &:= [\tilde{\boldsymbol{\Phi}}_t^T, \tilde{\boldsymbol{\Theta}}_t^T, \boldsymbol{\beta}_t^T]^T \\
\tilde{\boldsymbol{r}}_t &:= [y_{t-1,SX}, \dots, y_{t-p,SX}, \\
&\qquad y_{t-m,SX}, \dots, y_{t-mP,SX}, \\
&\qquad \varepsilon_{t-1}, \dots, \varepsilon_{t-q}, \varepsilon_{t-m}, \dots, \varepsilon_{t-mQ}, \\
&\qquad x_{t,1}, \dots, x_{t,d}]^T \\
\boldsymbol{s}_{t,SX} &= \boldsymbol{s}_{t-1,SX} + \boldsymbol{e}_t \\
y_{t,SX} &= \tilde{\boldsymbol{r}}_t^T \boldsymbol{s}_{t,SX} + \varepsilon_t.
\end{aligned}
\tag{3.12}
$$

In Eq. (3.12), the size of the state is $p+q+P+Q+d$ which is exactly the number of parameters to estimate in a SARIMAX model. We note that we assume the sequence y_t is already ordinarily and/or seasonally differenced so that we do not account for differencing in the state space formulation, as in the Hamiltonian representation [70], and differences will be undone in prediction time, if any.

Given Eqs. (3.7), (3.8) and (3.12), we can now concatenate the state vectors of an LSTM network and a SARIMAX model as

$$
\begin{aligned}
\boldsymbol{s}_t &:= \begin{bmatrix} \boldsymbol{s}_{t,LSTM} \\ \boldsymbol{s}_{t,SX} \end{bmatrix} \\
&= [\boldsymbol{c}_t^T, \boldsymbol{h}_t^T, \boldsymbol{\theta}_t^T, \tilde{\boldsymbol{\Phi}}_t^T, \tilde{\boldsymbol{\Theta}}_t^T, \boldsymbol{\beta}_t^T]^T
\end{aligned}
\tag{3.13}
$$

and arrive at the hybrid state space as

$$\begin{aligned} \boldsymbol{s}_t &= \Omega(\boldsymbol{s}_{t-1}) + \boldsymbol{\eta}_t \\ y_t &= y_{t,LSTM} + y_{t,SX} + \varepsilon_t \\ &= \boldsymbol{w}_t^T \boldsymbol{h}_t + \tilde{\boldsymbol{r}}_t^T \boldsymbol{s}_{t,SX} + \varepsilon_t \\ &= \boldsymbol{\lambda}_t^T \boldsymbol{s}_t + \varepsilon_t, \end{aligned} \tag{3.14}$$

where $\boldsymbol{\eta}_t$ is a noise vector, $\Omega(.)$ is a nonlinear function that applies the nonlinearities depicted in Eq. (3.7) to LSTM's internal state vectors $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ in $\boldsymbol{s}_t$, and otherwise behaves as a shifting operator, e.g., yields $\tilde{\boldsymbol{\Phi}}_{t-1}$ when subjected to $\tilde{\boldsymbol{\Phi}}_t$, and $\boldsymbol{\lambda}_t$ is the gathered state coefficient vector comprising of zeros corresponding to $\boldsymbol{c}_t$ and $\boldsymbol{\theta}_t$ and otherwise carries $\boldsymbol{w}_t$ and $\tilde{\boldsymbol{r}}_t$. As for the measurement equation, we directly sum up the base models' predictions for the ultimate estimation. For the noises, we only assume that ε_t is independent of each component of $\boldsymbol{\eta}_t$. In the next section, we present a particle filtering-based solution to the nonlinear system in (3.14).

3.3.2 Estimation with Particle Filtering

Here, we introduce a particle filtering-based algorithm for addressing the state space in (3.14). Our main aim is to estimate the recursively defined state vector $\boldsymbol{s_t}$ at each time step in an online manner after having observed measurements $\boldsymbol{y}_t$ by then. In a probabilistic point of view, this translates to obtaining (or approximating) the conditional distribution $p(\boldsymbol{s_t} \,|\, \boldsymbol{y}_{0:t})$. As the state vector is recursively evolving, a recursive formulation for this distribution can be found by noting that the conditional joint distribution of $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_t$ can be factored as

$$p(\boldsymbol{s}_{0:t} \,|\, \boldsymbol{y}_{0:t}) = Z\, p(\boldsymbol{s}_0 \,|\, \boldsymbol{y}_0) \prod_{i=1}^{t} p(\boldsymbol{y}_i \,|\, \boldsymbol{s}_i) p(\boldsymbol{s}_i \,|\, \boldsymbol{s}_{i-1}), \tag{3.15}$$

where Z is a constant normalizing factor. Using Bayes' theorem and with some algebra, we arrive from (3.15) to the recursive relation

$$p(\boldsymbol{s}_{0:t} \,|\, \boldsymbol{y}_{0:t}) = \frac{p(\boldsymbol{y}_t \,|\, \boldsymbol{s}_t)\, p(\boldsymbol{s}_t \,|\, \boldsymbol{s}_{t-1})}{p(\boldsymbol{y}_t \,|\, \boldsymbol{y}_{0:t-1})} p(\boldsymbol{s}_{0:t-1} \,|\, \boldsymbol{y}_{0:t-1}), \tag{3.16}$$

i.e., we can reach the desired distribution $p(\boldsymbol{s}_{0:t} \,|\, \boldsymbol{y}_{0:t})$ recursively via $p(\boldsymbol{s}_{0:t-1} \,|\, \boldsymbol{y}_{0:t-1})$. However, due to analytical intractabilities, e.g., the right hand side in Eq. (3.16) requiring analytically unsolvable integrals, we proceed with approximations. To this end, the particle filtering theory proposes to approximate the distributions containing the state vector in Eq. (3.16) via a discrete random measure [25], i.e.,

$$p(\boldsymbol{s}) \approx \sum_{i=1}^{N} w^{(i)} \delta(\boldsymbol{s} - \boldsymbol{s}^{(i)}), \tag{3.17}$$

where $\delta(.)$ is the Dirac's impulse function, $\boldsymbol{s}^{(i)}$ are the *particles* simulating the state, $w^{(i)}$ are the *weights* associated with each particle and N is the total number of particles. Formally, a discrete random measure $\mathcal{S} := \{\boldsymbol{s}^{(i)}, w^{(i)}\}$ for $i \in [1, N]$ is providing the random particle-weight pairs. With this notion, the desired optimal state estimate (in the mean square error sense) can be approximated as

$$\begin{aligned} \mathbb{E}[\boldsymbol{s}_t \,|\, \boldsymbol{y}_{0:t}] &\approx \int \boldsymbol{s}_t \sum_{i=1}^{N} w_t^{(i)} \delta(\boldsymbol{s}_t - \boldsymbol{s}_t^{(i)}) \, d\boldsymbol{s}_t \\ &= \sum_{i=1}^{N} w_t^{(i)} \boldsymbol{s}_t^{(i)}, \end{aligned} \tag{3.18}$$

i.e., a weighted sum of the particles. The random measure $\mathcal{S}$ providing the particles is tied to the true conditional distribution of the state vector; however, it is in general unsuitable to sample from that distribution as it is unknown. Therefore, an easier-to-sample distribution which is related to the true distribution up to a proportionality is introduced, which is called the importance function [25]. To this end, using the following equivalence

$$\begin{aligned} \mathbb{E}_p[f(x)] &= E_q\left[\frac{p(x)}{q(x)} f(x)\right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(x_i)}{q(x_i)} f(x_i), \end{aligned} \tag{3.19}$$

where $\mathbb{E}_r$ denotes the expectation with respect to the distribution $r(.)$, we reach

$$\begin{aligned} \mathbb{E}_p[\boldsymbol{s}_t \,|\, \boldsymbol{y}_{0:t}] &= \mathbb{E}_q\left[\boldsymbol{s}_t \, \frac{p(\boldsymbol{s}_t \,|\, \boldsymbol{y}_{0:t})}{q(\boldsymbol{s}_t \,|\, \boldsymbol{y}_{0:t})} \,\middle|\, \boldsymbol{y}_{0:t}\right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(\boldsymbol{s}_t^{(i)} \,|\, \boldsymbol{y}_{0:t})}{q(\boldsymbol{s}_t^{(i)} \,|\, \boldsymbol{y}_{0:t})} \, \boldsymbol{s}_t^{(i)}. \end{aligned} \tag{3.20}$$

Then, it follows from Eqs. (3.18) and (3.20) that, the (unnormalized) weights are

$$w_t^{*(i)} = \frac{p(\boldsymbol{s}_t^{(i)} \mid \boldsymbol{y}_{0:t})}{q(\boldsymbol{s}_t^{(i)} \mid \boldsymbol{y}_{0:t})}, \tag{3.21}$$

from which the particle weights of $\mathcal{S}$ are found via

$$w_t^{(i)} = \frac{w_t^{*(i)}}{\sum_{i=1}^{N} w_t^{*(i)}}. \tag{3.22}$$

Since the weights are time-varying in this online learning framework, we derive a transition equation for the weights as well. This again relies on the recursive nature of the problem, by which we arrive at [72]

$$w_t^{(i)} = \frac{p(y_t|\boldsymbol{s}_t^{(i)})\, p(\boldsymbol{s}_t^{(i)}|\boldsymbol{s}_{t-1}^{(i)})}{q(\boldsymbol{s}_t^{(i)}|\boldsymbol{s}_{t-1}^{(i)}, y_t)} w_{t-1}^{(i)}. \tag{3.23}$$

We choose the importance distribution $q(.)$ in Eq. (3.23) to be the *prior* importance function as commonly done in the literature to minimize the variance of the weights over time with easier computational conditions [25]; hence, we have $q(\boldsymbol{s}_t^{(i)} \mid \boldsymbol{s}_{t-1}^{(i),y_t}) := p(\boldsymbol{s}_t^{(i)} \mid \boldsymbol{s}_{t-1}^{(i)})$. Then, Eq. (3.23) simplifies to give the transition of weights as

$$w_t^{(i)} = p(y_t|\boldsymbol{s}_t^{(i)})\, w_{t-1}^{(i)}. \tag{3.24}$$

Lastly, to deal with the well-known degeneracy problem [25, 72], where the variance of weights becomes too high to skew the weight distribution to an impulsive one, i.e., vast majority of the particles are appointed with nearly zero weights. This leads to undeserved attention to those weights in updating them via Eq. (3.24). This issue is addressed with the *effective* sample size [73] $N_{\text{eff}}^{-1} := \sum_{i=1}^{N} {w_t^{(i)}}^2$, which measures the degeneracy such that if it is very low, the variance of the weights are very high; in which case, we can eliminate the aforementioned particles with very small weights and instead explore the large weighted particles by resampling [25]. The overall online prediction algorithm is outlined in Algorithm 1.

Algorithm 1 Particle Filtering based Online Learning

Require: N, number of particles
Require: N_T, resampling threshold
  for $i = 1$ to N **do**
    Sample $\boldsymbol{s}_t^{(i)} \sim p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}^{(i)})$
    $w_t^{(i)} \leftarrow p(y_t|\boldsymbol{s}_t^{(i)})\, w_{t-1}^{(i)}$
  end for
  Total weight: $Z = \sum_{i=1}^{N} w_t^{(i)}$
  Normalization: $w_t^{(i)} \leftarrow w_t^{(i)}/Z, \forall i$
  Effective size: $N_{\text{eff}} \leftarrow 1/\sum_{i=1}^{N} {w_t^{(i)}}^2$
  if $N_{\text{eff}} < N_T$ **then**
    Resample $\boldsymbol{s}_t^{(i)}, \forall i$ as per [25]
    Uniformize weights: $w_t^{(i)} \leftarrow 1/N, \forall i$
  end if
  Compute the state estimate: $\hat{\boldsymbol{s}}_t \leftarrow \sum_{i=1}^{N} w_t^{(i)} \boldsymbol{s}_t^{(i)}$
  Compute the prediction: $\hat{y}_{t+1} \leftarrow \boldsymbol{\lambda}_t^T \hat{\boldsymbol{s}}_t$

3.4 Experiments

In this section, we illustrate the performance of the proposed architecture under different scenarios. Firstly, we consider the regression performance of the model over well-known real life datasets such as bank [44], elevators [45], kinematics [46] and pumadyn [45] datasets in an online setting. We then perform simulations over the yearly, daily and hourly subsets of the M4 competition [47] dataset, aiming one-step ahead forecasting. We lastly consider a financial dataset, Alcoa stock prices [48], where we demonstrate the generic nature of our architecture by replacing the LSTM part of our model with a GRU network. In each experiment, the only preprocessing step we apply to the datasets is standardization, i.e., taking the mean off and dividing by the standard deviation. In offline settings (e.g., with the M4 dataset), we first extract the mean and the standard deviation of the features across the entire training split of the data, and scale both the training and testing data with those same statistics. In online settings (e.g., in financial datasets), we apply standardization cumulatively via storing a running mean and running standard deviation as the stream provides data. We note that we do not extract any additional features from the datasets; the LSTM network

is tasked to automatically extract the features from the raw data in a nonlinear fashion. All the experiments were conducted using a computer with i7-6700HQ processor, 2.6 GHz CPU and 12 GB RAM.

3.4.1 Real Life Datasets

In this section, we evaluate the performance of the proposed model in an online learning setting. To this end, we consider four real life datasets. The description of the datasets can be found in Section 2.3.2.

We compare our model against five different models: the Naive model, SARIMAX (without a seasonal component), multilayer perceptron (MLP), light gradient boosting machine (LightGBM) and the conventional LSTM architecture. The Naive model predicts $\hat{y}_t = y_{t-1}$ for all time steps, i.e., it carries the last measurement as its prediction for the next time step. For the SARIMAX model, we tune the number of AR and MA components using a stepwise algorithm outlined in [51]. The MLP we use in our experiments approximate a nonlinear autoregressive process with exogenous regressors (ARX), i.e., the values of the input neurons are the lagged values of the target signal as well as any side information the data provides. We use a one-layer shallow network for which we tune number of hidden neurons, learning rate and the activation function in the hidden layer (ReLU [52] is used for the output neuron). LightGBM [16] is a hard tree-based gradient boosting machine, which, with well-designed features, achieves effective sequential data prediction. We tune the number of trees, learning rate, bagging fraction, maximum number of leaves and regularization constants for LightGBM. For the LSTM model, we tune the number of layers, number of hidden units in each layer and the learning rate. For our architecture, we search for the number of particles in the filter, number of hidden units for the LSTM component and ordinary & seasonal orders for the SARIMAX model from a random subset of a grid. Among these five models, however, only the Naive model, MLP and LSTM are amenable to online learning; for the other two models we proceed as follows. For SARIMAX, we use the underlying Kalman filter to perform training;

for LightGBM, we could perform re-training over the accumulated data at each time step; however, since this would already become too much time consuming, we introduce a queue of fixed size, say 50, and when it is fully filled, we append the new data to the historical data to refit the model; the queue is then emptied, and so on.

To assess the performance of online regression, we measure time accumulated mean squared errors of models, which is computed as the cumulative sum of MSE's normalized by the data length at a given time t. In Table 3.1, we report the cumulative error at the last time step for all models. We do not apply any preprocessing to the data for the Naive, SARIMAX and LightGBM models, while applying the standardization described in Section 3.4 to other models.

The proposed model (LSTM-SX) performs better on all datasets except for Elevators over the final cumulative error metric compared to other models. For example, in the Pumadyn dataset where the number of exogenous variables is the highest, our model performs significantly better than all models except for MLP, for which the difference is still considerable given the scale. This suggests the importance of an automatic feature extractor from the raw sequential data. Furthermore, the hybrid architecture consistently achieves better scores than the pure LSTM network; this emphasizes the contribution of the SARIMAX part, which not only addresses time series specific components, e.g., linear effect of shocks, but also allows for a direct contribution of exogenous variables to the final prediction. Overall, the results allude to the effectiveness of the framework in an online setting.

Table 3.1: Cumulative errors of the models on Bank, Elevators, Kinematics and Pumadyn datasets

	Bank	Elevators	Kinematics	Pumadyn
Naive	0.0307	0.0714	0.1305	0.0218
SARIMAX	0.0183	0.0465	0.0897	0.0107
MLP	0.0165	**0.0043**	0.0766	0.0019
LightGBM	0.0203	0.0234	0.0991	0.0143
LSTM	0.0190	0.0119	0.0884	0.0094
LSTM-SX	**0.0129**	0.0098	**0.0627**	**0.0013**

Table 3.2: Mean MAPE performances of the models across M4 datasets

	Hourly	Daily	Yearly
Naive	0.50915	0.09608	0.67152
SARIMAX	0.15549	0.05425	0.20032
MLP	0.19380	0.02940	0.22447
LightGBM	0.19468	0.01602	0.38881
LSTM	0.20282	0.06448	0.18032
LSTM-SX	**0.13761**	**0.01575**	**0.14120**

3.4.2 M4 Competition Datasets

Here we use the M4 competition data as decribed in Section 2.3.3 and compare our model with five models presented in Section 3.4.1. For each dataset, we separate a validation sequence from the end of each time series as long as the desired prediction horizon. For example, a series in the hourly dataset has its last 48 samples spared for validation of the hyperparameters. The searched hyperparameters of models as well as the preprocessing setup are as described in Section 3.4.1. We use MAPE to evaluate models' performance across series.

The mean MAPE scores of the models over the datasets are seen in Table 3.2. The LSTM-SX model achieves better performance than other models in all datasets, albeit with a close runner up in the daily dataset, LightGBM. In the yearly dataset, which consists of sparse data, LSTM-SX model adapts quickly than other models, especially the LSTM network. This implies the linear SARIMAX part helps attenuate the overfitting nature of a complex network, i.e., providing a regularization effect. On the other hand, in the daily dataset, which has very long sequences, our model also outperforms the others, suggesting that the history-caring feature extractor LSTM part in the architecture is in use as well. Lastly, our model's better score in the highly seasonal hourly dataset shows that the time series specific component, SARIMAX, models the seasonality well and justifies its existence in the integrated architecture. Collectively, the proposed model achieves better on average over several M4 competition datasets of varying characteristics.

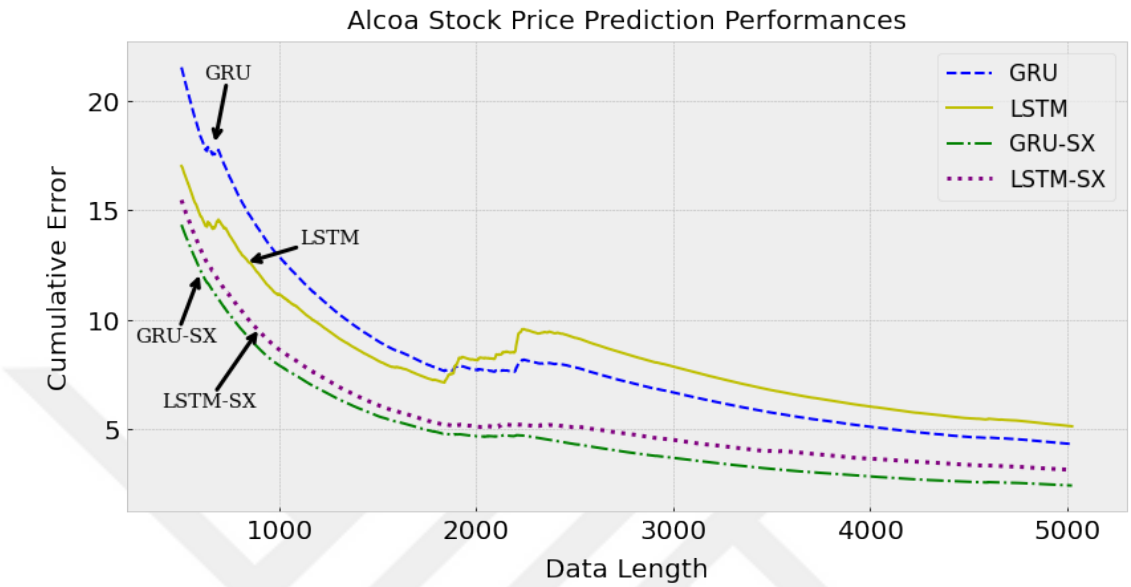

Figure 3.1: Time accumulated errors of standalone and hybrid models for the Alcoa stock price dataset for the period 2000-2020.

3.4.3 Genericness Study

Here, we show the generic nature of our architecture via substituting the LSTM network with a gated recurrent unit (GRU) network [23], which is another powerful RNN variant. A GRU cell, much like an LSTM cell, is geared with gated routing mechanisms to avoid the vanishing gradient problem [54]. Its transition equations analogous to Eq. (3.5) are given as

$$\begin{aligned} \boldsymbol{z}_t &= \sigma(\boldsymbol{W}_z \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_z) \\ \boldsymbol{q}_t &= \sigma(\boldsymbol{W}_q \{\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_r) \\ \tilde{\boldsymbol{h}}_t &= \tanh(\boldsymbol{W}_h \{\boldsymbol{q}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\} + \boldsymbol{b}_h) \\ \boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t, \end{aligned} \tag{3.25}$$

where $\boldsymbol{h}_t$ is the hidden state vector, $\boldsymbol{x}_t$ is the input, $\boldsymbol{z}_t$ and $\boldsymbol{q}_t$ are the update and reset gate vectors, respectively, and $\tilde{\boldsymbol{h}}_t$ is the candidate state vector, whose corresponding stacked weight matrices are $\boldsymbol{W}_z, \boldsymbol{W}_q$ and $\boldsymbol{W}_h$. The corresponding biases are represented by $\boldsymbol{b}_z, \boldsymbol{b}_r$ and $\boldsymbol{b}_h$. To implement a GRU-SX, we directly replace the LSTM equations with GRU equations; this shows the flexibility of the framework.

We consider four models in this section: LSTM, GRU, LSTM-SX and GRU-SX. We assess the performances of them against a financial dataset in an online setting. We use the Alcoa stock price dataset [48], which is comprised of the daily stock price values between the years 2000 and 2020. As side information, we use the lagged prices with a window size of 5. We only apply standardization to the price values in a cumulative manner. We hold out 10% of the data from the beginning and use it for the hyperparameter validation. Tuned hyperparameters are the same as those in M4 datasets. Fig. 3.1 illustrates the performance of the models as the data length varies. We observe that proposed hybrid models consistently achieve lower accumulated errors than the purely recurrent models, showing the efficacy of the framework. GRU network begins with a rather big error margin but closes the gap after 2000 data points and performs better than LSTM then on. The hybrid models perform close to each other yet the GRU-SX model performs consistently better. The standalone LSTM and GRU models are not only rather slower to react to the sudden change in the stock prices around 2,200 data points, which corresponds to the global financial crisis in the late 2008, but also face a noticable increase on cumulative error unlike the hybrid models.

Chapter 4

Conclusion

We studied nonlinear prediction/regression in an online setting and introduced two hybrid architectures composed of an LSTM and a soft GBDT, and an LSTM and SARIMAX. In both of the end-to-end architectures, the enhanced recurrent neural network acts as a feature extractor from the raw sequential data. In the first architecture, soft GBDT employs an effective supervised regressor role while the SARIMAX model in the second architecture helps address the time series specific parts inherent to data in a robust manner, e.g., seasonality and direct effect of exogenous variables. For the first framework, the joint optimization is achieved thanks to the *soft* decision trees as the weak learners in the boosting chain; for the second one, the novel state space formulation introduced for the SARIMAX model not only leads to a single-pass estimation of its parameters but also allows for the concatenation of the state spaces, resulting in joint optimization. Stochastic gradient descent and particle filtering are used for training the integrated models, respectively, for both of which we also provide the update equations. We emphasize that the proposed frameworks are generic so that one can use other deep learning architectures, e.g., GRUs, and differentiable machine learning algorithms as well as time series models that have state space representations in place of the specific models presented. We achieve consistent and significant performance gains in our experiments over conventional methods as well as the disjoint counterparts on various well-known real life datasets while

demonstrating the generic nature of the frameworks. We also provide the source code of the models for reproducibility.

Bibliography

[1] D. McQueen, P. Hyland, and S. Watson, "Monte Carlo simulation of residential electricity demand for forecasting maximum demand on distribution networks," *IEEE Transactions on Power Systems*, vol. 19, no. 3, pp. 1685–1689, 2004.

[2] N. Khodor, G. Carrault, D. Matelot, H. Amoud, M. Khalil, N. Thillaye du Boullay, F. Carre, and A. Hernández, "Early syncope detection during head up tilt test by analyzing interactions between cardio-vascular signals," *Digital Signal Processing*, vol. 49, pp. 86–94, 2016.

[3] M. Lagha, M. Tikhemirine, S. Bergheul, T. Rezoug, and M. Bettayeb, "Denoised estimation of the weather Doppler spectrum by the wavelet method," *Digital Signal Processing*, vol. 23, no. 1, pp. 322–328, 2013.

[4] C. Soguero-Ruiz, F. J. Gimeno-Blanes, I. Mora-Jiménez, M. del Pilar Martínez-Ruiz, and J. L. Rojo-Álvarez, "Statistical nonlinear analysis for reliable promotion decision-making," *Digital Signal Processing*, vol. 33, pp. 156–168, 2014.

[5] L. Liu, L. Shao, and P. Rockett, "Human action recognition based on boosted feature selection and naive Bayes nearest-neighbor classification," *Signal Processing*, vol. 93, no. 6, pp. 1521–1530, 2013. Special issue on Machine Learning in Intelligent Image Processing.

[6] X. An, C. Hu, G. Liu, and H. Lin, "Distributed online gradient boosting on data stream over multi-agent networks," *Signal Processing*, vol. 189, p. 108253, 2021.

[7] F. Petropoulos et al., "Forecasting: theory and practice," *International Journal of Forecasting*, Jan 2022.

[8] A. Singer, G. Wornell, and A. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital Signal Processing: A Review Journal*, vol. 4, pp. 207–221, Oct. 1994.

[9] R. Tanno, K. Arulkumaran, D. Alexander, A. Criminisi, and A. Nori, "Adaptive neural trees," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6166–6175, PMLR, 09–15 Jun 2019.

[10] A. C. Tsoi, *Gradient based learning methods*, pp. 27–62. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[11] H. Xu, R. Ma, L. Yan, and Z. Ma, "Two-stage prediction of machinery fault trend based on deep learning for time series analysis," *Digital Signal Processing*, vol. 117, p. 103150, 2021.

[12] I. Crignon, B. Dubuisson, and G. Le Guillou, "The ternary decision tree: A multistage classifier with reject," *Signal Processing*, vol. 5, no. 5, pp. 433–443, 1983.

[13] B. C. Civek, I. Delibalta, and S. S. Kozat, "Highly efficient hierarchical online nonlinear regression using second order methods," *Signal Processing*, vol. 137, pp. 22–32, 2017.

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189 – 1232, 2001.

[15] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[17] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The M5 competition: Background, organization, and implementation," *International Journal of Forecasting*, 2021.

[18] H. Liu and Z. Long, "An improved deep learning model for predicting stock market price time series," *Digital Signal Processing*, vol. 102, p. 102741, 2020.

[19] F. Wen and Z. Wang, "Distributed Kalman filtering for robust state estimation over wireless sensor networks under malicious cyber attacks," *Digital Signal Processing*, vol. 78, pp. 92–97, 2018.

[20] R. T. Clemen, "Combining forecasts: A review and annotated bibliography," *International Journal of Forecasting*, vol. 5, no. 4, pp. 559–583, 1989.

[21] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, (Berlin, Heidelberg), pp. 1–15, Springer Berlin Heidelberg, 2000.

[22] K. Greff, R. Srivastava, J. Koutník, B. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, 03 2015.

[23] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[24] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018.

[25] 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.

[26] E. S. Gardner Jr., "Exponential smoothing: The state of the art," *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[27] J. Feng, Y. Xu, Y. Jiang, and Z. Zhou, "Soft gradient boosting machine," *CoRR*, vol. abs/2006.04059, 2020.

[28] O. Irsoy, O. T. Yildiz, and E. Alpaydin, "Soft decision trees," in *International Conference on Pattern Recognition*, 2012.

[29] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," *CoRR*, vol. abs/1711.09784, 2017.

[30] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò, "Deep neural decision forests," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1467–1475, 2015.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[32] G. Biau, E. Scornet, and J. Welbl, "Neural random forests," *CoRR*, vol. abs/1604.07143, 2016.

[33] H. Hazimeh, N. Ponomareva, P. Mol, Z. Tan, and R. Mazumder, "The tree ensemble layer: Differentiability meets conditional computation," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 4138–4148, PMLR, 2020.

[34] H. Chen, S. M. Lundberg, and S. Lee, "Hybrid gradient boosting trees and neural networks for forecasting operating room data," *CoRR*, vol. abs/1801.07384, 2018.

[35] Y. Huang, Y. He, R. Lu, X. Li, and X. Yang, "Thermal infrared object tracking via unsupervised deep correlation filters," *Digital Signal Processing*, vol. 123, p. 103432, 2022.

[36] Y. Huang, Y. He, R. Lu, X. Li, and X. Yang, "Thermal infrared object tracking via unsupervised deep correlation filters," *Digital Signal Processing*, vol. 123, p. 103432, 2022.

[37] X. Liao, Y. Yu, B. Li, Z. Li, and Z. Qin, "A new payload partition strategy in color image steganography," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 3, pp. 685–696, 2020.

[38] X. Liao, J. Yin, M. Chen, and Z. Qin, "Adaptive payload distribution in multiple images steganography based on image texture features," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 897–911, 2022.

[39] X. Liao, K. Li, X. Zhu, and K. J. R. Liu, "Robust detection of image operator chain with two-stream convolutional neural network," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 955–968, 2020.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[41] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural computation*, vol. 12, pp. 2451–71, 10 2000.

[42] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[43] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, (San Francisco, CA, USA), p. 148–156, Morgan Kaufmann Publishers Inc., 1996.

[44] L. Torgo, "Regression data sets."

[45] J. Alcala-Fdez, A. Fernández, J. Luengo, J. Derrac, S. Garc'ia, L. Sanchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, pp. 255–287, 01 2010.

[46] C. E. Rasmussen, R. M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, "Delve data sets."

[47] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The M4 competition: 100,000 time series and 61 forecasting methods," *International Journal of Forecasting*, vol. 36, no. 1, pp. 54–74, 2020. M4 Competition.

[48] Alcoa Inc., "Common stock."

[49] E. W. Frees, *Regression Modeling with Actuarial and Financial Applications*. International Series on Actuarial Science, Cambridge University Press, 2009.

[50] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, *Efficient BackProp*, pp. 9–50. Springer Berlin Heidelberg, 1998.

[51] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for r," *Journal of Statistical Software*, vol. 27, no. 3, p. 1–22, 2008.

[52] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (Madison, WI, USA), p. 807–814, Omnipress, 2010.

[53] M. Du, "Improving LSTM neural networks for better short-term wind power predictions," *CoRR*, vol. abs/1907.00489, 2019.

[54] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, p. III–1310–III–1318, JMLR.org, 2013.

[55] T. Hill, M. O'Connor, and W. Remus, "Neural network models for time series forecasts," *Management Science*, vol. 42, no. 7, pp. 1082–1092, 1996.

[56] I. Kaastra and M. Boyd, "Designing a neural network for forecasting financial and economic time series," *Neurocomputing*, vol. 10, no. 3, pp. 215–236, 1996. Financial Applications, Part II.

[57] C. Fan, J. Wang, W. Gang, and S. Li, "Assessment of deep recurrent neural network-based strategies for short-term building energy predictions," *Applied Energy*, vol. 236, pp. 700–710, 2019.

[58] M. N. Fekri, H. Patel, K. Grolinger, and V. Sharma, "Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network," *Applied Energy*, vol. 282, p. 116177, 2021.

[59] A. Sagheer and M. Kotb, "Time series forecasting of petroleum production using deep LSTM recurrent networks," *Neurocomputing*, vol. 323, pp. 203–213, 2019.

[60] A. Kisvari, Z. Lin, and X. Liu, "Wind power forecasting – a data-driven method along with gated recurrent neural network," *Renewable Energy*, vol. 163, pp. 1895–1909, 2021.

[61] M. E. Aydin and S. S. Kozat, "A hybrid framework for sequential data prediction with end-to-end optimization," *Digital Signal Processing*, vol. 129, p. 103687, 2022.

[62] H. Nazaripouya, B. Wang, Y. Wang, P. Chu, H. R. Pota, and R. Gadh, "Univariate time series prediction of solar power using a hybrid wavelet-ARMA-NARX prediction method," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, pp. 1–5, 2016.

[63] L. Deng and J. Chen, "Sequence classification using the high-level features extracted from deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6844–6848, 2014.

[64] Y. Jeong and S. Lee, "Recurrent neural network-adapted nonlinear ARMA-GARCH model with application to S&P 500 index data," *Journal of the Korean Data And Information Science Society*, vol. 30, pp. 1187–1195, 09 2019.

[65] E. Box George, M. Jenkins Gwilym, C. Reinsel Gregory, and M. Ljung Greta, "Time series analysis: forecasting and control," *San Francisco: Holden Bay*, 1976.

[66] N. S. Arunraj, D. Ahrens, and M. Fernandes, "Application of SARIMAX model to forecast daily sales in food retail industry," *International Journal of Operations Research and Information Systems (IJORIS)*, vol. 7, no. 2, pp. 1–21, 2016.

[67] Y. Grenier, "Time-dependent ARMA modeling of nonstationary signals," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 31, no. 4, pp. 899–911, 1983.

[68] E. J. Hannan and J. Rissanen, "Recursive estimation of mixed autoregressive-moving average order," *Biometrika*, vol. 69, no. 1, pp. 81–94, 1982.

[69] J. Durbin and S. J. Koopman, *Time series analysis by state space methods*, vol. 38. OUP Oxford, 2012.

[70] J. D. Hamilton, "State-space models," *Handbook of econometrics*, vol. 4, pp. 3039–3080, 1994.

[71] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, p. 35, 1960.

[72] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, pp. 197 – 208, 2000.

[73] J. Liu, "Metropolized independent sampling with comparisons to rejection sampling and importance sampling," *Statistics and Computing*, vol. 6, 08 2000.