Deep Learning for Time Series Prediction & Decision Making Over Time

Bryan Lim
St Cross College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2020

Acknowledgements

Firstly, I would like to express my immense gratitude to my supervisors Dr. Stefan Zohren and Prof. Stephen Roberts for their guidance throughout my DPhil. Their support and encouragement has been invaluable in broadening my research horizons and has helped me deepen my expertise in the field. I would like to thank everyone I have worked with at the Oxford-Man Institute – both for the provision of funding support and for being an integral part of my experience at Oxford. I also thank Xiaowen Dong, Martin Schmalz and Michael Osborne for their feedback during my confirmation and transfer of status, which has helped improved the quality of this thesis.

I am grateful to all the research collaborators with whom I have had the pleasure of working during my DPhil. In particular, I would like to thank Sercan Arik and Nicolas Loeff at Google Cloud AI for all the thought discussions on time series forecasting, which sparked many research ideas and lead to the development of the TFT. To Anthony Ledford and Thomas Flury at Man AHL, thank you for your insights into systematic trading which were very helpful in the presentation of the DMN. I also thank Prof. Mihaela van der Schaar and her group, especially Ahmed Alaa, Jinsung Yoon, James Jordon and Alexis Bellot, for their patience in the early phases of my studies – their advice and assistance was vital in helping me shape my approach to research.

Above all, I would like to thank my parents, Goh Soo Cheng and Michael Lim, and my siblings, Shawn Lim and Michelle Lim, for their infinite support through the ups and downs of this academic adventure.

Abstract

In this thesis, we develop a collection of state-of-the-art deep learning models for time series forecasting. Primarily focusing on a closer alignment with traditional methods in time series modelling, we adopt three main directions of research – 1) novel architectures, 2) hybrid models, and 3) feature extraction. Firstly, we propose two new architectures for general one-step-ahead and multi-horizon forecasting. With the Recurrent Neural Filter (RNF), we take a closer look at the relationship between recurrent neural networks and Bayesian filtering, so as to improve representation learning for one-step-ahead forecasts. For multi-horizon forecasting, we propose the Temporal Fusion Transformer (TFT) – an attention-based model designed to accommodate the full range of inputs present in common problem scenarios. Secondly, we investigate the use of hybrid models to enhance traditional quantitative models with deep learning components – using domain-specific knowledge can be used to guide neural network training. Through applications in finance (Deep Momentum Networks) and medicine (Disease-Atlas), we demonstrate that hybrid models can effectively improve forecasting performance over pure methods in either category. Finally, we explore the feature learning capabilities of deep neural networks to devise features for general forecasting models. Considering an application in systemic risk management, we devise the Autoencoder Reconstruction Ratio (ARR) – an indicator to measure the degree of co-movement between asset returns. When fed as an input into a variety of models, we show that the ARR can help to improve short-term predictions of various risk metrics.

On top of improvements in forecasting performance, we also investigate extensions to enable decision support using deep neural networks, by helping users to better understand their data. With Recurrent Marginal Structural Networks (RMSNs), we introduce a general framework to train deep neural networks to learn causal effects over time, using ideas from marginal structural modelling in epidemiology. In addition, we also propose three practical interpretability use-cases for the TFT, demonstrating how attention weights can be analysed to provide insights into temporal dynamics.

Contents

Chapter 1

Introduction

1.1 Motivations

Time series analysis is an integral part of many disciplines – providing researchers with the analytical tools required to study temporal dynamics. Ranging from simple autoregressive models regularly used in econometrics [4], to more sophisticated particle filtering from signal processing [2], numerous methods have been develop to make predictions of and reason about time-varying behavior. In addition to achieving high degrees of predictive accuracy, these traditional methods also provide a degree of insight into the relationships of the underlying models, given the generative or parametric nature of the models used. For instance, analyzing p-values of coefficients in vector autoregressive models allows researchers to identify significant inputs for the prediction problem, and state space models allow for the inference of otherwise unobservable latent states.

With the rapid growth in data availability over time, machine learning methods have increasingly been used for time series prediction tasks in applications such as retail demand forecasting [6], healthcare [15], and finance [8]. Favoured for their ability to learn relationships in a purely data driven manner, time series forecasting with deep learning has risen to prominence in recent times – fueled by phenomenal successes in sequence modelling [28, 29] and reinforcement learning [27] tasks in other domains. However, new architectures have largely developed independently from the field of time series analysis, focusing on alleviating specific limitations in the context of *sequence* prediction – e.g. difficulties in learning long-term dependencies in Recurrent Neural Networks (RNNs) [11]. As such, improving the alignment of neural network architectures with traditional time series models could help to improve forecasting accuracy – potentially by taking into account nuances in the dataset, or by improving representation learning [1].

Furthermore, while researchers primarily concentrate on prediction accuracy when developing models, practitioners rely on model forecasts more to inform their decision making and optimise their actions over time. In retail applications for instance, stores can use sales forecasting models to determine product demand, which allows them to optimise inventory management [24]. As such, providing components which help users to understand the temporal relationships within their time series datasets can also be beneficial for decision making. Moreover, with more deep learning systems featuring in mission critical-applications, such as in patient diagnosis [15] and portfolio management [8], understanding the key drivers of a model's prediction can improve the users trust in its forecast. Given the black-box nature of deep neural networks, this motivates the need for the development of components which provides insights into the relationships learnt by the model.

1.2 Contributions & Outline of Report

1.2.1 Research Directions

This thesis focuses on the development of state-of-the-art time deep learning models for time series forecasting, along with extensions to use deep neural networks to facilitate decision support. We divide our proposed improvements into the two main research areas below – with a full outline of research trends in the literature review of Chapter 2.

Time Series Forecasting: We adopt three main approaches to utilise deep learning in high-performance time series forecasting applications. For a start, we develop novel deep learning architectures to improve representation learning in one-step-ahead and multi-horizon time series forecasting. Next, we demonstrate the use of hybrid models in a variety of applications, using deep learning components to enhance well-studied quantitative models for a given domain. Lastly, we assess the use of deep neural networks as a feature extraction mechanism for high-frequency data – using autoencoders to extract useful features to feed into other forecasting models.

Facilitating Decision Making Over Time: In addition to improvements in forecasting, we also explore extensions to improve the decision support capabilities of deep neural networks in several research directions. Firstly, we investigate the use of deep neural networks to learn time-varying causal effects, training them with biased observational data typically collected in many industrial settings. With causal deep

learning models, decision makers can then perform scenario analyses through counterfactual simulations, allowing them to optimise their decisions in scenarios that are historically uncommon or not present in the data. Secondly, we examine methods that incorporate explainability into high-performance deep learning architectures, which allow users to better understand general relationships present within their datasets. Finally, while we do not explicitly propose new methods for probabilistic modelling, we design most of our forecasting models to incorporate measures of uncertainty as well. These uncertainty estimates allow users to determine how confident a model is in its own forecast, so as to enhance the user's trust in a model's outputs.

1.2.2 Thesis Contributions & Outline

According to the research directions outlined in the previous section, we group methods into chapters based on their main area of contribution:

Novel Architectures: In Chapter 3, we propose two new state-of-the-art deep learning architectures for time series forecasting. With Recurrent Neural Filters (RNF) [18], we explore the relationships between recurrent neural networks (RNNs) and Bayesian filtering – identifying that RNNs can be viewed as a simultaneous approximation of both state transition and update steps. As such, we propose a novel RNN architecture that closely aligns with the Bayesian filtering steps, using separate encoders to capture distinct representations for each step. From a network calibration standpoint, we improve representation learning by training the RNF in a multitask fashion – with each encoder to be trained directly using a common emissions decoder. In addition, our proposed skip training approach acts as a form of input dropout, and improves generalisation by allowing encoders to operate independently from each other. From a filtering point of view, the RNF encoders can be separated at run-time based the availability of inputs, allowing them to be used in forecasting with missing data or in an autoregressive fashion for multi-horizon prediction. Considering applications in finance and electricity load forecasting, we demonstrate the performance improvements over other recurrent models for both one-step-ahead and multi-horizon forecasts, while providing comparable uncertainty estimates.

Despite their simplicity, autoregressive models for multi-horizon prediction typically assume that inputs are homogeneously available across time. However, many practical applications of multi-horizon forecasting typically contain a complex mix of inputs – including static covariates, known future inputs and other time series

that can only be observed in the past. To account for the full range of inputs available, we propose a new multi-horizon forecasting model we call the Temporal Fusion Transformer (TFT) [14], which combines state-of-the-art performance with inherent interpretability. To capture temporal representations at different scales, the TFT uses a sequence-to-sequence layer for local processing and interpretable attention blocks to learn long-term dependencies. On top of temporal self-attention layers that attend to important time steps in the lookback window, the TFT also facilitates interpretability by using variable selection networks to select relevant inputs for the prediction task. Using a range of real-world datasets, we demonstrate significant performance improvements over other multi-horizon forecasting models, and present three practical interpretability use-cases for the TFT.

Hybrid Models: While the flexibility of deep neural networks allows them to learn complex relationships in a data-driven fashion, they also can face issues with overfitting in noisy time series datasets [19] – particularly when datasets are small [22]. To alleviate this, a recent trend in deep learning research has emerged in the development of hybrid models, which parameterise well-studied quantitative models for a given domain using neural network components. Hybrid models allow for model builders to inject prior knowledge into neural network designs, guiding network calibration by restricting the form of the output learnt by the model. In Chapter 4, we introduce two new hybrid models which customised for domain-specific applications. With Deep Momentum Networks (DMNs) [16], we consider an application in finance – adopting the use of hybrid models for systematic trading. DMNs extend common trend following signals using deep learning, using neural network components to generate trading rules which are combined with the volatility scaling framework of time series momentum. We also consider a medical application with the Disease-Atlas [15], which utilises a hybrid model for co-morbidity management in patients with Cystic Fibrosis. The Disease-Atlas extends joint models for longitudinal and time-to-event data commonly used in biostatistics – using a shared network architecture to learn associations between longitudinal variables and generate parameters for predictive distributions of each variable. This allows medical professionals to jointly forecast multiple clinical outcomes, including the survival and disease onset probabilities along with expected biomarker values. For both DMNs and Disease-Atlas, we demonstrate performance improvements over pure quantitative or deep learning benchmarks – highlighting the benefits of the hybrid modelling approach.

Causal Inference Over Time: In line with new methods to facilitate decision support, Chapter 5 investigates the use of deep neural networks to learn time-dependent causal effects. Motivated by increasing prevalence of electronic health records in hospitals, we focus on the problem of forecasting time-dependent treatment effects in medical settings – although similar methods can be used in other domains. The key challenge with observational data is the presence of time-dependent confounders, which can introduce bias when no adjustments are made to neural network training. To account for time-dependent counfounding effects, we developed a class of models known as Recurrent Marginal Structural Networks (RMSN) [13], which extend marginal structural models (MSMs) in epidemiology for neural network training. RMSNs adopt an inverse probability of treatment weighting (IPTW) approach to adjust for time-dependent confounding – using a set of networks to estimate probabilities of treatment application and censoring at each time step. Propensity weights are computed based on these probabilities, which then adjust the loss function of a separate sequence-to-sequence model that predicts treatment effects. Using a clinical realistic simulation model for non-small cell lung cancer, we demonstrate that RMSNs improves the performance of both standard MSMs and other machine learning baselines in learning unbiased treatment effects.

Feature Extraction from High Frequency Data: Apart from being used directly for prediction tasks, the representation learning capabilities of deep neural networks have also been re-purposed for high-quality feature extraction across a variety of tasks [7, 30, 23]. Considering a financial use-case in systemic risk prediction, we explore the use of autoencoders to extract useful features for long-term predictions in Chapter 6. In the spirit of traditional techniques that compute real-time estimates from high-frequency data – such as HEAVY models [25, 21, 26] which estimate volatility using intra-day returns – we present the Autoencoder Reconstruction Ratio (ARR) [17] as an early-warning indicator of changes in asset correlations. Using deep sparse denoising autoencoder for dimensionality reduction, the ARR is computed by aggregating the reconstruction errors of high-frequency returns over various horizons. Through tests on intra-day index returns, we demonstrate that autoencoders do provide a better model for high-frequency data – reducing reconstruction errors when compared to simpler methods based on principal component analysis. Furthermore, we show that ARR can enhance the accuracy of short-term predictions of volatility and market crashes when included as an additional input into forecasting models.

1.2.2.1 Presentation Format

This thesis is presented in the *integrated thesis format*[1], with the main chapters consisting of first-author papers that have been accepted or are under consideration for publication. This includes the literature review, which has been submitted as a survey paper. To ensure compliance with any copyright restrictions, the papers have been reproduced in the original format used in their pre-print versions.

1.3 Publications

At the time of writing, the papers presented in this thesis have been accepted for publication at the venues below:

Ch. 2: Literature Review

- **B. Lim**, S. Zohren. *Time Series Forecasting With Deep Learning: A Survey.* Philosophical Transactions of the Royal Society of London A, 2021. Weblink: `https://arxiv.org/abs/2004.13408`.

Ch. 3: Novel Architectures for Time Series Forecasting

- **B. Lim**, S. O. Arik, N. Loeff, T. Pfister. *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting.* Submitted, 2020. Weblink: `https://arxiv.org/abs/1912.09363`.

- **B. Lim**, S. Zohren, S. Roberts. *Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction.* Proceedings of the International Joint Conference On Neural Networks (IJCNN), 2020. Weblink: `https://arxiv.org/abs/1901.08096`.

Ch. 4: Incorporating Domain Knowledge With Hybrid Models

- **B. Lim**, S. Zohren, S. Roberts. *Enhancing Time Series Momentum Strategies Using Deep Neural Networks.* Journal of Financial Data Science (JFDS), 2019. Weblink: `https://arxiv.org/abs/1904.04912`.

[1] `https://www.mpls.ox.ac.uk/graduate-school/information-for-postgraduate-research-students/submitting-your-thesis`

- **B. Lim**, M. van der Schaar. *Disease-Atlas: Navigating Disease Trajectories with Deep Learning*. Proceedings of the Machine Learning for Healthcare Conference (MLHC), 2018. Weblink: `https://arxiv.org/abs/1803.10254`.

The Disease-Atlas has also been presented at the following workshops:

- **B. Lim**, M. van der Schaar. *Disease-Atlas: Navigating Disease Trajectories with Deep Learning*. IJCAI International Workshop on Biomedical informatics with Optimization and Machine learning (IJCAI-BOOM), 2018. [**Best Paper Award**]

- **B. Lim**, M. van der Schaar. *Forecasting Disease Trajectories in Alzheimer's Disease Using Deep Learning*. KDD Workshop on Machine Learning for Medicine and Healthcare, 2018.

- **B. Lim**, T. Daniels, A. Floto, M. van der Schaar. *Forecasting Clinical Trajectories in Cystic Fibrosis using Deep Learning*. North American Cystic Fibrosis Conference, 2018.

Ch. 5: Learning Time-Dependent Causal Effects

- **B. Lim**, A. Alaa, M. van der Schaar. *Forecasting Treatment Responses Over Time Using Marginal Structural Models*. Advances in Neural Information Processing Systems (NeurIPS), 2018. Weblink: `http://papers.nips.cc/paper/7977-forecasting-treatment-responses-over-time-using-recurrent-marginal-structural-networks`.

Ch. 6: Extracting Useful Features From High Frequency Data

- **B. Lim**, S. Zohren, S. Roberts. *Detecting Changes in Asset Co-Movement Using the Autoencoder Reconstruction Ratio*. Risk, 2020. Weblink: `https://arxiv.org/abs/2002.02008`.

Chapter 2

Literature Review

Publications Included

- **B. Lim**, S. Zohren. *Time Series Forecasting With Deep Learning: A Survey.* Philosophical Transactions of the Royal Society A, 2021. Weblink: `https://arxiv.org/abs/2004.13408`.

PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
Deep learning, time series modelling

Keywords:
Deep neural networks, time series forecasting, uncertainty estimation, hybrid models, interpretability, counterfactual prediction

Author for correspondence:
Bryan Lim
e-mail: blim@robots.ox.ac.uk

THE ROYAL SOCIETY PUBLISHING

Time Series Forecasting With Deep Learning: A Survey

Bryan Lim[1] and Stefan Zohren[1]

[1]Department of Engineering Science, University of Oxford, Oxford, UK

Numerous deep learning architectures have been developed to accommodate the diversity of time series datasets across different domains. In this article, we survey common encoder and decoder designs used in both one-step-ahead and multi-horizon time series forecasting – describing how temporal information is incorporated into predictions by each model. Next, we highlight recent developments in hybrid deep learning models, which combine well-studied statistical models with neural network components to improve pure methods in either category. Lastly, we outline some ways in which deep learning can also facilitate decision support with time series data.

1. Introduction

Time series modelling has historically been a key area of academic research – forming an integral part of applications in topics such as climate modelling [1], biological sciences [2] and medicine [3], as well as commercial decision making in retail [4] and finance [5] to name a few. While traditional methods have focused on parametric models informed by domain expertise – such as autoregressive (AR) [6], exponential smoothing [7, 8] or structural time series models [9] – modern machine learning methods provide a means to learn temporal dynamics in a purely data-driven manner [10]. With the increasing data availability and computing power in recent times, machine learning has become a vital part of the next generation of time series forecasting models.

Deep learning in particular has gained popularity in recent times, inspired by notable achievements in image classification [11], natural language processing [12] and reinforcement learning [13]. By incorporating bespoke architectural assumptions – or inductive biases [14] – that reflect the nuances of underlying datasets, deep neural networks are able to learn complex data representations [15], which alleviates the need for manual feature engineering and model design. The availability of open-source backpropagation frameworks [16, 17] has also simplified the network training, allowing for the customisation for network components and loss functions.

Given the diversity of time-series problems across various domains, numerous neural network design choices have emerged. In this article, we summarise the common approaches to time series prediction using deep neural networks. Firstly, we describe the state-of-the-art techniques available for common forecasting problems – such as multi-horizon forecasting and uncertainty estimation. Secondly, we analyse the emergence of a new trend in hybrid models, which combine both domain-specific quantitative models with deep learning components to improve forecasting performance. Next, we outline two key approaches in which neural networks can be used to facilitate decision support, specifically through methods in interpretability and counterfactual prediction. Finally, we conclude with some promising future research directions in deep learning for time series prediction – specifically in the form of continuous-time and hierarchical models.

While we endeavour to provide a comprehensive overview of modern methods in deep learning, we note that our survey is by no means all-encompassing. Indeed, a rich body of literature exists for automated approaches to time series forecasting - including automatic parametric model selection [18], and traditional machine learning methods such as kernel regression [19] and support vector regression [20]. In addition, Gaussian processes [21] have been extensively used for time series prediction – with recent extensions including deep Gaussian processes [22], and parallels in deep learning via neural processes [23]. Furthermore, older models of neural networks have been used historically in time series applications, as seen in [24] and [25].

2. Deep Learning Architectures for Time Series Forecasting

Time series forecasting models predict future values of a target $y_{i,t}$ for a given entity i at time t. Each entity represents a logical grouping of temporal information – such as measurements from individual weather stations in climatology, or vital signs from different patients in medicine – and can be observed at the same time. In the simplest case, one-step-ahead forecasting models take the form:

$$\hat{y}_{i,t+1} = f(y_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t}, \boldsymbol{s}_i), \tag{2.1}$$

where $\hat{y}_{i,t+1}$ is the model forecast, $y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$, $\boldsymbol{x}_{i,t-k:t} = \{\boldsymbol{x}_{i,t-k}, \ldots, \boldsymbol{x}_{i,t}\}$ are observations of the target and exogenous inputs respectively over a look-back window k, s_i is static metadata associated with the entity (e.g. sensor location), and $f(.)$ is the prediction function learnt by the model. While we focus on univariate forecasting in this survey (i.e. 1-D targets), we note that the same components can be extended to multivariate models without loss of generality [26, 27, 28, 29, 30]. For notational simplicity, we omit the entity index i in subsequent sections unless explicitly required.

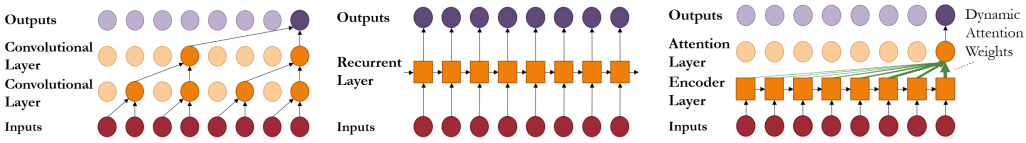
(a) Basic Building Blocks

Deep neural networks learn predictive relationships by using a series of non-linear layers to construct intermediate feature representations [15]. In time series settings, this can be viewed as encoding relevant historical information into a latent variable $\boldsymbol{z}_t$, with the final forecast produced using $\boldsymbol{z}_t$ alone:

$$f(y_{t-k:t}, \boldsymbol{x}_{t-k:t}, \boldsymbol{s}) = g_{\text{dec}}(\boldsymbol{z}_t), \tag{2.2}$$

$$\boldsymbol{z}_t = g_{\text{enc}}(y_{t-k:t}, \boldsymbol{x}_{t-k:t}, \boldsymbol{s}), \tag{2.3}$$

where $g_{\text{enc}}(.)$, $g_{\text{dec}}(.)$ are encoder and decoder functions respectively, and recalling that that subscript i from Equation (2.1) been removed to simplify notation (e.g. $y_{i,t}$ replaced by y_t). These encoders and decoders hence form the basic building blocks of deep learning architectures, with the choice of network determining the types of relationships that can be learnt by our model. In this section, we examine modern design choices for encoders, as overviewed in Figure 1, and their relationship to traditional temporal models. In addition, we explore common network outputs and loss functions used in time series forecasting applications.



(a) CNN Model. (b) RNN Model. (c) Attention-based Model.

Figure 1: Incorporating temporal information using different encoder architectures.

(i) Convolutional Neural Networks

Traditionally designed for image datasets, convolutional neural networks (CNNs) extract local relationships that are invariant across spatial dimensions [11, 31]. To adapt CNNs to time series datasets, researchers utilise multiple layers of causal convolutions [32, 33, 34] – i.e. convolutional filters designed to ensure only past information is used for forecasting. For an intermediate feature at hidden layer l, each causal convolutional filter takes the form below:

$$\boldsymbol{h}_t^{l+1} = A\Big((\boldsymbol{W} * \boldsymbol{h})\,(l,t) \Big), \tag{2.4}$$

$$(\boldsymbol{W} * \boldsymbol{h})\,(l,t) = \sum_{\tau=0}^{k} \boldsymbol{W}(l,\tau)\boldsymbol{h}_{t-\tau}^{l}, \tag{2.5}$$

where $\boldsymbol{h}_t^l \in \mathbb{R}^{\mathcal{H}_{in}}$ is an intermediate state at layer l at time t, $*$ is the convolution operator, $\boldsymbol{W}(l,\tau) \in \mathbb{R}^{\mathcal{H}_{out}\times\mathcal{H}_{in}}$ is a fixed filter weight at layer l, and $A(.)$ is an activation function, such as a sigmoid function, representing any architecture-specific non-linear processing. For CNNs that use a total of L convolutional layers, we note that the encoder output is then $\boldsymbol{z}_t = \boldsymbol{h}_t^L$.

Considering the 1-D case, we can see that Equation (2.5) bears a strong resemblance to finite impulse response (FIR) filters in digital signal processing [35]. This leads to two key implications for temporal relationships learnt by CNNs. Firstly, in line with the spatial invariance assumptions for standard CNNs, temporal CNNs assume that relationships are time-invariant – using the same set of filter weights at each time step and across all time. In addition, CNNs are only able to use inputs within its defined lookback window, or receptive field, to make forecasts. As such, the receptive field size k needs to be tuned carefully to ensure that the model can make use of all relevant historical information. It is worth noting that a single causal CNN layer with a linear activation function is equivalent to an auto-regressive (AR) model.

Dilated Convolutions Using standard convolutional layers can be computationally challenging where long-term dependencies are significant, as the number of parameters scales directly with the size of the receptive field. To alleviate this, modern architectures frequently make use of dilated covolutional layers [32, 33], which extend Equation (2.5) as below:

$$(\boldsymbol{W} * \boldsymbol{h})\,(l,t,d_l) = \sum_{\tau=0}^{\lfloor k/d_l \rfloor} \boldsymbol{W}(l,\tau)\boldsymbol{h}_{t-d_l\tau}^{l}, \tag{2.6}$$

where $\lfloor . \rfloor$ is the floor operator and d_l is a layer-specific dilation rate. Dilated convolutions can hence be interpreted as convolutions of a down-sampled version of the lower layer features – reducing resolution to incorporate information from the distant past. As such, by increasing the dilation rate with each layer, dilated convolutions can gradually aggregate information at different time blocks, allowing for more history to be used in an efficient manner. With the WaveNet architecture of [32] for instance, dilation rates are increased in powers of 2 with adjacent time blocks aggregated in each layer – allowing for 2^l time steps to be used at layer l as shown in Figure 1a.

(ii) Recurrent Neural Networks

Recurrent neural networks (RNNs) have historically been used in sequence modelling [31], with strong results on a variety of natural language processing tasks [36]. Given the natural interpretation of time series data as sequences of inputs and targets, many RNN-based architectures have been developed for temporal forecasting applications [37, 38, 39, 40]. At its core, RNN cells contain an internal memory state which acts as a compact summary of past information. The memory state is recursively updated with new observations at each time step as shown in Figure 1b, i.e.:

$$\boldsymbol{z}_t = \nu\left(\boldsymbol{z}_{t-1}, y_t, \boldsymbol{x}_t, \boldsymbol{s}\right), \tag{2.7}$$

Where $\boldsymbol{z}_t \in \mathbb{R}^{\mathcal{H}}$ here is the hidden internal state of the RNN, and $\nu(.)$ is the learnt memory update function. For instance, the Elman RNN [41], one of the simplest RNN variants, would take the form below:

$$y_{t+1} = \gamma_y(\boldsymbol{W}_y \boldsymbol{z}_t + \boldsymbol{b}_y), \tag{2.8}$$

$$\boldsymbol{z}_t = \gamma_z(\boldsymbol{W}_{z_1} \boldsymbol{z}_{t-1} + \boldsymbol{W}_{z_2} y_t + \boldsymbol{W}_{z_3} \boldsymbol{x}_t + \boldsymbol{W}_{z_4} \boldsymbol{s} + \boldsymbol{b}_z), \tag{2.9}$$

Where $\boldsymbol{W}_{\cdot}, \boldsymbol{b}_{\cdot}$ are the linear weights and biases of the network respectively, and $\gamma_y(.), \gamma_z(.)$ are network activation functions. Note that RNNs do not require the explicit specification of a lookback window as per the CNN case. From a signal processing perspective, the main recurrent layer – i.e. Equation (2.9) – thus resembles a non-linear version of infinite impulse response (IIR) filters.

Long Short-term Memory Due to the infinite lookback window, older variants of RNNs can suffer from limitations in learning long-range dependencies in the data [42, 43] – due to issues with exploding and vanishing gradients [31]. Intuitively, this can be seen as a form of resonance in the memory state. Long Short-Term Memory networks (LSTMs) [44] were hence developed to address these limitations, by improving gradient flow within the network. This is achieved through the use of a cell state $\boldsymbol{c}_t$ which stores long-term information, modulated through a series of gates as below:

$$\textit{Input gate}: \quad \boldsymbol{i}_t = \sigma(\boldsymbol{W}_{i_1} \boldsymbol{z}_{t-1} + \boldsymbol{W}_{i_2} y_t + \boldsymbol{W}_{i_3} \boldsymbol{x}_t + \boldsymbol{W}_{i_4} \boldsymbol{s} + \boldsymbol{b}_i), \tag{2.10}$$

$$\textit{Output gate}: \quad \boldsymbol{o}_t = \sigma(\boldsymbol{W}_{o_1} \boldsymbol{z}_{t-1} + \boldsymbol{W}_{o_2} y_t + \boldsymbol{W}_{o_3} \boldsymbol{x}_t + \boldsymbol{W}_{o_4} \boldsymbol{s} + \boldsymbol{b}_o), \tag{2.11}$$

$$\textit{Forget gate}: \quad \boldsymbol{f}_t = \sigma(\boldsymbol{W}_{f_1} \boldsymbol{z}_{t-1} + \boldsymbol{W}_{f_2} y_t + \boldsymbol{W}_{f_3} \boldsymbol{x}_t + \boldsymbol{W}_{f_4} \boldsymbol{s} + \boldsymbol{b}_f), \tag{2.12}$$

where $\boldsymbol{z}_{t-1}$ is the hidden state of the LSTM, and $\sigma(.)$ is the sigmoid activation function. The gates modify the hidden and cell states of the LSTM as below:

$$\textit{Hidden state}: \quad \boldsymbol{z}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{2.13}$$

$$\textit{Cell state}: \quad \boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_{c_1} \boldsymbol{z}_{t-1} + \boldsymbol{W}_{c_2} y_t + \boldsymbol{W}_{c_3} \boldsymbol{x}_t + \boldsymbol{W}_{c_4} \boldsymbol{s} + \boldsymbol{b}_c), \tag{2.14}$$

Where $\odot$ is the element-wise (Hadamard) product, and $\tanh(.)$ is the tanh activation function.

Relationship to Bayesian Filtering As examined in [39], Bayesian filters [45] and RNNs are both similar in their maintenance of a hidden state which is recursively updated over time. For Bayesian filters, such as the Kalman filter [46], inference is performed by updating the sufficient statistics of the latent state – using a series of state transition and error correction steps. As the Bayesian filtering steps use deterministic equations to modify sufficient statistics, the RNN can be viewed as a simultaneous approximation of both steps – with the memory vector containing all relevant information required for prediction.

(iii) Attention Mechanisms

The development of attention mechanisms [47, 48] has also lead to improvements in long-term dependency learning – with Transformer architectures achieving state-of-the-art performance in multiple natural language processing applications [12, 49, 50]. Attention layers aggregate temporal features using dynamically generated weights (see Figure 1c), allowing the network to directly focus on significant time steps in the past – even if they are very far back in the lookback window. Conceptually, attention is a mechanism for a key-value lookup based on a given query [51], taking the form below:

$$\boldsymbol{h}_t = \sum_{\tau=0}^{k} \alpha(\boldsymbol{\kappa}_t, \boldsymbol{q}_\tau)\boldsymbol{v}_{t-\tau}, \tag{2.15}$$

Where the key $\boldsymbol{\kappa}_t$, query $\boldsymbol{q}_\tau$ and value $\boldsymbol{v}_{t-\tau}$ are intermediate features produced at different time steps by lower levels of the network. Furthermore, $\alpha(\boldsymbol{\kappa}_t, \boldsymbol{q}_\tau) \in [0, 1]$ is the attention weight for $t-\tau$ generated at time t, and $\boldsymbol{h}_t$ is the context vector output of the attention layer. Note that multiple attention layers can also be used together as per the CNN case, with the output from the final layer forming the encoded latent variable $\boldsymbol{z}_t$.

Recent work has also demonstrated the benefits of using attention mechanisms in time series forecasting applications, with improved performance over comparable recurrent networks [52, 53, 54]. For instance, [52] use attention to aggregate features extracted by RNN encoders, with attention weights produced as below:

$$\boldsymbol{\alpha}(t) = \text{softmax}(\boldsymbol{\eta}_t), \tag{2.16}$$

$$\boldsymbol{\eta}_t = \mathbf{W}_{\eta_1} \tanh(\mathbf{W}_{\eta_2}\boldsymbol{\kappa}_{t-1} + \mathbf{W}_{\eta_3}\boldsymbol{q}_\tau + \boldsymbol{b}_\eta), \tag{2.17}$$

where $\boldsymbol{\alpha}(t) = [\alpha(t, 0), \ldots \alpha(t, k)]$ is a vector of attention weights, $\boldsymbol{\kappa}_{t-1}, \boldsymbol{q}_t$ are outputs from LSTM encoders used for feature extraction, and softmax(.) is the softmax activation function. More recently, Transformer architectures have also been considered in [53, 54], which apply scalar-dot product self-attention [49] to features extracted within the lookback window. From a time series modelling perspective, attention provides two key benefits. Firstly, networks with attention are able to directly attend to any significant events that occur. In retail forecasting applications, for example, this includes holiday or promotional periods which can have a positive effect on sales. Secondly, as shown in [54], attention-based networks can also learn regime-specific temporal dynamics – by using distinct attention weight patterns for each regime.

(iv) Outputs and Loss Functions

Given the flexibility of neural networks, deep neural networks have been used to model both discrete [55] and continuous [37, 56] targets – by customising of decoder and output layer of the neural network to match the desired target type. In one-step-ahead prediction problems, this can be as simple as combining a linear transformation of encoder outputs (i.e. Equation (2.2)) together with an appropriate output activation for the target. Regardless of the form of the target, predictions can be further divided into two different categories – point estimates and probabilistic forecasts.

Point Estimates A common approach to forecasting is to determine the expected value of a future target. This essentially involves reformulating the problem to a classification task for discrete outputs (e.g. forecasting future events), and regression task for continuous outputs – using the encoders described above. For the binary classification case, the final layer of the decoder then features a linear layer with a sigmoid activation function – allowing the network to predict the probability of event occurrence at a given time step. For one-step-ahead forecasts of binary and continuous targets, networks are trained using binary cross-entropy and mean square error loss

functions respectively:

$$\mathcal{L}_{classification} = -\frac{1}{T}\sum_{t=1}^{T} y_t \log(\hat{y}_t) + (1-y_t)\log(1-\hat{y}_t) \tag{2.18}$$

$$\mathcal{L}_{regression} = \frac{1}{T}\sum_{t=1}^{T} (y_t - \hat{y}_t)^2 \tag{2.19}$$

While the loss functions above are the most common across applications, we note that the flexibility of neural networks also allows for more complex losses to be adopted - e.g. losses for quantile regression [56] and multinomial classification [32].

Probabilistic Outputs While point estimates are crucial to predicting the future value of a target, understanding the uncertainty of a model's forecast can be useful for decision makers in different domains. When forecast uncertainties are wide, for instance, model users can exercise more caution when incorporating predictions into their decision making, or alternatively rely on other sources of information. In some applications, such as financial risk management, having access to the full predictive distribution will allow decision makers to optimise their actions in the presence of rare events – e.g. allowing risk managers to insulate portfolios against market crashes.

A common way to model uncertainties is to use deep neural networks to generate parameters of known distributions [27, 37, 38]. For example, Gaussian distributions are typically used for forecasting problems with continuous targets, with the networks outputting means and variance parameters for the predictive distributions at each step as below:

$$y_{t+\tau} \sim N(\mu(t,\tau), \zeta(t,\tau)^2), \tag{2.20}$$

$$\mu(t,\tau) = \boldsymbol{W}_{\mu}\boldsymbol{h}_t^L + \boldsymbol{b}_{\mu}, \tag{2.21}$$

$$\zeta(t,\tau) = \text{softplus}(\boldsymbol{W}_{\Sigma}\boldsymbol{h}_t^L + \boldsymbol{b}_{\Sigma}), \tag{2.22}$$

where $\boldsymbol{h}_t^L$ is the final layer of the network, and softplus$(.)$ is the softplus activation function to ensure that standard deviations take only positive values.

(b) Multi-horizon Forecasting Models

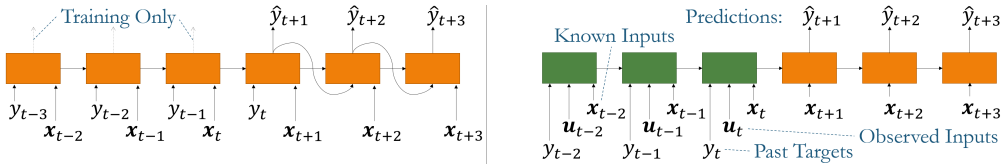
In many applications, it is often beneficial to have access to predictive estimates at multiple points in the future – allowing decision makers to visualise trends over a future horizon, and optimise their actions across the entire path. From a statistical perspective, multi-horizon forecasting can be viewed as a slight modification of one-step-ahead prediction problem (i.e. Equation (2.1)) as below:

$$\hat{y}_{t+\tau} = f(y_{t-k:t}, \boldsymbol{x}_{t-k:t}, \boldsymbol{u}_{t-k:t+\tau}, \boldsymbol{s}, \tau), \tag{2.23}$$

where $\tau \in \{1, \ldots, \tau_{max}\}$ is a discrete forecast horizon, $\boldsymbol{u}_t$ are known future inputs (e.g. date information, such as the day-of-week or month) across the entire horizon, and $\boldsymbol{x}_t$ are inputs that can only be observed historically. In line with traditional econometric approaches [57, 58], deep learning architectures for multi-horizon forecasting can be divided into iterative and direct methods – as shown in Figure 2 and described in detail below.

(i) Iterative Methods

Iterative approaches to multi-horizon forecasting typically make use of autoregressive deep learning architectures [37, 39, 40, 53] – producing multi-horizon forecasts by recursively feeding samples of the target into future time steps (see Figure 2a). By repeating the procedure to generate multiple trajectories, forecasts are then produced using the sampling distributions for target values at each step. For instance, predictive means can be obtained using the Monte Carlo estimate $\hat{y}_{t+\tau} = \sum_{j=1}^{J} \tilde{y}_{t+\tau}^{(j)}/J$, where $\tilde{y}_{t+\tau}^{(j)}$ is a sample taken based on the model of Equation (2.20). As autoregressive models are trained in the exact same fashion as one-step-ahead prediction models



(a) Iterative Methods (b) Direct Methods

Figure 2: Main types of multi-horizon forecasting models. Colours used to distinguish between model weights – with iterative models using a common model across the entire horizon and direct methods taking a sequence-to-sequence approach.

(i.e. via backpropagation through time), the iterative approach allows for the easy generalisation of standard models to multi-step forecasting. However, as a small amount of error is produced at each time step, the recursive structure of iterative methods can potentially lead to large error accumulations over longer forecasting horizons. In addition, iterative methods assume that all inputs but the target are known at run-time – requiring only samples of the target to be fed into future time steps. This can be a limitation in many practical scenarios where observed inputs exist, motivating the need for more flexible methods.

(ii) Direct Methods

Direct methods alleviate the issues with iterative methods by producing forecasts directly using all available inputs. They typically make use of sequence-to-sequence architectures [52, 54, 56], using an encoder to summarise past information (i.e. targets, observed inputs and a priori known inputs), and a decoder to combine them with known future inputs – as depicted in Figure 2b. As described in [59], alternative approach is to use simpler models to directly produce a fixed-length vector matching the desired forecast horizon. This, however, does require the specification of a maximum forecast horizon (i.e. τ_{max}), with predictions made only at the predefined discrete intervals.

3. Incorporating Domain Knowledge with Hybrid Models

Despite its popularity, the efficacy of machine learning for time series prediction has historically been questioned – as evidenced by forecasting competitions such as the M-competitions [60]. Prior to the M4 competition of 2018 [61], the prevailing wisdom was that sophisticated methods do not produce more accurate forecasts, and simple models with ensembling had a tendency to do better [59, 62, 63]. Two key reasons have been identified to explain the underperformance of machine learning methods. Firstly, the flexibility of machine learning methods can be a double-edged sword – making them prone to overfitting [59]. Hence, simpler models may potentially do better in low data regimes, which are particularly common in forecasting problems with a small number of historical observations (e.g. quarterly macroeconomic forecasts). Secondly, similar to stationarity requirements of statistical models, machine learning models can be sensitive to how inputs are pre-processed [26, 37, 59], which ensure that data distributions at training and test time are similar.

A recent trend in deep learning has been in developing hybrid models which address these limitations, demonstrating improved performance over pure statistical or machine learning models in a variety of applications [38, 64, 65, 66]. Hybrid methods combine well-studied quantitative time series models together with deep learning – using deep neural networks to generate model parameters at each time step. On the one hand, hybrid models allow domain experts to inform neural network training using prior information – reducing the hypothesis space of the network and improving generalisation. This is especially useful for small datasets [38], where there is a greater risk of overfitting for deep learning models. Furthermore, hybrid models allow for the separation of stationary and non-stationary components, and avoid the need for custom input pre-processing. An example of this is the Exponential Smoothing RNN (ES-RNN) [64], winner of the M4 competition, which uses exponential smoothing to capture non-stationary trends and

learns additional effects with the RNN. In general, hybrid models utilise deep neural networks in two manners: a) to encode time-varying parameters for non-probabilistic parametric models [64, 65, 67], and b) to produce parameters of distributions used by probabilistic models [38, 40, 66].

(a) Non-probabilistic Hybrid Models

With parametric time series models, forecasting equations are typically defined analytically and provide point forecasts for future targets. Non-probabilistic hybrid models hence modify these forecasting equations to combine statistical and deep learning components. The ES-RNN for example, utilises the update equations of the Holt-Winters exponential smoothing model [8] – combining multiplicative level and seasonality components with deep learning outputs as below:

$$\hat{y}_{i,t+\tau} = \exp(\boldsymbol{W}_{ES}\boldsymbol{h}^L_{i,t+\tau} + \boldsymbol{b}_{ES}) \times l_{i,t} \times \gamma_{i,t+\tau}, \tag{3.1}$$

$$l_{i,t} = \beta^{(i)}_1 y_{i,t}/\gamma_{i,t} + (1-\beta^{(i)}_1) l_{i,t-1}, \tag{3.2}$$

$$\gamma_{i,t} = \beta^{(i)}_2 y_{i,t}/l_{i,t} + (1-\beta^{(i)}_2) \gamma_{i,t-\kappa}, \tag{3.3}$$

where $\boldsymbol{h}^L_{i,t+\tau}$ is the final layer of the network for the τth-step-ahead forecast, $l_{i,t}$ is a level component, $\gamma_{i,t}$ is a seasonality component with period κ, and $\beta^{(i)}_1, \beta^{(i)}_2$ are entity-specific static coefficients. From the above equations, we can see that the exponential smoothing components $(l_{i,t}, \gamma_{i,t})$ handle the broader (e.g. exponential) trends within the datasets, reducing the need for additional input scaling.

(b) Probabilistic Hybrid Models

Probabilistic hybrid models can also be used in applications where distribution modelling is important – utilising probabilistic generative models for temporal dynamics such as Gaussian processes [40] and linear state space models [38]. Rather than modifying forecasting equations, probabilistic hybrid models use neural networks to produce parameters for predictive distributions at each step. For instance, Deep State Space Models [38] encode time-varying parameters for linear state space models as below – performing inference via the Kalman filtering equations [46]:

$$y_t = \boldsymbol{a}(\boldsymbol{h}^L_{i,t+\tau})^T \boldsymbol{l}_t + \phi(\boldsymbol{h}^L_{i,t+\tau})\epsilon_t, \tag{3.4}$$

$$\boldsymbol{l}_t = \boldsymbol{F}(\boldsymbol{h}^L_{i,t+\tau})\boldsymbol{l}_{t-1} + \boldsymbol{q}(\boldsymbol{h}^L_{i,t+\tau}) + \boldsymbol{\Sigma}(\boldsymbol{h}^L_{i,t+\tau}) \odot \boldsymbol{\Sigma}_t, \tag{3.5}$$

where $\boldsymbol{l}_t$ is the hidden latent state, $\boldsymbol{a}(.)$, $\boldsymbol{F}(.)$, $\boldsymbol{q}(.)$ are linear transformations of $\boldsymbol{h}^L_{i,t+\tau}$, $\phi(.)$, $\boldsymbol{\Sigma}(.)$ are linear transformations with softmax activations, $\epsilon_t \sim N(0,1)$ is a univariate residual and $\boldsymbol{\Sigma}_t \sim N(0, \mathbb{I})$ is a multivariate normal random variable.

4. Facilitating Decision Support Using Deep Neural Networks

Although model builders are mainly concerned with the accuracy of their forecasts, end-users typically use predictions to *guide their future actions*. For instance, doctors can make use of clinical forecasts (e.g. probabilities of disease onset and mortality) to help them prioritise tests to order, formulate a diagnosis and determine a course of treatment. As such, while time series forecasting is a crucial preliminary step, a better understanding of both temporal dynamics and the motivations behind a model's forecast can help users further optimise their actions. In this section, we explore two directions in which neural networks have been extended to facilitate decision support with time series data – focusing on methods in interpretability and causal inference.

(a) Interpretability With Time Series Data

With the deployment of neural networks in mission-critical applications [68], there is a increasing need to understand both *how* and *why* a model makes a certain prediction. Moreover, end-users can

have little prior knowledge with regards to the relationships present in their data, with datasets growing in size and complexity in recent times. Given the black-box nature of standard neural network architectures, a new body of research has emerged in methods for interpreting deep learning models. We present a summary below – referring the reader to dedicated surveys for more in-depth analyses [69, 70].

Techniques for Post-hoc Interpretability Post-hoc interpretable models are developed to interpret trained networks, and helping to identify important features or examples without modifying the original weights. Methods can mainly be divided into two main categories. Firstly, one possible approach is to apply simpler interpretable surrogate models between the inputs and outputs of the neural network, and rely on the approximate model to provide explanations. For instance, Local Interpretable Model-Agnostic Explanations (LIME) [71] identify relevant features by fitting instance-specific linear models to perturbations of the input, with the linear coefficients providing a measure of importance. Shapley additive explanations (SHAP) [72] provide another surrogate approach, which utilises Shapley values from cooperative game theory to identify important features across the dataset. Next, gradient-based method – such as saliency maps [73, 74] and influence functions [75] – have been proposed, which analyse network gradients to determine which input features have the greatest impact on loss functions. While post-hoc interpretability methods can help with feature attributions, they typically ignore any sequential dependencies between inputs – making it difficult to apply them to complex time series datasets.

Inherent Interpretability with Attention Weights An alternative approach is to directly design architectures with explainable components, typically in the form of strategically placed attention layers. As attention weights are produced as outputs from a softmax layer, the weights are constrained to sum to 1, i.e. $\sum_{\tau=0}^{k} \alpha(t, \tau) = 1$. For time series models, the outputs of Equation (2.15) can hence also be interpreted as a weighted average over temporal features, using the weights supplied by the attention layer at each step. An analysis of attention weights can then be used to understand the relative importance of features at each time step. Instance-wise interpretability studies have been performed in [53, 55, 76], where the authors used specific examples to show how the magnitudes of $\alpha(t, \tau)$ can indicate which time points were most significant for predictions. By analysing distributions of attention vectors across time, [54] also shows how attention mechanisms can be used to identify persistent temporal relationships – such as seasonal patterns – in the dataset.

(b) Counterfactual Predictions & Causal Inference Over Time

In addition to understanding the relationships learnt by the networks, deep learning can also help to facilitate decision support by producing predictions outside of their observational datasets, or counterfactual forecasts. Counterfactual predictions are particularly useful for scenario analysis applications – allowing users to evaluate how different sets of actions can impact target trajectories. This can be useful both from a historical angle, i.e. determining what would have happened if a different set of circumstances had occurred, and from a forecasting perspective, i.e. determining which actions to take to optimise future outcomes.

While a large class of deep learning methods exists for estimating causal effects in static settings [77, 78, 79], the key challenge in time series datasets is the presence of time-dependent confounding effects. This arises due to circular dependencies when actions that can affect the target are also conditional on observations of the target. Without any adjusting for time-dependent confounders, straightforward estimations techniques can results in biased results, as shown in [80]. Recently, several methods have emerged to train deep neural networks while adjusting for time-dependent confounding, based on extensions of statistical techniques and the design of new loss functions. With statistical methods, [81] extends the inverse-probability-of-treatment-weighting (IPTW) approach of marginal structural models in epidemiology – using one set of networks to estimate treatment application probabilities, and a sequence-to-sequence model to learn unbiased predictions. Another approach in [82] extends the G-computation framework, jointly modelling

distributions of the target and actions using deep learning. In addition, new loss functions have been proposed in [83], which adopts domain adversarial training to learn balanced representations of patient history.

5. Conclusions and Future Directions

With the growth in data availability and computing power in recent times, deep neural networks architectures have achieved much success in forecasting problems across multiple domains. In this article, we survey the main architectures used for time series forecasting – highlighting the key building blocks used in neural network design. We examine how they incorporate temporal information for one-step-ahead predictions, and describe how they can be extended for use in multi-horizon forecasting. Furthermore, we outline the recent trend of hybrid deep learning models, which combine statistical and deep learning components to outperform pure methods in either category. Finally, we summarise two ways in which deep learning can be extended to improve decision support over time, focusing on methods in interpretability and counterfactual prediction.

Although a large number of deep learning models have been developed for time series forecasting, some limitations still exist. Firstly, deep neural networks typically require time series to be discretised at regular intervals, making it difficult to forecast datasets where observations can be missing or arrive at random intervals. While some preliminary research on continuous-time models has been done via Neural Ordinary Differential Equations [84], additional work needs to be done to extend this work for datasets with complex inputs (e.g. static variables) and to benchmark them against existing models. In addition, as mentioned in [85], time series often have a hierarchical structure with logical groupings between trajectories – e.g. in retail forecasting, where product sales in the same geography can be affected by common trends. As such, the development of architectures which explicit account for such hierarchies could be an interesting research direction, and potentially improve forecasting performance over existing univariate or multivariate models.

Competing Interests. The author(s) declare that they have no competing interests.

References

1 Mudelsee M. Trend analysis of climate time series: A review of methods. Earth-Science Reviews. 2019;190:310 – 322.
2 Stoffer DS, Ombao H. Editorial: Special issue on time series analysis in the biological sciences. Journal of Time Series Analysis. 2012;33(5):701–703.
3 Topol EJ. High-performance medicine: the convergence of human and artificial intelligence. Nature Medicine. 2019 Jan;25(1):44–56.
4 Böse JH, Flunkert V, Gasthaus J, Januschowski T, Lange D, Salinas D, et al. Probabilistic Demand Forecasting at Scale. Proc VLDB Endow. 2017 Aug;10(12):1694–1705.
5 Andersen TG, Bollerslev T, Christoffersen PF, Diebold FX. Volatility Forecasting. National Bureau of Economic Research; 2005. 11188.
6 Box GEP, Jenkins GM. Time Series Analysis: Forecasting and Control. Holden-Day; 1976.
7 Gardner Jr ES. Exponential smoothing: The state of the art. Journal of Forecasting. 1985;4(1):1–28.
8 Winters PR. Forecasting Sales by Exponentially Weighted Moving Averages. Management Science. 1960;6(3):324–342.
9 Harvey AC. Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press; 1990.
10 Ahmed NK, Atiya AF, Gayar NE, El-Shishiny H. An Empirical Comparison of Machine Learning Models for Time Series Forecasting. Econometric Reviews. 2010;29(5-6):594–621.
11 Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in Neural Information Processing Systems 25 (NIPS); 2012. p. 1097–1105.
12 Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers); 2019. p. 4171–4186.

13 Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529:484–503.
14 Baxter J. A Model of Inductive Bias Learning. J Artif Int Res. 2000;12(1):149–198.
15 Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013;35(8):1798–1828.
16 Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al.. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Software available from tensorflow.org. Available from: http://tensorflow.org/.
17 Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems 32; 2019. p. 8024–8035.
18 Hyndman RJ, Khandakar Y. Automatic time series forecasting: the forecast package for R. Journal of Statistical Software. 2008;26(3):1–22.
19 Nadaraya EA. On Estimating Regression. Theory of Probability and Its Applications. 1964;9(1):141–142.
20 Smola AJ, Schölkopf B. A Tutorial on Support Vector Regression. Statistics and Computing. 2004;14(3):199–222.
21 Williams CKI, Rasmussen CE. Gaussian Processes for Regression. In: Advances in Neural Information Processing Systems (NIPS); 1996. .
22 Damianou A, Lawrence N. Deep Gaussian Processes. In: Proceedings of the Conference on Artificial Intelligence and Statistics (AISTATS); 2013. .
23 Garnelo M, Rosenbaum D, Maddison C, Ramalho T, Saxton D, Shanahan M, et al. Conditional Neural Processes. In: Proceedings of the International Conference on Machine Learning (ICML); 2018. .
24 Waibel A. Modular Construction of Time-Delay Neural Networks for Speech Recognition. Neural Comput. 1989;1(1):39–46.
25 Wan E. Time Series Prediction by Using a Connectionist Network with Internal Delay Lines. In: Time Series Prediction. Addison-Wesley; 1994. p. 195–217.
26 Sen R, Yu HF, Dhillon I. Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting. In: Advances in Neural Information Processing Systems (NeurIPS); 2019. .
27 Wen R, Torkkola K. Deep Generative Quantile-Copula Models for Probabilistic Forecasting. In: ICML Time Series Workshop; 2019. .
28 Li Y, Yu R, Shahabi C, Liu Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In: (Proceedings of the International Conference on Learning Representations ICLR); 2018. .
29 Ghaderi A, Sanandaji BM, Ghaderi F. Deep Forecast: Deep Learning-based Spatio-Temporal Forecasting. In: ICML Time Series Workshop; 2017. .
30 Salinas D, Bohlke-Schneider M, Callot L, Medico R, Gasthaus J. High-dimensional multivariate forecasting with low-rank Gaussian Copula Processes. In: Advances in Neural Information Processing Systems (NeurIPS); 2019. .
31 Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.
32 van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, et al. WaveNet: A Generative Model for Raw Audio. arXiv e-prints. 2016 Sep;p. arXiv:1609.03499.
33 Bai S, Zico Kolter J, Koltun V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv e-prints. 2018;p. arXiv:1803.01271.
34 Borovykh A, Bohte S, Oosterlee CW. Conditional Time Series Forecasting with Convolutional Neural Networks. arXiv e-prints. 2017;p. arXiv:1703.04691.
35 Lyons RG. Understanding Digital Signal Processing (2nd Edition). USA: Prentice Hall PTR; 2004.
36 Young T, Hazarika D, Poria S, Cambria E. Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. IEEE Computational Intelligence Magazine. 2018;13(3):55–75.
37 Salinas D, Flunkert V, Gasthaus J. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. arXiv e-prints. 2017;p. arXiv:1704.04110.
38 Rangapuram SS, Seeger MW, Gasthaus J, Stella L, Wang Y, Januschowski T. Deep State Space Models for Time Series Forecasting. In: Advances in Neural Information Processing Systems (NIPS); 2018. .

39 Lim B, Zohren S, Roberts S. Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction. In: International Joint Conference on Neural Networks (IJCNN); 2020. .
40 Wang Y, Smola A, Maddix D, Gasthaus J, Foster D, Januschowski T. Deep Factors for Forecasting. In: Proceedings of the International Conference on Machine Learning (ICML); 2019. .
41 Elman JL. Finding structure in time. Cognitive Science. 1990;14(2):179 – 211.
42 Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks. 1994;5(2):157–166.
43 Kolen JF, Kremer SC. In: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies; 2001. p. 237–243.
44 Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997 Nov;9(8):1735–1780.
45 Srkk S. Bayesian Filtering and Smoothing. Cambridge University Press; 2013.
46 Kalman RE. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering. 1960;82(1):35.
47 Bahdanau D, Cho K, Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate. In: Proceedings of the International Conference on Learning Representations (ICLR); 2015. .
48 Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. 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); 2014. .
49 Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is All you Need. In: Advances in Neural Information Processing Systems (NIPS); 2017. .
50 Dai Z, Yang Z, Yang Y, Carbonell J, Le Q, Salakhutdinov R. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL); 2019. .
51 Graves A, Wayne G, Danihelka I. Neural Turing Machines. CoRR. 2014;abs/1410.5401.
52 Fan C, Zhang Y, Pan Y, Li X, Zhang C, Yuan R, et al. Multi-Horizon Time Series Forecasting with Temporal Attention Learning. In: Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining (KDD); 2019. .
53 Li S, Jin X, Xuan Y, Zhou X, Chen W, Wang YX, et al. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In: Advances in Neural Information Processing Systems (NeurIPS); 2019. .
54 Lim B, Arik SO, Loeff N, Pfister T. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. arXiv e-prints. 2019;p. arXiv:1912.09363.
55 Choi E, Bahadori MT, Sun J, Kulas JA, Schuetz A, Stewart WF. RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism. In: Advances in Neural Information Processing Systems (NIPS); 2016. .
56 Wen R, et al. A Multi-Horizon Quantile Recurrent Forecaster. In: NIPS 2017 Time Series Workshop; 2017. .
57 Taieb SB, Sorjamaa A, Bontempi G. Multiple-output modeling for multi-step-ahead time series forecasting. Neurocomputing. 2010;73(10):1950 – 1957.
58 Marcellino M, Stock J, Watson M. A Comparison of Direct and Iterated Multistep AR Methods for Forecasting Macroeconomic Time Series. Journal of Econometrics. 2006;135:499–526.
59 Makridakis S, Spiliotis E, Assimakopoulos V. Statistical and Machine Learning forecasting methods: Concerns and ways forward. PLOS ONE. 2018 03;13(3):1–26.
60 Hyndman R. A brief history of forecasting competitions. International Journal of Forecasting. 2020;36(1):7–14.
61 The M4 Competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting. 2020;36(1):54 – 74.
62 Fildes R, Hibon M, Makridakis S, Meade N. Generalising about univariate forecasting methods: further empirical evidence. International Journal of Forecasting. 1998;14(3):339 – 358.
63 Makridakis S, Hibon M. The M3-Competition: results, conclusions and implications. International Journal of Forecasting. 2000;16(4):451 – 476. The M3- Competition.
64 Smyl S. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting. 2020;36(1):75 – 85. M4 Competition.
65 Lim B, Zohren S, Roberts S. Enhancing Time-Series Momentum Strategies Using Deep Neural Networks. The Journal of Financial Data Science. 2019;.

66 Grover A, Kapoor A, Horvitz E. A Deep Hybrid Model for Weather Forecasting. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD); 2015. .

67 Binkowski M, Marti G, Donnat P. Autoregressive Convolutional Neural Networks for Asynchronous Time Series. In: Proceedings of the International Conference on Machine Learning (ICML); 2018. .

68 Moraffah R, Karami M, Guo R, Raglin A, Liu H. Causal Interpretability for Machine Learning – Problems, Methods and Evaluation. arXiv e-prints. 2020;p. arXiv:2003.03934.

69 Chakraborty S, Tomsett R, Raghavendra R, Harborne D, Alzantot M, Cerutti F, et al. Interpretability of deep learning models: A survey of results. In: 2017 IEEE SmartWorld Conference Proceedings); 2017. p. 1–6.

70 Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence. 2019 May;1(5):206–215.

71 Ribeio M, Singh S, Guestrin C. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In: KDD; 2016. .

72 Lundberg S, Lee SI. A Unified Approach to Interpreting Model Predictions. In: Advances in Neural Information Processing Systems (NIPS); 2017. .

73 Simonyan K, Vedaldi A, Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. arXiv e-prints. 2013;p. arXiv:1312.6034.

74 Siddiqui SA, Mercier D, Munir M, Dengel A, Ahmed S. TSViz: Demystification of Deep Learning Models for Time-Series Analysis. IEEE Access. 2019;7:67027–67040.

75 Koh PW, Liang P. Understanding Black-box Predictions via Influence Functions. In: Proceedings of the International Conference on Machine Learning(ICML; 2017. .

76 Bai T, Zhang S, Egleston BL, Vucetic S. Interpretable Representation Learning for Healthcare via Capturing Disease Progression through Time. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD); 2018. .

77 Yoon J, Jordon J, van der Schaar M. GANITE: Estimation of Individualized Treatment Effects using Generative Adversarial Nets. In: International Conference on Learning Representations (ICLR); 2018. .

78 Hartford J, Lewis G, Leyton-Brown K, Taddy M. Deep IV: A Flexible Approach for Counterfactual Prediction. In: Proceedings of the 34th International Conference on Machine Learning (ICML); 2017. .

79 Alaa AM, Weisz M, van der Schaar M. Deep Counterfactual Networks with Propensity Dropout. In: Proceedings of the 34th International Conference on Machine Learning (ICML); 2017. .

80 Mansournia MA, Etminan M, Danaei G, Kaufman JS, Collins G. Handling time varying confounding in observational research. BMJ. 2017;359.

81 Lim B, Alaa A, van der Schaar M. Forecasting Treatment Responses Over Time Using Recurrent Marginal Structural Networks. In: NeurIPS; 2018. .

82 Li R, Shahn Z, Li J, Lu M, Chakraborty P, Sow D, et al. G-Net: A Deep Learning Approach to G-computation for Counterfactual Outcome Prediction Under Dynamic Treatment Regimes. arXiv e-prints. 2020;p. arXiv:2003.10551.

83 Bica I, Alaa AM, Jordon J, van der Schaar M. Estimating counterfactual treatment outcomes over time through adversarially balanced representations. In: International Conference on Learning Representations(ICLR); 2020. .

84 Chen RTQ, Rubanova Y, Bettencourt J, Duvenaud D. Neural Ordinary Differential Equations. In: Proceedings of the International Conference on Neural Information Processing Systems (NIPS); 2018. .

85 Fry C, Brundage M. The M4 Forecasting Competition – A Practitioner's View. International Journal of Forecasting. 2019;.

Chapter 3

Novel Architectures For Time Series Forecasting

Publications Included

- **B. Lim**, S. O. Arik, N. Loeff, T. Pfister. *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting.* Submitted, 2020. Weblink: `https://arxiv.org/abs/1912.09363`.
- **B. Lim**, S. Zohren, S. Roberts. *Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction.* Proceedings of the International Joint Conference On Neural Networks (IJCNN), 2020. Weblink: `https://arxiv.org/abs/1901.08096`.

Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting

Bryan Lim[a,1,*], Sercan Ö. Arık[b], Nicolas Loeff[b], Tomas Pfister[b]

[a] *University of Oxford, UK*
[b] *Google Cloud AI, USA*

Abstract

Multi-horizon forecasting often contains a complex mix of inputs – including static (i.e. time-invariant) covariates, known future inputs, and other exogenous time series that are only observed in the past – without any prior information on how they interact with the target. Several deep learning methods have been proposed, but they are typically 'black-box' models which do not shed light on how they use the full range of inputs present in practical scenarios. In this paper, we introduce the Temporal Fusion Transformer (TFT) – a novel attention-based architecture which combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics. To learn temporal relationships at different scales, TFT uses recurrent layers for local processing and interpretable self-attention layers for long-term dependencies. TFT utilizes specialized components to select relevant features and a series of gating layers to suppress unnecessary components, enabling high performance in a wide range of scenarios. On a variety of real-world datasets, we demonstrate significant performance improvements over existing benchmarks, and showcase three practical interpretability use cases of TFT.

Keywords: Deep learning, Interpretability, Time series, Multi-horizon forecasting, Attention mechanisms, Explainable AI.

1. Introduction

Multi-horizon forecasting, i.e. the prediction of variables-of-interest at multiple future time steps, is a crucial problem within time series machine learning. In contrast to one-step-ahead predictions, multi-horizon forecasts provide users with access to estimates across the entire path, allowing them to optimize their actions at multiple steps in future (e.g. retailers optimizing the inventory for

*Corresponding authors

Email addresses: blim@robots.ox.ac.uk (Bryan Lim), soarik@google.com (Sercan Ö. Arık), nloeff@google.com (Nicolas Loeff), tpfister@google.com (Tomas Pfister)

[1] Completed as part of internship with Google Cloud AI Research.

the entire upcoming season, or clinicians optimizing a treatment plan for a patient). Multi-horizon forecasting has many impactful real-world applications in retail [1, 2], healthcare [3, 4] and economics [5]) – performance improvements to existing methods in such applications are highly valuable.

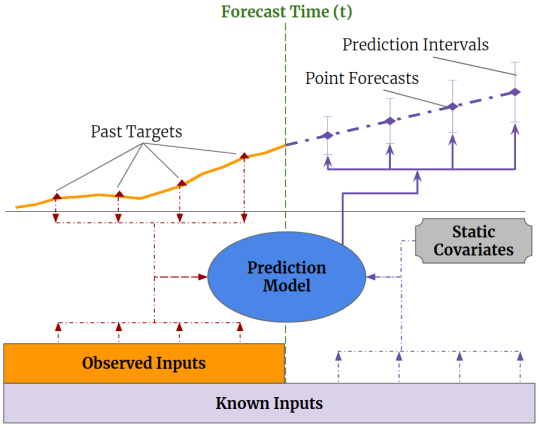


Figure 1: Illustration of multi-horizon forecasting with static covariates, past-observed and apriori-known future time-dependent inputs.

Practical multi-horizon forecasting applications commonly have access to a variety of data sources, as shown in Fig. 1, including known information about the future (e.g. upcoming holiday dates), other exogenous time series (e.g. historical customer foot traffic), and static metadata (e.g. location of the store) – without any prior knowledge on how they interact. This heterogeneity of data sources together with little information about their interactions makes multi-horizon time series forecasting particularly challenging.

Deep neural networks (DNNs) have increasingly been used in multi-horizon forecasting, demonstrating strong performance improvements over traditional time series models [6, 7, 8]. While many architectures have focused on variants of recurrent neural network (RNN) architectures [9, 6, 10], recent improvements have also used attention-based methods to enhance the selection of relevant time steps in the past [11] – including Transformer-based models [12]. However, these often fail to consider the different types of inputs commonly present in multi-horizon forecasting, and either assume that all exogenous inputs are known into the future [9, 6, 12] – a common problem with autoregressive models – or neglect important static covariates [10] – which are simply concatenated with other time-dependent features at each step. Many recent improvements in time series models have resulted from the alignment of architectures with unique data characteristics [13, 14]. We argue and demonstrate that similar performance gains can also be reaped by designing networks with suitable inductive biases for multi-horizon forecasting.

In addition to not considering the heterogeneity of common multi-horizon forecasting inputs, most current architectures are 'black-box' models where fore-

casts are controlled by complex nonlinear interactions between many parameters. This makes it difficult to explain how models arrive at their predictions, and in turn makes it challenging for users to trust a model's outputs and model builders to debug it. Unfortunately, commonly-used explainability methods for DNNs are not well-suited for applying to time series. In their conventional form, post-hoc methods (e.g. LIME [15] and SHAP [16]) do not consider the time ordering of input features. For example, for LIME, surrogate models are independently constructed for each data-point, and for SHAP, features are considered independently for neighboring time steps. Such post-hoc approaches would lead to poor explanation quality as dependencies between time steps are typically significant in time series. On the other hand, some attention-based architectures are proposed with inherent interpretability for sequential data, primarily language or speech – such as the Transformer architecture [17]. The fundamental caveat to apply them is that multi-horizon forecasting includes many different types of input features, as opposed to language or speech. In their conventional form, these architectures can provide insights into relevant time steps for multi-horizon forecasting, but they cannot distinguish the importance of different features at a given timestep. Overall, in addition to the need for new methods to tackle the heterogeneity of data in multi-horizon forecasting for high performance, new methods are also needed to render these forecasts interpretable, given the needs of the use cases.

In this paper we propose the Temporal Fusion Transformer (TFT) – an attention-based DNN architecture for multi-horizon forecasting that achieves high performance while enabling new forms of interpretability. To obtain significant performance improvements over state-of-the-art benchmarks, we introduce multiple novel ideas to align the architecture with the full range of potential inputs and temporal relationships common to multi-horizon forecasting – specifically incorporating (1) static covariate encoders which encode context vectors for use in other parts of the network, (2) gating mechanisms throughout and sample-dependent variable selection to minimize the contributions of irrelevant inputs, (3) a sequence-to-sequence layer to locally process known and observed inputs, and (4) a temporal self-attention decoder to learn any long-term dependencies present within the dataset. The use of these specialized components also facilitates interpretability; in particular, we show that TFT enables three valuable interpretability use cases: helping users identify (i) globally-important variables for the prediction problem, (ii) persistent temporal patterns, and (iii) significant events. On a variety of real-world datasets, we demonstrate how TFT can be practically applied, as well as the insights and benefits it provides.

2. Related Work

DNNs for Multi-horizon Forecasting: Similarly to traditional multi-horizon forecasting methods [18, 19], recent deep learning methods can be categorized into iterated approaches using autoregressive models [9, 6, 12] or direct methods based on sequence-to-sequence models [10, 11].

Iterated approaches utilize one-step-ahead prediction models, with multi-step predictions obtained by recursively feeding predictions into future inputs. Approaches with Long Short-term Memory (LSTM) [20] networks have been considered, such as Deep AR [9] which uses stacked LSTM layers to generate parameters of one-step-ahead Gaussian predictive distributions. Deep State-Space Models (DSSM) [6] adopt a similar approach, utilizing LSTMs to generate parameters of a predefined linear state-space model with predictive distributions produced via Kalman filtering – with extensions for multivariate time series data in [21]. More recently, Transformer-based architectures have been explored in [12], which proposes the use of convolutional layers for local processing and a sparse attention mechanism to increase the size of the receptive field during forecasting. Despite their simplicity, iterative methods rely on the assumption that the values of all variables excluding the target are known at forecast time – such that only the target needs to be recursively fed into future inputs. However, in many practical scenarios, numerous useful time-varying inputs exist, with many unknown in advance. Their straightforward use is hence limited for iterative approaches. TFT, on the other hand, explicitly accounts for the diversity of inputs – naturally handling static covariates and (past-observed and future-known) time-varying inputs.

In contrast, direct methods are trained to explicitly generate forecasts for multiple predefined horizons at each time step. Their architectures typically rely on sequence-to-sequence models, e.g. LSTM encoders to summarize past inputs, and a variety of methods to generate future predictions. The Multi-horizon Quantile Recurrent Forecaster (MQRNN) [10] uses LSTM or convolutional encoders to generate context vectors which are fed into multi-layer perceptrons (MLPs) for each horizon. In [11] a multi-modal attention mechanism is used with LSTM encoders to construct context vectors for a bi-directional LSTM decoder. Despite performing better than LSTM-based iterative methods, interpretability remains challenging for such standard direct methods. In contrast, we show that by interpreting attention patterns, TFT can provide insightful explanations about temporal dynamics, and do so while maintaining state-of-the-art performance on a variety of datasets.

Time Series Interpretability with Attention: Attention mechanisms are used in translation [17], image classification [22] or tabular learning [23] to identify salient portions of input for each instance using the magnitude of attention weights. Recently, they have been adapted for time series with interpretability motivations [7, 12, 24], using LSTM-based [25] and transformer-based [12] architectures. However, this was done without considering the importance of static covariates (as the above methods blend variables at each input). TFT alleviates this by using separate encoder-decoder attention for static features at each time step on top of the self-attention to determine the contribution time-varying inputs.

Instance-wise Variable Importance with DNNs: Instance (i.e. sample)-wise variable importance can be obtained with post-hoc explanation methods [15, 16, 26] and inherently intepretable models [27, 24]. Post-hoc explanation methods, e.g. LIME [15], SHAP [16] and RL-LIM [26], are applied on pre-

trained black-box models and often based on distilling into a surrogate interpretable model, or decomposing into feature attributions. They are not designed to take into account the time ordering of inputs, limiting their use for complex time series data. Inherently-interpretable modeling approaches build components for feature selection directly into the architecture. For time series forecasting specifically, they are based on explicitly quantifying time-dependent variable contributions. For example, Interpretable Multi-Variable LSTMs [27] partitions the hidden state such that each variable contributes uniquely to its own memory segment, and weights memory segments to determine variable contributions. Methods combining temporal importance and variable selection have also been considered in [24], which computes a single contribution coefficient based on attention weights from each. However, in addition to the shortcoming of modelling only one-step-ahead forecasts, existing methods also focus on *instance-specific* (i.e. sample-specific) interpretations of attention weights – without providing insights into global temporal dynamics. In contrast, the use cases in Sec. 7 demonstrate that TFT is able to analyze global temporal relationships and allows users to interpret global behaviors of the model on the whole dataset – specifically in the identification of any persistent patterns (e.g. seasonality or lag effects) and regimes present.

3. Multi-horizon Forecasting

Let there be I unique entities in a given time series dataset – such as different stores in retail or patients in healthcare. Each entity i is associated with a set of static covariates $\boldsymbol{s}_i \in \mathbb{R}^{m_s}$, as well as inputs $\boldsymbol{\chi}_{i,t} \in \mathbb{R}^{m_\chi}$ and scalar targets $y_{i,t} \in \mathbb{R}$ at each time-step $t \in [0, T_i]$. Time-dependent input features are subdivided into two categories $\boldsymbol{\chi}_{i,t} = \left[\boldsymbol{z}_{i,t}^T, \boldsymbol{x}_{i,t}^T\right]^T$ – observed inputs $\boldsymbol{z}_{i,t} \in \mathbb{R}^{(m_z)}$ which can only be measured at each step and are unknown beforehand, and known inputs $\boldsymbol{x}_{i,t} \in \mathbb{R}^{m_x}$ which can be predetermined (e.g. the day-of-week at time t).

In many scenarios, the provision for prediction intervals can be useful for optimizing decisions and risk management by yielding an indication of likely best and worst-case values that the target can take. As such, we adopt quantile regression to our multi-horizon forecasting setting (e.g. outputting the 10^{th}, 50^{th} and 90^{th} percentiles at each time step). Each quantile forecast takes the form:

$$\hat{y}_i(q, t, \tau) = f_q\left(\tau, y_{i,t-k:t}, \boldsymbol{z}_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t+\tau}, \boldsymbol{s}_i\right), \tag{1}$$

where $\hat{y}_{i,t+\tau}(q, t, \tau)$ is the predicted q^{th} sample quantile of the τ-step-ahead forecast at time t, and $f_q(.)$ is a prediction model. In line with other direct methods, we simultaneously output forecasts for τ_{max} time steps – i.e. $\tau \in \{1, \ldots, \tau_{max}\}$. We incorporate all past information within a finite look-back window k, using target and known inputs only up till and including the forecast start time t (i.e. $y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$) and known inputs across the entire

range (i.e. $\boldsymbol{x}_{i,t-k:t+\tau} = \{\boldsymbol{x}_{i,t-k}, \ldots, \boldsymbol{x}_{i,t}, \ldots, \boldsymbol{x}_{i,t+\tau}\}$). [2]

4. Model Architecture

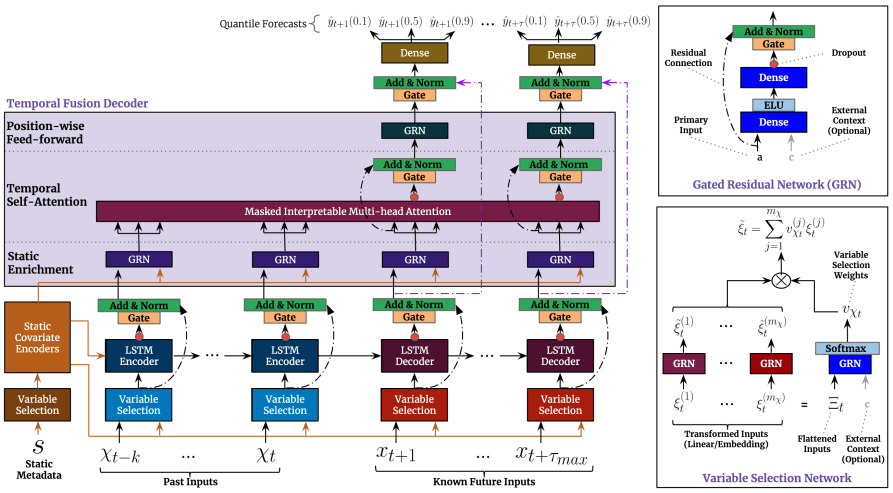


Figure 2: TFT architecture. TFT inputs static metadata, time-varying past inputs and time-varying a priori known future inputs. Variable Selection is used for judicious selection of the most salient features based on the input. Gated Residual Network blocks enable efficient information flow with skip connections and gating layers. Time-dependent processing is based on LSTMs for local processing, and multi-head attention for integrating information from any time step.

We design TFT to use canonical components to efficiently build feature representations for each input type (i.e. static, known, observed inputs) for high forecasting performance on a wide range of problems. The major constituents of TFT are:

1. **Gating mechanisms** to skip over any unused components of the architecture, providing adaptive depth and network complexity to accommodate a wide range of datasets and scenarios.
2. **Variable selection networks** to select relevant input variables at each time step.
3. **Static covariate encoders** to integrate static features into the network, through encoding of context vectors to condition temporal dynamics.
4. **Temporal processing** to learn both long- and short-term temporal relationships from both observed and known time-varying inputs. A sequence-to-sequence layer is employed for local processing, whereas long-term dependencies are captured using a novel interpretable multi-head attention block.

[2] For notation simplicity, we omit the subscript i unless explicitly required.

5. **Prediction intervals** via quantile forecasts to determine the range of likely target values at each prediction horizon.

Fig. 2 shows the high level architecture of Temporal Fusion Transformer (TFT), with individual components described in detail in the subsequent sections.

4.1. Gating Mechanisms

The precise relationship between exogenous inputs and targets is often unknown in advance, making it difficult to anticipate which variables are relevant. Moreover, it is difficult to determine the extent of required non-linear processing, and there may be instances where simpler models can be beneficial – e.g. when datasets are small or noisy. With the motivation of giving the model the flexibility to apply non-linear processing only where needed, we propose Gated Residual Network (GRN) as shown in in Fig. 2 as a building block of TFT. The GRN takes in a primary input $\boldsymbol{a}$ and an optional context vector $\boldsymbol{c}$ and yields:

$$\text{GRN}_{\omega}\left(\boldsymbol{a}, \boldsymbol{c}\right) = \text{LayerNorm}\left(\boldsymbol{a} + \text{GLU}_{\omega}(\boldsymbol{\eta}_1)\right), \tag{2}$$

$$\boldsymbol{\eta}_1 = \boldsymbol{W}_{1,\omega}\ \boldsymbol{\eta}_2 + \boldsymbol{b}_{1,\omega}, \tag{3}$$

$$\boldsymbol{\eta}_2 = \text{ELU}\left(\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega}\right), \tag{4}$$

where ELU is the Exponential Linear Unit activation function [28], $\boldsymbol{\eta}_1 \in \mathbb{R}^{d_{model}}, \boldsymbol{\eta}_2 \in \mathbb{R}^{d_{model}}$ are intermediate layers, LayerNorm is standard layer normalization of [29], and ω is an index to denote weight sharing. When $\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega} >> 0$, the ELU activation would act as an identity function and when $\boldsymbol{W}_{2,\omega}\ \boldsymbol{a} + \boldsymbol{W}_{3,\omega}\ \boldsymbol{c} + \boldsymbol{b}_{2,\omega} << 0$, the ELU activation would generate a constant output, resulting in linear layer behavior. We use component gating layers based on Gated Linear Units (GLUs) [30] to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset. Letting $\boldsymbol{\gamma} \in \mathbb{R}^{d_{model}}$ be the input, the GLU then takes the form:

$$\text{GLU}_{\omega}(\boldsymbol{\gamma}) = \sigma(\boldsymbol{W}_{4,\omega}\ \boldsymbol{\gamma} + \boldsymbol{b}_{4,\omega}) \odot (\boldsymbol{W}_{5,\omega}\ \boldsymbol{\gamma} + \boldsymbol{b}_{5,\omega}), \tag{5}$$

where $\sigma(.)$ is the sigmoid activation function, $\boldsymbol{W}_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}$, $\boldsymbol{b}_{(.)} \in \mathbb{R}^{d_{model}}$ are the weights and biases, $\odot$ is the element-wise Hadamard product, and d_{model} is the hidden state size (common across TFT). GLU allows TFT to control the extent to which the GRN contributes to the original input $\boldsymbol{a}$ – potentially skipping over the layer entirely if necessary as the GLU outputs could be all close to 0 in order to surpress the nonlinear contribution. For instances without a context vector, the GRN simply treats the contex input as zero – i.e. $\boldsymbol{c} = 0$ in Eq. (4). During training, dropout is applied before the gating layer and layer normalization – i.e. to $\boldsymbol{\eta}_1$ in Eq. (3).

4.2. Variable Selection Networks

While multiple variables may be available, their relevance and specific contribution to the output are typically unknown. TFT is designed to provide

instance-wise variable selection through the use of variable selection networks applied to both static covariates and time-dependent covariates. Beyond providing insights into which variables are most significant for the prediction problem, variable selection also allows TFT to remove any unnecessary noisy inputs which could negatively impact performance. Most real-world time series datasets contain features with less predictive content, thus variable selection can greatly help model performance via utilization of learning capacity only on the most salient ones.

We use entity embeddings [31] for categorical variables as feature representations, and linear transformations for continuous variables – transforming each input variable into a (d_{model})-dimensional vector which matches the dimensions in subsequent layers for skip connections. All static, past and future inputs make use of separate variable selection networks (as denoted by different colors in Fig. 2). Without loss of generality, we present the variable selection network for past inputs – noting that those for other inputs take the same form.

Let $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{model}}$ denote the transformed input of the j-th variable at time t, with $\boldsymbol{\Xi}_t = \left[\boldsymbol{\xi}_t^{(1)^T}, \ldots, \boldsymbol{\xi}_t^{(m_\chi)^T}\right]^T$ being the flattened vector of all past inputs at time t. Variable selection weights are generated by feeding both $\boldsymbol{\Xi}_t$ and an external context vector $\boldsymbol{c}_s$ through a GRN, followed by a Softmax layer:

$$\boldsymbol{v}_{\chi_t} = \text{Softmax}\left(\text{GRN}_{v_\chi}(\boldsymbol{\Xi}_t, \boldsymbol{c}_s)\right), \tag{6}$$

where $\boldsymbol{v}_{\chi_t} \in \mathbb{R}^{m_\chi}$ is a vector of variable selection weights, and $\boldsymbol{c}_s$ is obtained from a static covariate encoder (see Sec. 4.3). For static variables, we note that the context vector $\boldsymbol{c}_s$ is omitted – given that it already has access to static information.

At each time step, an additional layer of non-linear processing is employed by feeding each $\boldsymbol{\xi}_t^{(j)}$ through its own GRN:

$$\tilde{\boldsymbol{\xi}}_t^{(j)} = \text{GRN}_{\tilde{\xi}(j)}\left(\boldsymbol{\xi}_t^{(j)}\right), \tag{7}$$

where $\tilde{\boldsymbol{\xi}}_t^{(j)}$ is the processed feature vector for variable j. We note that each variable has its own $\text{GRN}_{\xi(j)}$, with *weights shared across all time steps* t. Processed features are then weighted by their variable selection weights and combined:

$$\tilde{\boldsymbol{\xi}}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\boldsymbol{\xi}}_t^{(j)}, \tag{8}$$

where $v_{\chi_t}^{(j)}$ is the j-th element of vector $\boldsymbol{v}_{\chi_t}$.

4.3. Static Covariate Encoders

In contrast with other time series forecasting architectures, the TFT is carefully designed to integrate information from static metadata, using separate GRN encoders to produce four different context vectors, $\boldsymbol{c}_s$, $\boldsymbol{c}_e$, $\boldsymbol{c}_c$, and $\boldsymbol{c}_h$. These contect vectors are wired into various locations in the temporal fusion

decoder (Sec. 4.5) where static variables play an important role in processing. Specifically, this includes contexts for (1) temporal variable selection ($\boldsymbol{c}_s$), (2) local processing of temporal features ($\boldsymbol{c}_c, \boldsymbol{c}_h$), and (3) enriching of temporal features with static information ($\boldsymbol{c}_e$). As an example, taking $\boldsymbol{\zeta}$ to be the output of the static variable selection network, contexts for temporal variable selection would be encoded according to $\boldsymbol{c}_s = GRN_{c_s}(\boldsymbol{\zeta})$.

4.4. Interpretable Multi-Head Attention

The TFT employs a self-attention mechanism to learn long-term relationships across different time steps, which we modify from multi-head attention in transformer-based architectures [17, 12] to enhance explainability. In general, attention mechanisms scale values $\boldsymbol{V} \in \mathbb{R}^{N \times d_V}$ based on relationships between keys $\boldsymbol{K} \in \mathbb{R}^{N \times d_{attn}}$ and queries $\boldsymbol{Q} \in \mathbb{R}^{N \times d_{attn}}$ as below:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = A(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}, \tag{9}$$

where $A()$ is a normalization function. A common choice is scaled dot-product attention [17]:

$$A(\boldsymbol{Q}, \boldsymbol{K}) = \text{Softmax}(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d_{attn}}). \tag{10}$$

To improve the learning capacity of the standard attention mechanism, multi-head attention is proposed in [17], employing different heads for different representation subspaces:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_{m_H}]\, \boldsymbol{W}_H, \tag{11}$$

$$\boldsymbol{H}_h = \text{Attention}(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}, \boldsymbol{V}\, \boldsymbol{W}_V^{(h)}), \tag{12}$$

where $\boldsymbol{W}_K^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $\boldsymbol{W}_V^{(h)} \in \mathbb{R}^{d_{model} \times d_V}$ are head-specific weights for keys, queries and values, and $\boldsymbol{W}_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{model}}$ linearly combines outputs concatenated from all heads $\boldsymbol{H}_h$.

Given that different values are used in each head, attention weights alone would not be indicative of a particular feature's importance. As such, we modify multi-head attention to share values in each head, and employ additive aggregation of all heads:

$$\text{InterpretableMultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \tilde{\boldsymbol{H}}\, \boldsymbol{W}_H, \tag{13}$$

$$\tilde{\boldsymbol{H}} = \tilde{A}(\boldsymbol{Q}, \boldsymbol{K})\, \boldsymbol{V}\, \boldsymbol{W}_V, \tag{14}$$

$$= \left\{1/H \sum_{h=1}^{m_H} A\left(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}\right)\right\} \boldsymbol{V}\, \boldsymbol{W}_V, \tag{15}$$

$$= 1/H \sum_{h=1}^{m_H} \text{Attention}(\boldsymbol{Q}\, \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\, \boldsymbol{W}_K^{(h)}, \boldsymbol{V}\, \boldsymbol{W}_V), \tag{16}$$

where $\boldsymbol{W}_V \in \mathbb{R}^{d_{model} \times d_V}$ are value weights shared across all heads, and $\boldsymbol{W}_H \in \mathbb{R}^{d_{attn} \times d_{model}}$ is used for final linear mapping. From Eq. (15), we see that each

head can learn different temporal patterns, while attending to a common set of input features – which can be interpreted as a simple ensemble over attention weights into combined matrix $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K})$ in Eq. (14). Compared to $A(\boldsymbol{Q}, \boldsymbol{K})$ in Eq. (10), $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K})$ yields an increased representation capacity in an efficient way.

4.5. Temporal Fusion Decoder

The temporal fusion decoder uses the series of layers described below to learn temporal relationships present in the dataset:

4.5.1. Locality Enhancement with Sequence-to-Sequence Layer

In time series data, points of significance are often identified in relation to their surrounding values – such as anomalies, change-points or cyclical patterns. Leveraging local context, through the construction of features that utilize pattern information on top of point-wise values, can thus lead to performance improvements in attention-based architectures. For instance, [12] adopts a single convolutional layer for locality enhancement – extracting local patterns using the same filter across all time. However, this might not be suitable for cases when observed inputs exist, due to the differing number of past and future inputs. As such, we propose the application of a sequence-to-sequence model to naturally handle these differences – feeding $\tilde{\boldsymbol{\xi}}_{t-k:t}$ into the encoder and $\tilde{\boldsymbol{\xi}}_{t+1:t+\tau_{max}}$ into the decoder. This then generates a set of uniform temporal features which serve as inputs into the temporal fusion decoder itself – denoted by $\boldsymbol{\phi}(t, n) \in \{\boldsymbol{\phi}(t, -k), \ldots, \boldsymbol{\phi}(t, \tau_{max})\}$ with n being a position index. For comparability with commonly-used sequence-to-sequence baselines, we consider the use of an LSTM encoder-decoder – although other models can potentially be adopted as well. This also serves as a replacement for standard positional encoding, providing an appropriate inductive bias for the time ordering of the inputs. Moreover, to allow static metadata to influence local processing, we use the $\boldsymbol{c}_c, \boldsymbol{c}_h$ context vectors from the static covariate encoders to initialize the cell state and hidden state respectively for the first LSTM in the layer. We also employ a gated skip connection over this layer:

$$\tilde{\boldsymbol{\phi}}(t, n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\xi}}_{t+n} + \text{GLU}_{\tilde{\phi}}(\boldsymbol{\phi}(t, n))\right), \tag{17}$$

where $n \in [-k, \tau_{max}]$ is a position index.

4.5.2. Static Enrichment Layer

As static covariates often have a significant influence on the temporal dynamics (e.g. genetic information on disease risk), we introduce a static enrichment layer that enhances temporal features with static metadata. For a given position index n, static enrichment takes the form:

$$\boldsymbol{\theta}(t, n) = \text{GRN}_{\theta}\left(\tilde{\boldsymbol{\phi}}(t, n), \boldsymbol{c}_e\right), \tag{18}$$

where the weights of GRN_{ϕ} are shared across the entire layer, and $\boldsymbol{c}_e$ is a context vector from a static covariate encoder.

4.5.3. Temporal Self-Attention Layer

Following static enrichment, we next apply self-attention. All static-enriched temporal features are first grouped into a single matrix – i.e. $\boldsymbol{\Theta}(t) = [\boldsymbol{\theta}(t, -k), \ldots, \boldsymbol{\theta}(t, \tau)]^T$ – and interpretable multi-head attention (see Sec. 4.4) is applied at each forecast time (with $N = \tau_{max} + k + 1$):

$$\boldsymbol{B}(t) = \text{InterpretableMultiHead}(\boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t)), \tag{19}$$

to yield $\boldsymbol{B}(t) = [\boldsymbol{\beta}(t, -k), \ldots, \boldsymbol{\beta}(t, \tau_{max})]$. $d_V = d_{attn} = d_{model}/m_H$ are chosen, where m_H is the number of heads. Decoder masking [17, 12] is applied to the multi-head attention layer to ensure that each temporal dimension can only attend to features preceding it. Besides preserving causal information flow via masking, the self-attention layer allows TFT to pick up long-range dependencies that may be challenging for RNN-based architectures to learn. Following the self-attention layer, an additional gating layer is also applied to facilitate training:

$$\boldsymbol{\delta}(t, n) = \text{LayerNorm}(\boldsymbol{\theta}(t, n) + \text{GLU}_{\delta}(\boldsymbol{\beta}(t, n))). \tag{20}$$

4.5.4. Position-wise Feed-forward Layer

We apply an additional non-linear processing to the outputs of the self-attention layer. Similar to the static enrichment layer, this makes use of GRNs:

$$\boldsymbol{\psi}(t, n) = \text{GRN}_{\psi}\left(\boldsymbol{\delta}(t, n)\right), \tag{21}$$

where the weights of GRN_{ψ} are shared across the entire layer. As per Fig. 2, we also apply a gated residual connection which skips over the entire transformer block, providing a direct path to the sequence-to-sequence layer – yielding a simpler model if additional complexity is not required, as shown below:

$$\tilde{\boldsymbol{\psi}}(t, n) = \text{LayerNorm}\left(\tilde{\boldsymbol{\phi}}(t, n) + \text{GLU}_{\tilde{\psi}}(\boldsymbol{\psi}(t, n))\right), \tag{22}$$

4.6. Quantile Outputs

In line with previous work [10], TFT also generates prediction intervals on top of point forecasts. This is achieved by the simultaneous prediction of various percentiles (e.g. 10^{th}, 50^{th} and 90^{th}) at each time step. Quantile forecasts are generated using linear transformation of the output from the temporal fusion decoder:

$$\hat{y}(q, t, \tau) = \boldsymbol{W}_q\, \tilde{\boldsymbol{\psi}}(t, \tau) + b_q, \tag{23}$$

where $\boldsymbol{W}_q \in \mathbb{R}^{1 \times d}, b_q \in \mathbb{R}$ are linear coefficients for the specified quantile q. We note that forecasts are only generated for horizons in the future – i.e. $\tau \in \{1, \ldots, \tau_{max}\}$.

5. Loss Functions

TFT is trained by jointly minimizing the quantile loss [10], summed across all quantile outputs:

$$\mathcal{L}(\Omega, \boldsymbol{W}) = \sum_{y_t \in \Omega} \sum_{q \in \mathcal{Q}} \sum_{\tau=1}^{\tau_{max}} \frac{QL\left(y_t,\ \hat{y}(q, t-\tau, \tau),\ q\right)}{M \tau_{max}} \tag{24}$$

$$QL(y,\ \hat{y},\ q) = q(y - \hat{y})_+ + (1-q)(\hat{y} - y)_+, \tag{25}$$

where Ω is the domain of training data containing M samples, $\boldsymbol{W}$ represents the weights of TFT, $\mathcal{Q}$ is the set of output quantiles (we use $\mathcal{Q} = \{0.1, 0.5, 0.9\}$ in our experiments, and $(.)_+ = \max(0, .)$. For out-of-sample testing, we evaluate the normalized quantile losses across the entire forecasting horizon – focusing on P50 and P90 risk for consistency with previous work [9, 6, 12]:

$$q\text{-Risk} = \frac{2 \sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{max}} QL\left(y_t,\ \hat{y}(q, t-\tau, \tau),\ q\right)}{\sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{max}} |y_t|}, \tag{26}$$

where $\tilde{\Omega}$ is the domain of test samples. Full details on hyperparameter optimization and training can be found in Appendix A.

6. Performance Evaluation

6.1. Datasets

We choose datasets to reflect commonly observed characteristics across a wide range of challenging multi-horizon forecasting problems. To establish a baseline and position with respect to prior academic work, we first evaluate performance on the Electricity and Traffic datasets used in [9, 6, 12] – which focus on simpler univariate time series containing known inputs only alongside the target. Next, the Retail dataset helps us benchmark the model using the full range of complex inputs observed in multi-horizon prediction applications (see Sec. 3) – including rich static metadata and observed time-varying inputs. Finally, to evaluate robustness to over-fitting on smaller noisy datasets, we consider the financial application of volatility forecasting – using a dataset much smaller than others. Broad descriptions of each dataset can be found below:

- **Electricity:** The UCI Electricity Load Diagrams Dataset, containing the electricity consumption of 370 customers – aggregated on an hourly level as in [32]. In accordance with [9], we use the past week (i.e. 168 hours) to forecast over the next 24 hours.
- **Traffic:** The UCI PEM-SF Traffic Dataset describes the occupancy rate (with $y_t \in [0, 1]$) of 440 SF Bay Area freeways – as in [32]. It is also aggregated on an hourly level as per the electricity dataset, with the same look back window and forecast horizon.

- **Retail:** Favorita Grocery Sales Dataset from the Kaggle competition [33], that combines metadata for different products and the stores, along with other exogenous time-varying inputs sampled at the daily level. We forecast log product sales 30 days into the future, using 90 days of past information.
- **Volatility (or Vol.):** The OMI realized library [34] contains daily realized volatility values of 31 stock indices computed from intraday data, along with their daily returns. For our experiments, we consider forecasts over the next week (i.e. 5 business days) using information over the past year (i.e. 252 business days).

6.2. Training Procedure

For each dataset, we partition all time series into 3 parts – a training set for learning, a validation set for hyperparameter tuning, and a hold-out test set for performance evaluation. Hyperparameter optimization is conducted via random search, using 240 iterations for Volatility, and 60 iterations for others. Full search ranges for all hyperparameters are below, with datasets and optimal model parameters listed in Table 1.

- **State size** – 10, 20, 40, 80, 160, 240, 320
- **Dropout rate** – 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9
- **Minibatch size** – 64, 128, 256
- **Learning rate** – 0.0001, 0.001, 0.01
- **Max. gradient norm** – 0.01, 1.0, 100.0
- **Num. heads** – 1, 4

To preserve explainability, we adopt only a single interpretable multi-head attention layer. For ConvTrans [12], we use the same fixed stack size (3 layers) and number of heads (8 heads) as in [12]. We keep the same attention model, and treat kernel sizes for the convolutional processing layer as a hyperparameter ($\in \{1, 3, 6, 9\}$) – as optimal kernel sizes are observed to be dataset dependent [12]. An open-source implementation of the TFT on these datasets can be found on GitHub[3] for full reproducibility.

6.3. Computational Cost

Across all datasets, each TFT model was also trained on a single GPU, and can be deployed without the need for extensive computing resources. For instance, using a NVIDIA Tesla V100 GPU, our optimal TFT model (for Electricity dataset) takes just slightly over 6 hours to train (each epoch being roughly 52 mins). The batched inference on the entire validation dataset (consisting 50,000 samples) takes 8 minutes. TFT training and inference times can be further reduced with hardware-specific optimizations.

Table 1: Information on dataset and optimal TFT configuration.

	Electricity	**Traffic**	**Retail**	**Vol.**
Dataset Details				
Target Type	$\mathbb{R}$	[0, 1]	$\mathbb{R}$	$\mathbb{R}$
Number of Entities	370	440	130k	41
Number of Samples	500k	500k	500k	$\sim$100k
Network Parameters				
k	168	168	90	252
τ_{max}	24	24	30	5
Dropout Rate	0.1	0.3	0.1	0.3
State Size	160	320	240	160
Number of Heads	4	4	4	1
Training Parameters				
Minibatch Size	64	128	128	64
Learning Rate	0.001	0.001	0.001	0.01
Max Gradient Norm	0.01	100	100	0.01

6.4. Benchmarks

We extensively compare TFT to a wide range of models for multi-horizon forecasting, based on the categories described in Sec. 2. Hyperparameter optimization is conducted using random search over a pre-defined search space, using the same number of iterations across all benchmarks for a give dataset. Additional details are included in Appendix A.

Direct methods: As TFT falls within this class of multi-horizon models, we primarily focus comparisons on deep learning models which directly generate prediction at future horizons, including: 1) simple sequence-to-sequence models with global contexts (Seq2Seq), and 2) the Multi-horizon Quantile Recurrent Forecaster (MQRNN) [10].

Iterative methods: To position with respect to the rich body of work on iterative models, we evaluate TFT using the same setup as [9] for the Electricity and Traffic datasets. This extends the results from [12] for 1) DeepAR [9], 2) DSSM [6], and 3) the Transformer-based architecture of [12] with local convolutional processing – which refer to as ConvTrans. For more complex datasets, we focus on the ConvTrans model given its strong outperformance over other iterative models in prior work, and DeepAR due to its popularity among practitioners. As models in this category require knowledge of all inputs in the future to generate predictions, we accommodate this for complex datasets by imputing unknown inputs with their last available value.

For simpler univariate datasets, we note that the results for ARIMA, ETS, TRMF, DeepAR, DSSM and ConvTrans have been reproduced from [12] in

[3]URL: `https://github.com/google-research/google-research/tree/master/tft`

Table 2: P50 and P90 quantile losses on a range of real-world datasets. Percentages in brackets reflect the increase in quantile loss versus TFT (lower q-Risk better), with TFT outperforming competing methods across all experiments, improving on the next best alternative method (underlined) between 3% and 26%.

	ARIMA	**ETS**	**TRMF**	**DeepAR**	**DSSM**
Electricity	0.154 (*+180%*)	0.102 (*+85%*)	0.084 (*+53%*)	0.075 (*+36%*)	0.083 (*+51%*)
Traffic	0.223 (*+135%*)	0.236 (*+148%*)	0.186 (*+96%*)	0.161 (*+69%*)	0.167 (*+76%*)
	ConvTrans	**Seq2Seq**	**MQRNN**	**TFT**	
Electricity	0.059 (*+7%*)	0.067 (*+22%*)	0.077 (*+40%*)	**0.055***	
Traffic	0.122 (*+28%*)	0.105 (*+11%*)	0.117 (*+23%*)	**0.095***	

(a) P50 losses on simpler univariate datasets.

	ARIMA	**ETS**	**TRMF**	**DeepAR**	**DSSM**
Electricity	0.102 (*+278%*)	0.077 (*+185%*)	-	0.040 (*+48%*)	0.056 (*+107%*)
Traffic	0.137 (*+94%*)	0.148 (*+110%*)	-	0.099 (*+40%*)	0.113 (*+60%*)
	ConvTrans	**Seq2Seq**	**MQRNN**	**TFT**	
Electricity	0.034 (*+26%*)	0.036 (*+33%*)	0.036 (*+33%*)	**0.027***	
Traffic	0.081 (*+15%*)	0.075 (*+6%*)	0.082 (*+16%*)	**0.070***	

(b) P90 losses on simpler univariate datasets.

	DeepAR	**CovTrans**	**Seq2Seq**	**MQRNN**	**TFT**
Vol.	0.050 (*+28%*)	0.047 (*+20%*)	0.042 (*+7%*)	0.042 (*+7%*)	**0.039***
Retail	0.574 (*+62%*)	0.429 (*+21%*)	0.411 (*+16%*)	0.379 (*+7%*)	**0.354***

(c) P50 losses on datasets with rich static or observed inputs.

	DeepAR	**CovTrans**	**Seq2Seq**	**MQRNN**	**TFT**
Vol.	0.024 (*+21%*)	0.024 (*+22%*)	0.021 (*+8%*)	0.021 (*+9%*)	**0.020***
Retail	0.230 (*+56%*)	0.192 (*+30%*)	0.157 (*+7%*)	0.152 (*+3%*)	**0.147***

(d) P90 losses on datasets with rich static or observed inputs.

Table 2 for consistency.

6.5. Results and Discussion

Table 2 shows that TFT significantly outperforms all benchmarks over the variety of datasets described in Sec. 6.1. For median forecasts, TFT yields 7% lower P50 and 9% lower P90 losses on average compared to the next best model – demonstrating the benefits of explicitly aligning the architecture with the general multi-horizon forecasting problem.

Comparing direct and iterative models, we observe the importance of accounting for the observed inputs – noting the poorer results of ConvTrans on complex datasets where observed input imputation is required (i.e. Volatility and Retail). Furthermore, the benefits of quantile regression are also observed when targets are not captured well by Gaussian distributions with direct models outperforming in those scenarios. This can be seen, for example, from the Traffic dataset where target distribution is significantly skewed – with more than 90% of occupancy rates falling between 0 and 0.1, and the remainder distributed evenly until 1.0.

6.6. Ablation Analysis

To quantify the benefits of each of our proposed architectural contribution, we perform an extensive ablation analysis – removing each component from the network as below, and quantifying the percentage increase in loss versus the original architecture:

- **Gating layers:** We ablate by replacing each GLU layer (Eq. (5)) with a simple linear layer followed by ELU.
- **Static covariate encoders:** We ablate by setting all context vectors to zero – i.e. $\boldsymbol{c}_s{=}\boldsymbol{c}_e{=}\boldsymbol{c}_c{=}\boldsymbol{c}_h{=}\boldsymbol{0}$ – and concatenating all transformed static inputs to all time-dependent past and future inputs.
- **Instance-wise variable selection networks:** We ablate by replacing the softmax outputs of Eq. 6 with trainable coefficients, and removing the networks generating the variable selection weights. We retain, however, the variable-wise GRNs (see Eq. (7)), maintaining a similar amount of non-linear processing.
- **Self-attention layers:** We ablate by replacing the attention matrix of the interpretable multi-head attention layer (Eq. 14) with a matrix of trainable parameters $\boldsymbol{W}_A$ – i.e. $\tilde{A}(\boldsymbol{Q}, \boldsymbol{K}) = \boldsymbol{W}_A$, where $\boldsymbol{W}_A \in \mathbb{R}^{N \times N}$. This prevents TFT from attending to different input features at different times, helping evaluation of the importance of instance-wise attention weights.
- **Sequence-to-sequence layers for local processing:** We ablate by replacing the sequence-to-sequence layer of Sec. 4.5.1 with standard positional encoding used in [17].

Ablated networks are trained across for each dataset using the hyperparameters of Table 1. Fig. 3 shows that the effects on both P50 and P90 losses are similar across all datasets, with all components contributing to performance improvements on the whole.

In general, the components responsible for capturing temporal relationships, local processing and self-attention layers, have the largest impact on performance, with P90 loss increases of $> 6\%$ on average and $> 20\%$ on select datasets when ablated. The diversity across time series datasets can also be seen from the differences in the ablation impact of the respective temporal components. Concretely, while local processing is critical in Traffic, Retail and Volatility, lower post-ablation P50 losses indicate that it can be detrimental in Electricity – with the self-attention layer playing a more vital role. A possible explanation is that persistent daily seasonality appears to dominate other temporal relationships in the Electricity dataset. For this dataset, Table B.4 of Appendix B also shows that the hour-of-day has the largest variable importance score across all temporal inputs, exceeding even the target (i.e. Power Usage) itself. In contrast to other dataset where past target observations are more significant (e.g. Traffic), direct attention to previous days seem to help learning daily seasonal patterns in Electricity – with local processing between adjacent time steps being less necessary. We can account for this by treating the sequence-to-sequence architecture in the temporal fusion decoder as a hyperparameter to tune, including an option for simple positional encoding without any local processing.

Static covariate encoders and instance-wise variable selection have the next largest impact – increasing P90 losses by more than 2.6% and 4.1% on average. The biggest benefits of these are observed for electricity dataset, where some of the input features get very low importance.

Finally, gating layer ablation also shows increases in P90 losses, with a 1.9% increase on average. This is the most significant on the volatility (with a 4.1% P90 loss increase), underlying the benefit of component gating for smaller and noisier datasets.

7. Interpretability Use Cases

Having established the performance benefits of our model, we next demonstrate how our model design allows for analysis of its individual components to interpret the general relationships it has learned. We demonstrate three interpretability use cases: (1) examining the importance of each input variable in prediction, (2) visualizing persistent temporal patterns, and (3) identifying any regimes or events that lead to significant changes in temporal dynamics. In contrast to other examples of attention-based interpretability [25, 12, 7] which zoom in on interesting but instance-specific examples, our methods focus on ways to aggregate the patterns across the entire dataset – extracting generalizable insights about temporal dynamics.

7.1. Analyzing Variable Importance

We first quantify variable importance by analyzing the variable selection weights described in Sec. 4.2. Concretely, we aggregate selection weights (i.e. $v_{\chi_t}^{(j)}$ in Eq. (8)) for each variable across our entire test set, recording the 10^{th}, 50^{th} and 90^{th} percentiles of each sampling distribution. As the Retail dataset

Table 3: Variable importance for the Retail dataset. The 10^{th}, 50^{th} and 90^{th} percentiles of the variable selection weights are shown, with values larger than 0.1 highlighted in purple. For static covariates, the largest weights are attributed to variables which uniquely identify different entities (i.e. item number and store number). For past inputs, past values of the target (i.e. log sales) are critical as expected, as forecasts are extrapolations of past observations. For future inputs, promotion periods and national holidays have the greatest influence on sales forecasts, in line with periods of increased customer spending.

	10%	50%	90%
Item Num	**0.198**	**0.230**	**0.251**
Store Num	**0.152**	**0.161**	**0.170**
City	0.094	0.100	0.124
State	0.049	0.060	0.083
Type	0.005	0.006	0.008
Cluster	**0.108**	**0.122**	**0.133**
Family	0.063	0.075	0.079
Class	**0.148**	**0.156**	**0.163**
Perishable	0.084	0.085	0.088

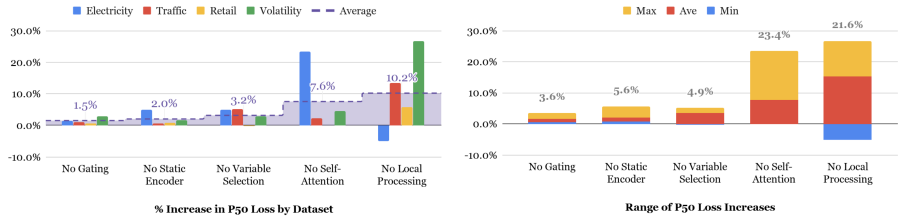
(a) Static Covariates

	10%	50%	90%
Transactions	0.029	0.033	0.037
Oil	0.062	0.081	0.105
On-promotion	0.072	0.075	0.078
Day of Week	0.007	0.007	0.008
Day of Month	0.083	0.089	0.096
Month	**0.109**	**0.122**	**0.136**
National Hol	**0.131**	**0.138**	**0.145**
Regional Hol	0.011	0.014	0.018
Local Hol	0.056	0.068	0.072
Open	0.027	0.044	0.067
Log Sales	**0.304**	**0.324**	**0.353**

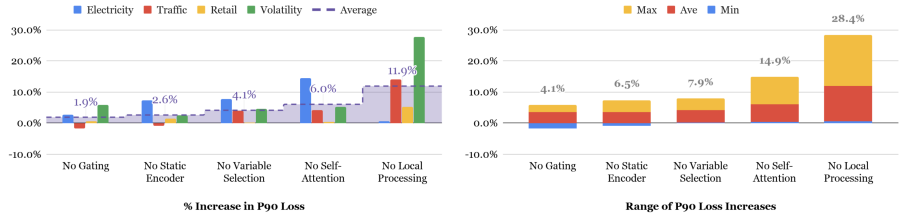
(b) Past Inputs

	10%	50%	90%
On-promotion	**0.155**	**0.170**	**0.182**
Day of Week	0.029	0.065	0.089
Day of Month	**0.056**	**0.116**	**0.138**
Month	**0.111**	**0.155**	**0.240**
National Hol	**0.145**	**0.220**	**0.242**
Regional Hol	0.012	0.014	0.060
Local Hol	**0.116**	**0.151**	**0.239**
Open	0.088	0.095	0.097

(c) Future Inputs



(a) Changes in P50 losses across ablation tests



(b) Changes in P90 losses across ablation tests

Figure 3: Results of ablation analysis. Both a) and b) show the impact of ablation on the P50 and P90 losses respectively. Results per dataset shown on the left, and the range across datasets shown on the right. While the precise importance of each is dataset-specific, all components contribute significantly on the whole – with the maximum percentage increase over all datasets ranging from 3.6% to 23.4% for P50 losses, and similarly from 4.1% to 28.4% for P90 losses.

contains the full set of available input types (i.e. static metadata, known inputs, observed inputs and the target), we present the results for its variable importance analysis in Table 3. We also note similar findings in other datasets, which are documented in Appendix B.1 for completeness. On the whole, the results show that the TFT extracts only a subset of key inputs that intuitively play a significant role in predictions. The analysis of persistent temporal patterns is often key to understanding the time-dependent relationships present in a given dataset. For instance, lag models are frequently adopted to study length of time required for an intervention to take effect [35] – such as the impact of a government's increase in public expenditure on the resultant growth in Gross National Product [36]. Seasonality models are also commonly used in econometrics to identify periodic patterns in a target-of-interest [37] and measure the length of each cycle. From a practical standpoint, model builders can use these insights to further improve the forecasting model – for instance by increasing the receptive field to incorporate more history if attention peaks are observed at the start of the lookback window, or by engineering features to directly incorporate seasonal effects. As such, using the attention weights present in the self-attention layer of the temporal fusion decoder, we present a method to identify similar persistent patterns – by measuring the contributions of features at fixed lags in the past on forecasts at various horizons. Combining Eq. (14) and (19), we see that the self-attention layer contains a matrix of attention weights at each forecast time t – i.e. $\tilde{A}(\boldsymbol{\phi}(t), \boldsymbol{\phi}(t))$. Multi-head attention outputs at each forecast horizon τ

(i.e. $\boldsymbol{\beta}(t,\tau)$) can then be described as an attention-weighted sum of lower level features at each position n:

$$\boldsymbol{\beta}(t,\tau) = \sum_{n=-k}^{\tau_{max}} \alpha(t,n,\tau)\ \tilde{\boldsymbol{\theta}}(t,n), \tag{27}$$

where $\alpha(t,n,\tau)$ is the (τ,n)-th element of $\tilde{A}(\boldsymbol{\phi}(t),\boldsymbol{\phi}(t))$, and $\tilde{\boldsymbol{\theta}}(t,n)$ is a row of $\tilde{\boldsymbol{\Theta}}(t) = \boldsymbol{\Theta}(t)\boldsymbol{W}_V$. Due to decoder masking, we also note that $\alpha(t,i,j) = 0$, $\forall i > j$. For each forecast horizon τ, the importance of a previous time point $n < \tau$ can hence be determined by analyzing distributions of $\alpha(t,n,\tau)$ across all time steps and entities.

7.2. Visualizing Persistent Temporal Patterns

Attention weight patterns can be used to shed light on the most important past time steps that the TFT model bases its decisions on. In contrast to other traditional and machine learning time series methods, which rely on model-based specifications for seasonality and lag analysis, the TFT can learn such patterns from raw training data.

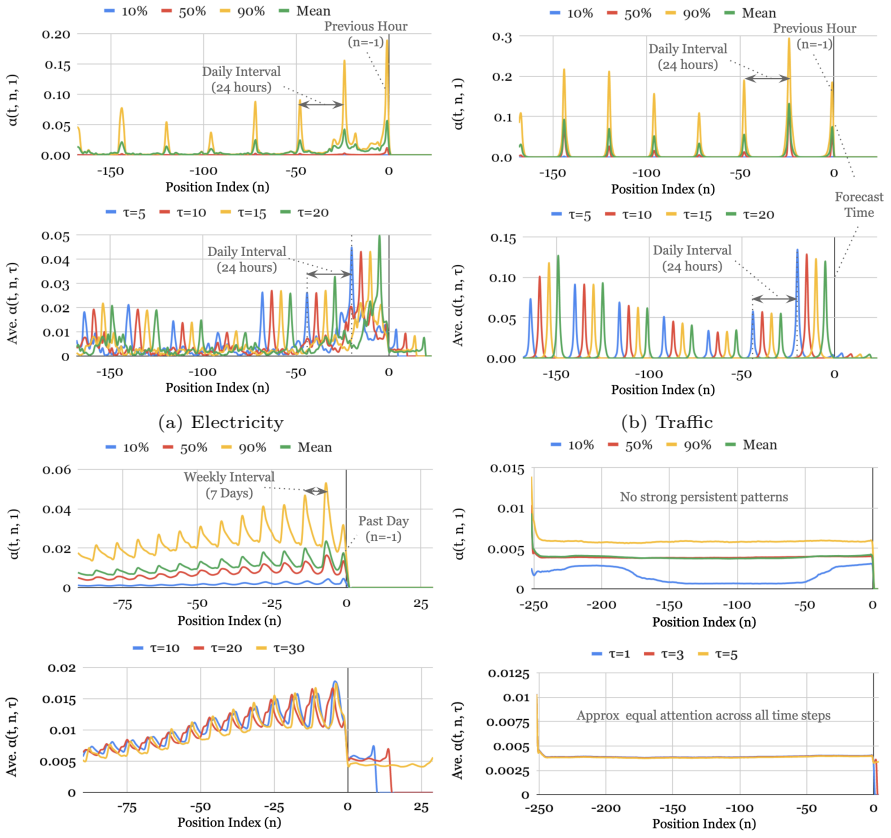
Fig. 4 shows the attention weight patterns across all our test datasets – with the upper graph plotting the mean along with the 10^{th}, 50^{th} and 90^{th} percentiles of the attention weights for one-step-ahead forecasts (i.e. $\alpha(t,1,\tau)$) over the test set, and the bottom graph plotting the average attention weights for various horizons (i.e. $\tau \in \{5,10,15,20\}$). We observe that the three datasets exhibit a seasonal pattern, with clear attention spikes at daily intervals observed for Electricity and Traffic, and a slightly weaker weekly patterns for Retail. For Retail, we also observe the decaying trend pattern, with the last few days dominating the importance.

No strong persistent patterns were observed for the Volatility – attention weights equally distributed across all positions on average. This resembles a moving average filter at the feature level, and – given the high degree of randomness associated with the volatility process – could be useful in extracting the trend over the entire period by smoothing out high-frequency noise.

TFT learns these persistent temporal patterns from the raw training data without any human hard-coding. Such capability is expected to be very useful in building trust with human experts via sanity-checking. Model developers can also use these towards model improvements, e.g. via specific feature engineering or data collection.

7.3. Identifying Regimes & Significant Events

Identifying sudden changes in temporal patterns can also be very useful, as temporary shifts can occur due to the presence of significant regimes or events. For instance, regime-switching behavior has been widely documented in financial markets [38], with returns characteristics – such as volatility – being observed to change abruptly between regimes. As such, identifying such regime changes provides strong insights into the underlying problem which is useful for identification of the significant events.



(a) Electricity (b) Traffic

(c) Retail (d) Volatility

Figure 4: Persistent temporal patterns across datasets. Clear seasonality observed for the Electricity, Traffic and Retail datasets, but no strong persistent patterns seen in Volatility dataset. Upper plot – percentiles of attention weights for one-step-ahead forecast. Lower plot – average attention weights for forecast at various horizons.

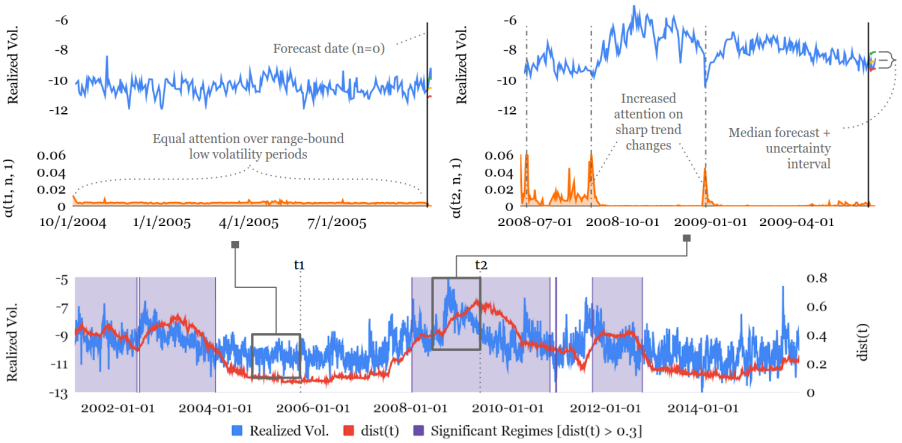


Figure 5: Regime identification for S&P 500 realized volatility. Significant deviations in attention patterns can be observed around periods of high volatility – corresponding to the peaks observed in dist(t). We use a threshold of dist$(t) > 0.3$ to denote significant regimes, as highlighted in purple. Focusing on periods around the 2008 financial crisis, the top right plot visualizes $\alpha(t, n, 1)$ midway through the significant regime, compared to the normal regime on the top left.

Firstly, for a given entity, we define the average attention pattern per forecast horizon as:

$$\bar{\alpha}(n, \tau) = \sum_{t=1}^{T} \alpha(t, j, \tau)/T, \tag{28}$$

and then construct $\bar{\boldsymbol{\alpha}}(\tau) = [\bar{\alpha}(-k, \tau), \ldots, \bar{\alpha}(\tau_{max}, \tau)]^T$. To compare similarities between attention weight vectors, we use the distance metric proposed by [39]:

$$\kappa(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{1 - \rho(p, q)}, \tag{29}$$

where $\rho(\boldsymbol{p}, \boldsymbol{q}) = \sum_j \sqrt{p_j q_j}$ is the Bhattacharya coefficient [40] measuring the overlap between discrete distributions – with p_j, q_j being elements of probability vectors $\boldsymbol{p}, \boldsymbol{q}$ respectively. For each entity, significant shifts in temporal dynamics are then measured using the distance between attention vectors at each point with the average pattern, aggregated for all horizons as below:

$$\text{dist}(t) = \sum_{\tau=1}^{\tau_{max}} \kappa\big(\bar{\boldsymbol{\alpha}}(\tau), \boldsymbol{\alpha}(t, \tau)\big)/\tau_{max}, \tag{30}$$

where $\boldsymbol{\alpha}(t, \tau) = [\alpha(t, -k, \tau), \ldots, \alpha(t, \tau_{max}, \tau)]^T$.

Using the volatility dataset, we attempt to analyse regimes by applying our distance metric to the attention patterns for the S&P 500 index over our training period (2001 to 2015). Plotting dist(t) against the target (i.e. log realized volatility) in the bottom chart of Fig. 5, significant deviations in attention patterns can be observed around periods of high volatility (e.g. the 2008 financial crisis) – corresponding to the peaks observed in dist(t). From the plots, we can

see that TFT appears to alter its behaviour between regimes – placing equal attention across past inputs when volatility is low, while attending more to sharp trend changes during high volatility periods – suggesting differences in temporal dynamics learned in each of these cases.

8. Conclusions

We introduce TFT, a novel attention-based deep learning model for interpretable high-performance multi-horizon forecasting. To handle static covariates, a priori known inputs, and observed inputs effectively across wide range of multi-horizon forecasting datasets, TFT uses specialized components. Specifically, these include: (1) sequence-to-sequence and attention based temporal processing components that capture time-varying relationships at different timescales, (2) static covariate encoders that allow the network to condition temporal forecasts on static metadata, (3) gating components that enable skipping over unnecessary parts of the network, (4) variable selection to pick relevant input features at each time step, and (5) quantile predictions to obtain output intervals across all prediction horizons. On a wide range of real-world tasks – on both simple datasets that contain only known inputs and complex datasets which encompass the full range of possible inputs – we show that TFT achieves state-of-the-art forecasting performance. Lastly, we investigate the general relationships learned by TFT through a series of interpretability use cases – proposing novel methods to use TFT to (i) analyze important variables for a given prediction problem, (ii) visualize persistent temporal relationships learned (e.g. seasonality), and (iii) identify significant regimes changes.

9. Acknowledgements

The authors gratefully acknowledge discussions with Yaguang Li, Maggie Wang, Jeffrey Gu, Minho Jin and Andrew Moore that contributed to the development of this paper.

References

[1] J.-H. Böse, et al., Probabilistic demand forecasting at scale, Proc. VLDB Endow. 10 (12) (2017) 1694–1705.

[2] P. Courty, H. Li, Timing of seasonal sales, The Journal of Business 72 (4) (1999) 545–572.

[3] B. Lim, A. Alaa, M. van der Schaar, Forecasting treatment responses over time using recurrent marginal structural networks, in: NeurIPS, 2018.

[4] J. Zhang, K. Nawata, Multi-step prediction for influenza outbreak by an adjusted long short-term memory, Epidemiology and infection 146 (7) (2018).

[5] C. Capistran, C. Constandse, M. Ramos-Francia, Multi-horizon inflation forecasts using disaggregated data, Economic Modelling 27 (3) (2010) 666 – 677.

[6] S. S. Rangapuram, et al., Deep state space models for time series forecasting, in: NIPS, 2018.

[7] A. Alaa, M. van der Schaar, Attentive state-space modeling of disease progression, in: NIPS, 2019.

[8] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The m4 competition: 100,000 time series and 61 forecasting methods, International Journal of Forecasting 36 (1) (2020) 54 – 74.

[9] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: Probabilistic forecasting with autoregressive recurrent networks, International Journal of Forecasting (2019).

[10] R. Wen, et al., A multi-horizon quantile recurrent forecaster, in: NIPS 2017 Time Series Workshop, 2017.

[11] C. Fan, et al., Multi-horizon time series forecasting with temporal attention learning, in: KDD, 2019.

[12] S. Li, et al., Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, in: NeurIPS, 2019.

[13] J. Koutník, K. Greff, F. Gomez, J. Schmidhuber, A clockwork rnn, in: ICML, 2014.

[14] D. Neil, et al., Phased lstm: Accelerating recurrent network training for long or event-based sequences, in: NIPS, 2016.

[15] M. Ribeiro, et al., "why should i trust you?" explaining the predictions of any classifier, in: KDD, 2016.

[16] S. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: NIPS, 2017.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: NIPS, 2017.

[18] S. B. Taieb, A. Sorjamaa, G. Bontempi, Multiple-output modeling for multi-step-ahead time series forecasting, Neurocomputing 73 (10) (2010) 1950 – 1957.

[19] M. Marcellino, J. Stock, M. Watson, A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series, Journal of Econometrics 135 (2006) 499–526.

[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.

[21] Y. Wang, et al., Deep factors for forecasting, in: ICML, 2019.

[22] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: CVPR, 2017.

[23] S. O. Arik, T. Pfister, Tabnet: Attentive interpretable tabular learning (2019). arXiv:1908.07442.

[24] E. Choi, et al., Retain: An interpretable predictive model for healthcare using reverse time attention mechanism, in: NIPS, 2016.

[25] H. Song, et al., Attend and diagnose: Clinical time series analysis using attention models, 2018.

[26] J. Yoon, S. O. Arik, T. Pfister, Rl-lim: Reinforcement learning-based locally interpretable modeling (2019). arXiv:1909.12367.

[27] T. Guo, T. Lin, N. Antulov-Fantulin, Exploring interpretable LSTM neural networks over multi-variable data, in: ICML, 2019.

[28] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), in: ICLR, 2016.

[29] J. Lei Ba, J. R. Kiros, G. E. Hinton, Layer Normalization, arXiv:1607.06450 (Jul 2016). arXiv:1607.06450.

[30] Y. Dauphin, A. Fan, M. Auli, D. Grangier, Language modeling with gated convolutional networks, in: ICML, 2017.

[31] Y. Gal, Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, in: NIPS, 2016.

[32] H.-F. Yu, N. Rao, I. S. Dhillon, Temporal regularized matrix factorization for high-dimensional time series prediction, in: NIPS, 2016.

[33] C. Favorita, Corporacion favorita grocery sales forecasting competition (2018).
URL https://www.kaggle.com/c/favorita-grocery-sales-forecasting/

[34] G. Heber, A. Lunde, N. Shephard, K. K. Sheppard, Oxford-man institute's realized library (2009).
URL https://realized.oxford-man.ox.ac.uk/

[35] S. Du, G. Song, L. Han, H. Hong, Temporal causal inference with time lag, Neural Computation 30 (1) (2018) 271–291.

[36] B. Baltagi, Distributed Lags and Dynamic Models, 2008, pp. 129–145.

[37] S. Hylleberg (Ed.), Modelling Seasonality, Oxford University Press, 1992.

[38] A. Ang, A. Timmermann, Regime changes and financial markets, Annual Review of Financial Economics 4 (1) (2012) 313–337.

[39] D. Comaniciu, V. Ramesh, P. Meer, Kernel-based object tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (5) (2003) 564–577.

[40] T. Kailath, The divergence and bhattacharyya distance measures in signal selection, IEEE Transactions on Communication Technology 15 (1) (1967) 52–60.

[41] E. Giovanis, The turn-of-the-month-effect: Evidence from periodic generalized autoregressive conditional heteroskedasticity (pgarch) model, International Journal of Economic Sciences and Applied Research 7 (2014) 43–61.

APPENDIX

Appendix A. Dataset and Training Details

We provide all the sufficient information on feature pre-processing and train/test splits to ensure reproducibility of our results.

Electricity: Per [9], we use 500k samples taken between 2014-01-01 to 2014-09-01 – using the first 90% for training, and the last 10% as a validation set. Testing is done over the 7 days immediately following the training set – as described in [9, 32]. Given the large differences in magnitude between trajectories, we also apply z-score normalization separately to each entity for real-valued inputs. In line with previous work, we consider the electricity usage, day-of-week, hour-of-day and and a time index – i.e. the number of time steps from the first observation – as real-valued inputs, and treat the entity identifier as a categorical variable.

Traffic: Tests on the Traffic dataset are also kept consistent with previous work, using 500k training samples taken before 2008-06-15 as per [9], and split in the same way as the Electricity dataset. For testing, we use the 7 days immediately following the training set, and z-score normalization was applied across all entities. For inputs, we also take traffic occupancy, day-of-week, hour-of-day and and a time index as real-valued inputs, and the entity identifier as a categorical variable.

Retail: We treat each product number-store number pair as a separate entity, with over 135k entities in total. The training set is made up of 450k samples taken between 2015-01-01 to 2015-12-01, validation set of 50k samples from the 30 days after the training set, and test set of all entities over the 30-day horizon following the validation set. We use all inputs from the Kaggle competition. Data is resampled at regular daily intervals, imputing any missing days using the last available observation. We include an additional 'open' flag to denote whether data is present on a given day. We group national, regional, and local holidays into separate variables. We apply a log-transform on the sales data, and adopt z-score normalization across all entities. We consider log sales, transactions, oil to be real-valued and the rest to be categorical.

Volatility: We use the data from 2000-01-03 to 2019-06-28 – with the training set consisting of data before 2016, the validation set from 2016-2017, and the test set data from 2018 onwards. For the target, we focus on 5-min sub-sampled realized volatility (i.e. the rv5_ss column), and add daily open-to-close returns as an extra exogenous input. Additional variables are included for the day-of-week, day-of-month, week-of-year, and month – along with a 'region' variable for each index (i.e. Americas, Europe or Asia). Finally, a time index is added to denote the number of days from the first day in the training set. We treat all date-related variables (i.e. day-of-week, day-of-month, week-of-year, and month) and the region as categorical inputs. A log transformation is applied to the target, and all inputs are z-score normalized across all entities.

Appendix B. Interpretability Results

Apart from Sec. 7, which highlights our most prominent findings, we present the remaining results here for completeness.

Appendix B.1. Variable Importance

Table B.4 shows the variable importance scores for the remaining Electricity, Traffic and Volatility datasets. As these datasets only have one static input, the network allocates full weight to the entity identifier for Electricity and Traffic, along with the region input for Volatility. We also observe two types of important time-dependent inputs – those related to past values of the target as before, and those related to calendar effects. For instance, the hour-of-day plays a significant roles for Electricity and Traffic datasets, echoing the daily seasonality observed in the next section. In the Volatility dataset, the day-of-month is observed to play a significant role in future inputs – potentially reflecting turn-of-month effects [41].

Table B.4: Variable importance scores for the Electricity, Traffic and Volatility datasets. The most significant variable of each input category is highlighted in purple. As before, past values of the target play a significant role – being the top 1 or 2 most significant past input across datasets. The role of seasonality can also be seen in Electricity and Traffic, where the past and future values of the hour-of-day is important for forecasts.

	10%	50%	90%
Static			
ID	**1.000**	**1.000**	**1.000**
Past			
Hour of Day	**0.437**	**0.462**	**0.473**
Day of Week	0.078	0.099	0.151
Time Index	0.066	0.077	0.092
Power Usage	0.342	0.359	0.366
Future			
Hour of Day	**0.718**	**0.738**	**0.739**
Day of Week	0.109	0.124	0.166
Time Index	0.114	0.137	0.155

(a) Electricity

	10%	50%	90%
Static			
ID	**1.000**	**1.000**	**1.000**
Past			
Hour of Day	0.285	0.296	0.300
Day of Week	0.117	0.122	0.124
Time Index	0.107	0.109	0.111
Occupancy	**0.471**	**0.473**	**0.483**
Future			
Hour of Day	**0.781**	**0.781**	**0.781**
Day of Week	0.099	0.100	0.102
Time Index	0.117	0.119	0.121

(b) Traffic

	10%	50%	90%
Static			
Region	**1.000**	**1.000**	**1.000**
Past			
Time Index	0.093	0.098	0.142
Day of Week	0.003	0.004	0.004
Day of Month	0.017	0.027	0.028
Week of Year	0.022	0.057	0.068
Month	0.008	0.009	0.011
Open-to-close Returns	0.078	0.158	0.178
Realised Vol	**0.620**	**0.647**	**0.714**
Future			
Time Index	0.011	0.014	0.024
Day of Week	0.019	0.072	0.299
Day of Month	**0.069**	**0.635**	**0.913**
Week of Year	0.026	0.060	0.227
Month	0.008	0.055	0.713

(c) Volatility

Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction

Bryan Lim, Stefan Zohren and Stephen Roberts
Oxford-Man Institute of Quantitative Finance
Department of Engineering Science
University of Oxford
Oxford, UK
{blim,zohren,sjrob}@robots.ox.ac.uk

***Abstract*—Despite the recent popularity of deep generative state space models, few comparisons have been made between network architectures and the inference steps of the Bayesian filtering framework – with most models simultaneously approximating both state transition and update steps with a single recurrent neural network (RNN). In this paper, we introduce the Recurrent Neural Filter (RNF), a novel recurrent autoencoder architecture that learns distinct representations for each Bayesian filtering step, captured by a series of encoders and decoders. Testing this on three real-world time series datasets, we demonstrate that the decoupled representations learnt improve the accuracy of one-step-ahead forecasts while providing realistic uncertainty estimates, and also facilitate multistep prediction through the separation of encoder stages.**

***Index Terms*—recurrent neural networks, Bayesian filtering, variational autoencoders, multistep forecasting**

I. Introduction

Bayesian filtering [1] has been extensively used within the domain of time series prediction, with numerous applications across different fields – including target tracking [2], robotics [3], finance [4], and medicine [5]. Performing inference via a series of prediction and update steps [6], Bayesian filters recursively update the posterior distribution of predictions – or the belief state [7] – with the arrival of new data. For many filter models – such as the Kalman filter [8] and the unscented Kalman filter [9] – deterministic functions are used at each step to adjust the sufficient statistics of the belief state, guided by generative models of the data. Each function quantifies the impact of different sources of information on latent state estimates – specifically time evolution and exogenous inputs in the prediction step, and realised observations in the update step. On top of efficient inference and uncertainty estimation, this decomposition of inference steps enables Bayes filters to be deployed in use cases beyond basic one-step-ahead prediction – with simple extensions for multistep prediction [10] and prediction in the presence of missing observations [11].

With the increasing use of deep neural networks for time series prediction, applications of recurrent variational autoencoder (RVAE) architectures have been investigated for forecasting non-linear state space models [12]–[15]. Learning dynamics directly from data, they avoid the need for explicit model specification – overcoming a key limitation in standard Bayes filters. However, these RVAEs focus predominantly on encapsulating the generative form of the state space model – implicitly condensing both state transition and update steps into a single representation learnt by the RVAE decoder – and make it impossible to decouple the Bayes filter steps.

Recent works in deep generative modelling have focused on the use of neural networks to learn independent factors of variation in static datasets – through the encouragement of disentangled representations [16], [17] or by learning causal mechanisms [18], [19]. While a wide range of training procedures and loss functions have been proposed [20], methods in general use dedicated network components to learn distinct interpretable relationships – ranging from orthogonalising latent representations in variational autoencoders [21] to learning independent modules for different causal pathways [18]. By understanding the relationships encapsulated by each component, we can subsequently decouple them for use in related tasks – allowing the learnt mechanisms to generalise to novel domains [18], [22] or to provide building blocks for transfer learning [21].

In this paper, we introduce the Recurrent Neural Filter (RNF) – a novel recurrent autoencoder architecture which aligns network modules (encoders and decoders) with the inference steps of the Bayes filter – making several contributions over standard approaches. Firstly, we propose a new training procedure to encourage independent representations within each module, by directly training intermediate encoders with a common emission decoder. In doing so, we augment the loss function with additional regularisation terms (see Section V), and directly encourage each encoder to learn functions to update the filter's belief state given available information. Furthermore, to encourage the decoupling of encoder stages, we randomly drop out the input dynamics and error correction encoders during training – which can be viewed as artificially introducing missingness to the inputs and observations respectively. Finally, we highlight performance gains for one-step-ahead predictions through experiments on 3 real-world time series datasets, and investigate multistep predictions as a use case for generalising the RNF's decoupled representations to other tasks – demonstrating performance improvements from the recursive application of the state transition encoders alone.

II. Related Work

RVAEs for State Space Modelling: The work of [12] identifies close parallels between RNNs and latent state space models, both consisting of an internal hidden state that drives output forecasts and observations. Using an RVAE architecture described as a variational RNN (VRNN), they build their recognition network (encoder) with RNNs and produce samples for the stochastic hidden state at each time point. Deep Kalman filters (DKFs) [13], [14] take this a step further by allowing for exogenous inputs in their network and incorporating a special KL loss term to penalise state transitions between time steps. Deep Variational Bayes Filters (DVBFs) [15] enhance the interpretability of DKFs by modelling state transitions with parametric – e.g. linear – models, which take in stochastic samples from the recognition model as inputs. In general, while the above models capture the generative modelling aspects of the state space framework, their inference procedure blends both state transition and error correction steps, obliging the recognition model to learn representations for both simultaneously. In contrast, the RNF uses separate neural network components to directly model the Bayes filter steps – leading to improvements in representation learning and enhanced predictive performance in time series applications.

Hybrid Approaches: In [23], the authors take a hybrid approach with the structured variational autoencoder (SVAE), proposing an efficient general inference framework that combines probabilistic graphical models for the latent state with neural network observation models. This is similar in spirit to the Kernel Kalman Filter [24], allowing for predictions to be made on complex observational datasets – such as raw images – by encoding high dimensional outputs onto a lower dimensional latent representation modelled with a dynamical systems model. Although SVAEs provide a degree of interpretability to temporal dynamics, they also require a parametric model to be defined for the latent states which may be challenging for arbitrary time series datasets. The RNF, in comparison, can learn the relationships directly from data, without the need for explicit model specification. The Kalman variational autoencoder (KVAE) [25] extends ideas from the SVAE, modelling latent state using a linear Gaussian state space model (LGSSM). To allow for non-linear dynamics, the KVAE uses a recognition model to produce time-varying parameters for the LGSSM, weighting a set of K constant parameters using weights generated by a neural network. Deep State Space Models (DSSM) [26] investigate a similar approach within the context of time series prediction, using an RNN to generate parameters of the LGSSM at each time step. While the LGSSM components do allow for the application of the Kalman filter, we note that updates to the time-varying weights from the RNN once again blend the prediction and update steps – making the separation of Bayes filter steps and generalisation to other tasks non-trivial. On the other hand, the RNF naturally supports simple extensions (e.g. multistep prediction) similarly to other Bayes filter – due to the close alignment of the RNF architecture with the Bayes filter steps and the use of decoupled representations across encoders and decoders.

Autoregressive Architectures: An alternative approach to deep generative modelling focuses on the autoregressive factorisation of the joint distribution of observations (i.e. $p(\boldsymbol{y}_{1:T}) = \prod_t p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t})$), directly generating the conditional distribution at each step. For instance, WaveNet [27] and Transformer [28], [29] networks use dilated CNNs and attention-based models to build predictive distributions. While successful in speech generation and language applications, these models suffer from several limitations in the context of time series prediction. Firstly, the CNN and attention models require the pre-specification of the amount of relevant history to use in predictions – with the size of the look-back window controlled by the length of the receptive field or extended context – which may be difficult when the data generating process is unknown. Furthermore, they also rely on a discretisation of the output, generating probabilities of occurrence within each discrete interval using a softmax layer. This can create generalisation issues for time series where outputs are unbounded. In contrast, the LSTM cells used in the RNF recognition model remove the need to define a look-back window, and the parametric distributions used for outputs are compatible with unbounded continuous observations.

In other works, the use of RNNs in autoregressive architectures for time series prediction have been explored in DeepAR models [30], where LSTM networks output Gaussian mean and standard deviation parameters of predictive distributions at each step. We include this as a benchmark in our tests, noting the improvements observed with the RNF through its alignment with the Bayesian filtering paradigm.

Predictive State Representations: Predictive state RNNs (PSRNN) [31]–[33] use an alternative formulation of the Bayes filter, utilising a state representation that corresponds to the statistics of the predictive distribution of future observations. Predictions are made using a two-stage regression approach modelled by their proposed architectures. Compared to alternative approaches, PSRNNs only produce point estimates for their forecasts – lacking the uncertainty bounds from predictive distributions produced by the RNF.

Non-Parametric State Space Models: Gaussian Process state space models (GP-SSMs) [34], [35] and variational approximations [36], provide an alternative non-parametric approach to forecasting non-linear state space models – modelling hidden states and observation dynamics using GPs. While they have similar benefits to Bayes filters (i.e. predictive uncertainties, natural multistep prediction etc.), inference at each time step has at least an $O(T)$ complexity in the number of past observations – either via sparse GP approximations or Kalman filter formulations [37]. In contrast, the RNF updates its belief state at each time point only with the latest observations and input, making it suitable for real-time prediction on high-frequency datasets.

RNNs for Multistep Prediction: Customised sequence-to-sequence architectures have been explored in [38], [39] for multistep time series prediction, typically predefining the forecast horizon, and using computationally expensive

customised training procedures to improve performance. In contrast, the RNF does not require the use of a separate training procedure for multistep predictions – hence reducing the computational overhead – and does not require the specification of a fixed forecast horizon.

III. Problem Definition

Let $\boldsymbol{y}_t = [y_t(1), \ldots, y_t(O)]^T$ be a vector of observations, driven by a set of stochastic hidden states $\boldsymbol{x}_t = [x_t(1), \ldots, x_t(J)]^T$ and exogenous inputs $\boldsymbol{u}_t = [u_t(1), \ldots, u_t(I)]^T$. We consider non-linear state space models of the following form:

$$\boldsymbol{y}_t \sim \Pi\big(\ f(\boldsymbol{x}_t)\ \big) \tag{1}$$

$$\boldsymbol{x}_t \sim N\big(\ \mu(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t),\ \Sigma(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)\ \big) \tag{2}$$

where Π is an arbitrary distribution parametrised by a non-linear function $f(\boldsymbol{x}_t)$, with $\mu(\cdot)$ and $\Sigma(\cdot)$ being mean and covariance functions respectively.

Bayes filters allow for efficient inference through the use of a belief state, i.e. a posterior distribution of hidden states given past observations $\boldsymbol{y}_{1:t} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}$ and inputs $\boldsymbol{u}_{1:t} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t\}$. This is achieved through the maintenance of a set sufficient statistics $\boldsymbol{\theta}_t$ – e.g. means and covariances $\boldsymbol{\theta}_t \in \{\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\}$ – which compactly summarise the historical data:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) = bel(\boldsymbol{x}_t; \boldsymbol{\theta}_t) \tag{3}$$

$$= N\left(\boldsymbol{x}_t;\ \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right) \tag{4}$$

where $bel(.)$ is a probability distribution function for the belief state.

For filters such as the Kalman filter – and non-linear variants like the unscented Kalman filter [9] – $\boldsymbol{\theta}_t$ is recursively updated through a series of prediction and update steps which take the general form:

Prediction (State Transition):

$$\tilde{\boldsymbol{\theta}}_t = \phi_u\left(\boldsymbol{\theta}_{t-1}, \boldsymbol{u}_t\right) \tag{5}$$

Update (Error Correction):

$$\boldsymbol{\theta}_t = \phi_y\left(\tilde{\boldsymbol{\theta}}_t, \boldsymbol{y}_t\right) \tag{6}$$

where $\phi_u(\cdot)$ and $\phi_y(\cdot)$ are non-linear deterministic functions. Forecasts can then be computed using one-step ahead predictive distributions:

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = \int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ bel\left(\boldsymbol{x}_t; \tilde{\boldsymbol{\theta}}_t\right)\ d\boldsymbol{x}_t. \tag{7}$$

In certain cases – e.g. with the Kalman filter – the predictive distribution can also be directly parameterised using analytical functions $g(.)$ for belief state statistics :

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = p\left(\boldsymbol{y}_t|\ g(\tilde{\boldsymbol{\theta}}_t)\right). \tag{8}$$

When observations are continuous, such as in standard linear Gaussian state space models, $\boldsymbol{y}_t$ can be modelled using a Normal distribution – i.e. $\boldsymbol{y}_t \sim N\left(g_\mu(\tilde{\boldsymbol{\theta}}_t),\ g_\Sigma(\tilde{\boldsymbol{\theta}}_t)\right)$.

IV. Recurrent Neural Filter

Recurrent Neural Filters use a series of encoders and decoders to learn independent representations for the Bayesian filtering steps. We investigate two RNF variants as described below, based on Equations (7) and (8) respectively.

Variational Autoencoder Form (VRNF) Firstly, we capture the belief state of Equation (4) using a recurrent VAE-based architecture. At run time, samples of $\boldsymbol{x}_t$ are generated from the encoder – approximating the integral of Equation (7) to compute the predictive distribution of $\boldsymbol{y}_t$.

Standard Autoencoder Form (RNF)[1] Much recent work has demonstrated the sensitivity of VAE performance to the choice of prior distribution, with suboptimal priors either having an "over-regularising" effect on the loss function during training [40]–[42], or leading to posterior collapse [43]. As such, we also implement an autoregressive version of the RNF based on Equation (8) – directly feeding encoder latent states into the common emission decoder.

A general architecture diagram for both forms is shown in Figure 1, with the main differences encapsulated within $z(s)$ (see Section IV-A).

A. Network Architecture

First, let $\boldsymbol{s}_t$ be a latent state that maps to sufficient statistics $\boldsymbol{\theta}_t$, which are obtained as outputs from our recognition model. Per Equations (5) and (6), inference at run-time is controlled through the recursive update of $\boldsymbol{s}_t$, using a series of Long Short-Term Memory (LSTM) [44] encoders with exponential linear unit (ELU) activations [45].

Encoder To directly estimate the impact of exogenous inputs on the belief state, the prediction step, Equation (5), is divided into two parts with separate LSTM units $\phi_x(\cdot)$ and $\phi_u(\cdot)$. We use $\boldsymbol{h}_t$ to represent all required memory components – i.e. both output vector and cell state for the standard LSTM – with $\boldsymbol{s}_t$ being the output of the cell. A third LSTM cell $\phi_y(\cdot)$ is then used for the update step, Equation (6), with the full set of equations below.

Prediction:

$$\textit{Propagation} \qquad \left[\tilde{\boldsymbol{s}}_t^{'}, \tilde{\boldsymbol{h}}_t^{'}\right] = \phi_x(\boldsymbol{h}_{t-1}) \tag{9}$$

$$\textit{Input Dynamics} \qquad \left[\tilde{\boldsymbol{s}}_t, \tilde{\boldsymbol{h}}_t\right] = \phi_u(\tilde{\boldsymbol{h}}_t^{'}, \boldsymbol{u}_t) \tag{10}$$

Update:

$$\textit{Error Correction} \qquad [\boldsymbol{s}_t, \boldsymbol{h}_t] = \phi_y(\tilde{\boldsymbol{h}}_t, \boldsymbol{y}_t) \tag{11}$$

For the variational RNF, hidden state variable $\boldsymbol{x}_t$ is modelled as multivariate Gaussian, given by:

$$\boldsymbol{x}_t \sim N(m(\tilde{\boldsymbol{s}}_t), V(\tilde{\boldsymbol{s}}_t)) \tag{12}$$

$$m(\tilde{\boldsymbol{s}}_t) = \boldsymbol{W}_m\tilde{\boldsymbol{s}}_t + \boldsymbol{b}_m \tag{13}$$

$$V(\tilde{\boldsymbol{s}}_t) = \text{diag}(\sigma(\tilde{\boldsymbol{s}}_t) \odot \sigma(\tilde{\boldsymbol{s}}_t)) \tag{14}$$

$$\sigma(\tilde{\boldsymbol{s}}_t) = \text{Softplus}(\boldsymbol{W}_\sigma\tilde{\boldsymbol{s}}_t + \boldsymbol{b}_\sigma), \tag{15}$$

[1]An open-source implementation of the standard RNF can be found at: https://github.com/sjblim/rnf-ijcnn-2020

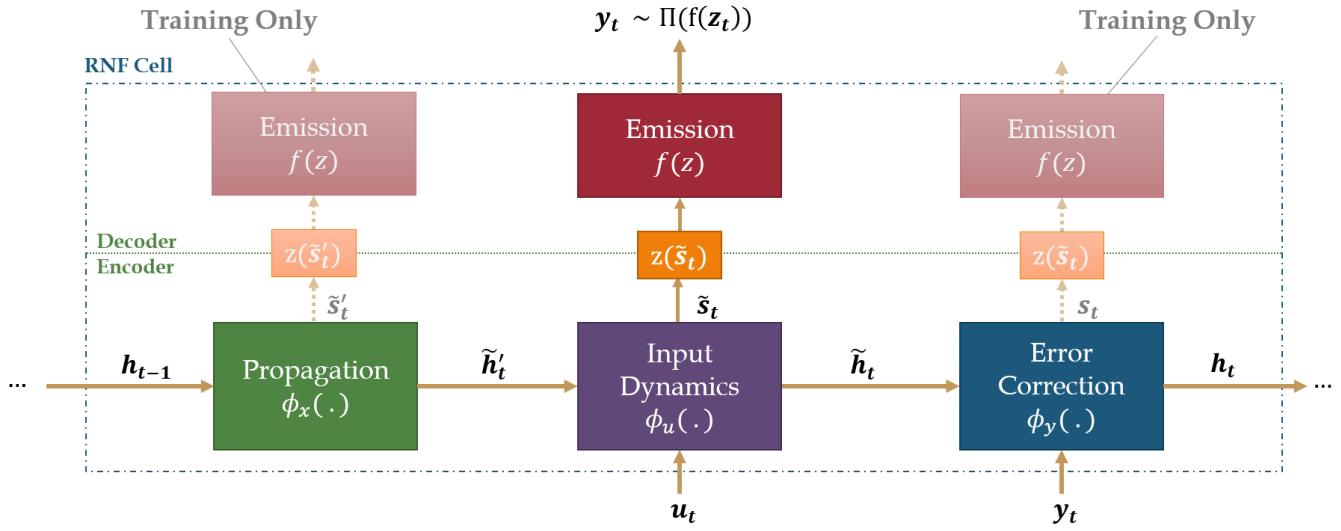


Fig. 1. RNF Network Architecture

where $\boldsymbol{W}_{(\cdot)}, \boldsymbol{b}_{(\cdot)}$ are the weights/biases of each layer, and $\odot$ is an element-wise (Hadamard) product.

For the standard RNF, the encoder state $\tilde{\boldsymbol{s}}_t$ is directly fed into the emission decoder leading to the following forms for $\tilde{\boldsymbol{z}}_t = z(\tilde{\boldsymbol{s}}_t)$:

$$z_{\text{VRNF}}(\tilde{\boldsymbol{s}}_t) = \boldsymbol{x}_t, \qquad z_{\text{RNF}}(\tilde{\boldsymbol{s}}_t) = \tilde{\boldsymbol{s}}_t. \tag{16}$$

While the connection to non-linear state-space models facilitates our interpretation of $z_{\text{RNF}}(\tilde{\boldsymbol{s}}_t)$, we note that the standard RNF no longer relies on an explicit generative model for the latent state $\boldsymbol{x}_t$. This potentially allows the standard RNF to learn more complex update rules for non-Gaussian latent states.

Decoder Given an encoder output $\tilde{\boldsymbol{z}}_t$, we use a multi-layer perceptron to model the emission function $f(\cdot)$:

$$f(\tilde{\boldsymbol{z}}_t) = \boldsymbol{W}_{z_2}\ \text{ELU}(\boldsymbol{W}_{z_1}\ \tilde{\boldsymbol{z}}_t + \boldsymbol{b}_{z_1}) + \boldsymbol{b}_{z_2}. \tag{17}$$

This allows us to handle both continuous or binary observations using the output models below:

$$\boldsymbol{y}_t^{\text{continuous}} \sim N\Big(f_\mu(\tilde{\boldsymbol{z}}_t)\ , \Gamma(\tilde{\boldsymbol{z}}_t)\Big), \tag{18}$$

$$\boldsymbol{y}_t^{\text{binary}} \sim \text{Bernoulli}\Big(\text{Sigmoid}(f(\tilde{\boldsymbol{z}}_t))\Big). \tag{19}$$

where $\Gamma(\tilde{\boldsymbol{z}}_t) = diag\,(g_\sigma(\tilde{\boldsymbol{z}}_t))$ is a time-dependent diagonal covariance matrix, and $g_\sigma(\tilde{\boldsymbol{z}}_t) = \text{Softplus}(f_\sigma(\tilde{\boldsymbol{z}}_t))$.

For $\boldsymbol{y}_t^{\text{continuous}}$, the weights W_{z_1}, b_{z_1} are shared between $f_\mu(\cdot)$ and $f_\sigma(\cdot)$ – i.e. both observation means and covariances are generated from the same encoder hidden layer.

B. Handling Missing Data and Multistep Prediction

From the above, we can see that each encoder learns how specific inputs (i.e. time evolution, exogenous inputs and the target) modify the belief state. As such, in a similar fashion to Bayes filters, we decouple the RNF stages at run-time based on the availability of inputs for prediction – allowing it to handle applications involving missing data or multistep forecasting.

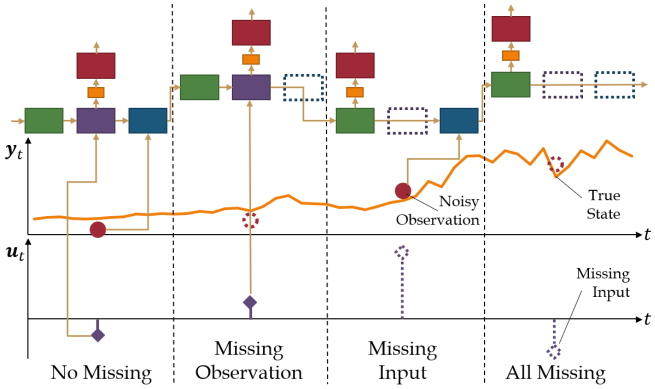


Fig. 2. RNF Configuration with Missing Data

Figure 2 demonstrates how the RNF stages can be combined to accommodate missing data, noting that the colour scheme of the encoders/decoders shown matches that of Figure 1. From the schematic, the propagation encoder – which is responsible for changes to the belief state due to time evolution – is always applied, with the input dynamics and error correction encoders only used when inputs or observations are observed respectively. Where inputs are available, the emission decoder is applied to the input dynamics encoder to generate predictions at each step. Failing that, the decoder is applied to the propagation encoder alone.Multistep forecasts can also be treated as predictions in the absence of inputs or observations, with the encoders used to project the belief state in a similar fashion to missing data.

V. Training Methodology

Considering the joint probability for a trajectory of length T, we train the standard RNF by minimising the negative log-

likelihood of the observations. For continuous observations, this involves Gaussian likelihoods from Equation (18):

$$\mathcal{L}_{\text{RNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\tilde{\boldsymbol{s}}_t), \tag{20}$$

$$\log p(\boldsymbol{y}_t|\tilde{\boldsymbol{s}}_t) = -\frac{1}{2}\sum_{j=1}^{J}\Bigg\{\log(2\pi g_\sigma(j,\tilde{\boldsymbol{s}}_t)^2) + \left\|\frac{\boldsymbol{y}_t(j) - f_\mu(j,\tilde{\boldsymbol{s}}_t)}{g_\sigma(j,\tilde{\boldsymbol{s}}_t)}\right\|^2\Bigg\}, \tag{21}$$

where $\boldsymbol{\omega}$ are the weights of the deep neural network, $f_\mu(j,\tilde{\boldsymbol{z}}_t)$ is the j-th element of $f_\mu(\tilde{\boldsymbol{z}}_t)$, and $g_\sigma(j,\tilde{\boldsymbol{z}}_t)$ the j-th element of $g_\sigma(\tilde{\boldsymbol{z}}_t)$.

For the VRNF, we adopt the Stochastic Gradient Variational Bayes (SGVB) estimator of [46] for our VAE evidence lower bound, expressing our loss function as:

$$\mathcal{L}_{\text{VRNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T}) = \sum_{t=1}^{T}\left\{\frac{1}{L}\sum_{i=1}^{L}\log p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)}(\tilde{\boldsymbol{s}}_t))\right\} - KL\big(q(\boldsymbol{x}_{1:T}) \;||\; p(\boldsymbol{x}_{1:T})\big), \tag{22}$$

where L is the number of samples used for calibration, $\boldsymbol{x}_k^{(i)}(\tilde{\boldsymbol{s}}_k)$ is the i-th sample given the latent state $\tilde{\boldsymbol{s}}_k$, and $KL(\cdot)$ is the KL divergence term defined based on the priors in Section V-A.

A. VAE Priors for VRNF

Using the generative model for $\boldsymbol{x}_t$ in Equation (2), we consider the definition of two priors for the VRNF, as described briefly below. A full definition can be found in Appendix[2] A, which also includes derivations for the KL term used in $\mathcal{L}_{\text{VRNF}}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T})$.

Kalman Filter Prior (VRNF-KF) Considering a linear Gaussian state space form for Equations (1) and (2), we can apply the Kalman filtering equations to obtained distributions for $\boldsymbol{x}_t$ at each time step (e.g. $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t})$). This also lets us analytically define how the means and covariances of the belief state change with different sets of information – aligning the VRNF's encoder stages with the filtering equations.

Neural Network Prior (VRNF-NN) In the spirit of the DKF [13], the analytical equations from the Kalman filter prior above can also be approximated using simple multilayer perceptrons. This would also allow belief state updates to accommodate non-linear states space dynamics, making it fa less restrictive prior model.

B. Encouraging Decoupled Representations

Combined Encoder Training To improve representation learning, the RNF is trained in a "multi-task" fashion – with each intermediate stage trained to encode latent states for output distributions. This is achieved by applying the same emissions decoder to all encoders during training as indicated in Figure 1, with each encoder/decoder aligned with the Bayesian filtering steps described in Section IV-A. Encoders are then trained jointly using the combined loss function below:

$$\mathcal{L}_{\text{combined}}(\boldsymbol{\omega}, \boldsymbol{y}_{1:T}, \boldsymbol{u}_{1:T}) = \underbrace{\mathcal{L}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}_{1:T})}_{\text{Input Dynamics}} + \overbrace{\underbrace{\alpha_x \mathcal{L}(\boldsymbol{\omega}, \tilde{\boldsymbol{s}}'_{1:T})}_{\text{Propagation}} + \underbrace{\alpha_y \mathcal{L}(\boldsymbol{\omega}, \boldsymbol{s}_{1:T})}_{\text{Error Correction}}}^{\text{Additional Regularisation Terms}}. \tag{23}$$

As such, the additional stages can be interpreted as regularisation terms for the VRNF or RNF loss functions – which we weight by constants α_x and α_y to control the relative importance of the intermediate encoder representations. For our main experiments, we place equal importance on all encoders, i.e. $\alpha_x = \alpha_y = 1$, to facilitate the subsequent separation of stages for multistep prediction – with a full ablation analysis performed to assess the impact of various α settings during training.

Furthermore, the error correction component $\phi_y(\cdot)$ can also be interpreted as a pure auto-encoding step for the latest observation, recovering distributions $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ based on filtered distributions of $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t})$. Given that all stages share the same emissions decoder, this obliges the network to learn representations for $\boldsymbol{s}_t$ that are able to reconstruct the current observation when it is available.

Introducing Artificial Missingness Next, to encourage the clean separation of encoder stages for generalisation to other tasks, we break dependencies between the encoders by introducing artificial missingness into the dataset – randomly dropping out inputs and observations with a missingness rate r. As encoders are only applied where data is present (see Figure 2), input dynamics and error correction encoders are hence randomly skipped over during training – encouraging the encoder to perform regardless of which encoder stage preceded it. This also bears a resemblance to input dropout during training, which we apply to competing benchmarks to ensure comparability.

VI. Performance Evaluation

A. Time Series Datasets

We conduct a series of tests on 3 real-world time series datasets to evaluate performance:

1) **Electricity:** The public UCI Individual Household Electric Power Consumption Data [47]
2) **Volatility:** A 30-min realised variance [48] dataset for 30 different stock indices
3) **Quote:** A high-frequency market microstructure dataset containing Barclays Level-1 quote data from Thomson Reuters Tick History (TRTH)

Details on input/output features and preprocessing are fully documented in Appendix C for reference.

B. Conduct of Experiment

Benchmarks: We compare the VRNF-KF, VRNF-NN and standard RNF against a range of autoregressive and RVAE benchmarks – including the DeepAR Model [30], Deep State

[2]URL for full paper with appendix: https://arxiv.org/abs/1901.08096

TABLE I
NORMALISED MSES FOR ONE-STEP-AHEAD PREDICTIONS

	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Electricity	0.908	1.000	2.002	0.867	0.861	0.852	**0.780***
Volatility	3.956	1.000	0.991	0.982	1.914	1.284	**0.976***
Quote	0.998	1.000	3.733	1.001	1.000	1.001	**0.997***

TABLE II
COVERAGE PROBABILITY OF ONE-STEP-AHEAD 90% PREDICTION INTERVAL

	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Electricity	0.966	0.964	0.981	0.965	**0.320**	**0.271**	**0.961***
Volatility	**0.997***	0.999	1.000	1.000	1.000	1.000	1.000
Quote	0.997	0.991	**0.005**	0.998	**0.924 ***	0.992	0.997

TABLE III
NORMALISED MSES FOR MULTISTEP PREDICTIONS WITH BOTH UNKNOWN AND KNOWN INPUTS

Input Type	Dataset	$\tau =$	DeepAR	DSSM	VRNN	DKF	VRNF-KF	VRNF-NN	RNF
Unknown Inputs	**Electricity**	5	3.260	3.308	3.080	2.946	2.607	2.015	**1.996***
		10	4.559	4.771	4.533	4.419	5.467	**3.506***	3.587
		20	6.555	6.827	6.620	6.524	9.817	**5.449***	6.098
	Volatility	5	3.945	1.628	0.994	0.986	4.084	1.020	**0.967***
		10	3.960	1.639	0.994	0.985	4.140	1.017	**0.967***
		20	3.955	1.641	0.993	0.983	4.163	1.014	**0.966***
	Quote	5	1.000	1.000	1.001	1.000	1.002	0.999	**0.998***
		10	1.000	1.001	1.000	1.001	1.009	1.002	**1.000***
		20	1.000	1.001	1.003	1.001	1.488	1.003	**1.000***
Known Inputs	**Electricity**	5	3.260	3.199	3.045	1.073	1.112	0.877	**0.813***
		10	4.559	4.382	4.470	1.008	1.180	0.882	**0.831***
		20	6.555	6.174	6.514	0.989	1.209	0.884	**0.846***
	Volatility	5	3.988	1.615	0.994	0.986	2.645	1.009	**0.981***
		10	3.992	1.620	0.994	0.985	2.652	1.009	**0.981***
		20	3.991	1.627	0.993	0.984	2.652	1.008	**0.980***
	Quote	5	1.000	1.000	0.998	1.000	1.001	1.000	**0.997***
		10	1.000	1.000	0.999	1.000	1.003	1.000	**0.998***
		20	1.000	1.000	1.003	1.000	1.009	1.000	**0.999***

Space Model (DSSM) [26], Variational RNN (VRNN) [12], and Deep Kalman Filter (DKF) [13].

For multistep prediction, we consider two potential use cases for exogenous inputs: (i) when future inputs are unknown beforehand and imputed using their last observed values, and (ii) when inputs are known in advance and used as given. When models require observations of $\boldsymbol{y}_t$ as inputs, we recursively feed outputs from the network as inputs at the next time step. These tweaks allow the benchmarks to be used for multistep prediction without modifying network architectures. For the RNF, we consider the application of the propagation encoder alone for the former case, and a combination of the propagation and input dynamics encoder for the latter – as detailed in Section IV-B.

Metrics: To determine the accuracy of forecasts, we evaluate the mean-squared-error (MSE) for single-step and multistep predictions, normalising each using the MSE of the one-step-ahead forecast for the best autoregressive model (i.e. the DSSM). For multistep forecasts, we measure the average squared error up to the maximum prediction horizon (τ). As observations are 1D continuous variables for all our datasets, we evaluate uncertainty estimates using the prediction interval coverage probability (PICP) of a 90% prediction interval, defined as:

$$\text{PICP} = \frac{1}{T}\sum_{t=1}^{T} c_t, \tag{24}$$

$$c_t = \begin{cases} 1, & \text{if } \psi(0.05,t) < y_t < \psi(0.95,t) \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

where $\psi(0.05,t)$ is the 5^{th} percentile of samples from $N\left(f(\boldsymbol{x}_t), \boldsymbol{\Gamma}\right)$.

Training Details: Please refer to Appendix B for full details of network calibration.

C. Results and Discussion

On the whole, the standard RNF demonstrates the best overall performance – improving MSEs in general for one-step-ahead

TABLE IV
NORMALISED MSES FOR ABLATION STUDIES

		Electricity				Volatility				Quote			
	$\tau =$	1	5	10	20	1	5	10	20	1	5	10	20
Unknown Inputs	**RNF**	-	**1.996***	**3.587***	**6.098***	-	**0.967***	**0.967***	**0.966***	-	**0.998***	**1.000***	**1.000***
	RNF-NS	-	2.801	**13.409**	45.625	-	1.006	1.006	1.005	-	1.137	1.294	1.260
	RNF-IO	-	14.047	14.803	15.414	-	1.377	1.458	1.494	-	1.029	1.042	1.045
Known Inputs	**RNF**	0.780	0.813	**0.831***	**0.846***	**0.976***	**0.981***	**0.981***	**0.980***	**0.997***	**0.997***	**0.998***	**0.999***
	RNF-NS	0.828	0.948	0.997	1.042	0.979	0.983	0.983	0.982	1.001	1.003	1.019	1.026
	RNF-IO	**0.770***	**0.809***	0.873	0.918	1.012	1.016	1.016	1.015	1.020	1.015	1.023	1.030

and multistep prediction. From the one-step-ahead MSEs in Table I, the RNF improves forecasting accuracy by 19.6% on average across all datasets and benchmarks. These results are also echoed for multistep predictions in Table III, with the RNF beating the majority of baselines for all horizons and datasets. The only exception is the slight out-performance of another RNF variant (the VRNF-NN) on the Electricity dataset with unknown inputs – possibly due to the adoption of a suitable prior for this specific dataset – with the standard RNF coming in a close second. The PICP results of Table II also show that performance is achieved without sacrificing the quality of uncertainty estimates, with the RNF outputting similar uncertainty intervals compared to other deep generative and autoregressive models. On the whole, this demonstrates the benefits of the proposed training approach for the RNF, which encourages decoupled representations using regularisation terms and skip training.

To measure the benefits of the skip-training approach and proposed regularisation terms, we also perform a simple ablation study and train the RNF without the proposed components. Table IV shows the normalised MSEs for the ablation studies, with one-step and multistep forecasts combined into the same table. Specifically, we test the RNF with no skip training in RNF-NS, and the RNF with only input dynamics outputs included in the loss function (i.e. $\alpha_x = \alpha_y = 0$) in RNF-IO. As inputs are always known for one-step-ahead predictions, normalised MSEs for $\tau = 1$ are omitted for unknown inputs. In general, the inclusion of both skip training and regularisation terms improves forecasting performance, particularly in the case of longer-horizon predictions. We observe this from the MSE improvements for all but short-term ($\tau \in \{1, 5\}$) predictions for known inputs, where the RNF-IO. However, the importance of both skip-training and regularisation can be seen from the large multistep MSEs of both the RNF-NS and RNF-IO on the Electricity dataset with unknown inputs– which results from error propagation when the input dynamics encoder is removed.

As mentioned in Section IV, the challenges of prior selection for VAE-based methods can be seen from the PICPs in Table II – with small PICPs for VRNF models indicative of miscalibrated distributions in the Electricity data, and the poor MSEs and PICPs for the VRNN indicative of posterior collapse on the Quote data. However, this can also be beneficial when applied to appropriate datasets – as seen from the closeness of the VRNF-KF's PICP to the expected 90% on the Quote data. As such, the autoregressive form of standard RNF leads to more reliable performance from both a prediction accuracy and uncertainty perspective – doing away with the need to define a prior for $\boldsymbol{x}_t$.

VII. CONCLUSIONS

In this paper, we introduce a novel recurrent autoencoder architecture, which we call the Recurrent Neural Filter (RNF), to learn decoupled representations for the Bayesian filtering steps – consisting of separate encoders for state propagation, input and error correction dynamics, and a common decoder to model emission. Based on experiments with three real-world time series datasets, the direct benefits of the architecture can be seen from the improvements in one-step-ahead predictive performance, while maintaining comparable uncertainty estimates to benchmarks. Due to its modular structure and close alignment with Bayesian filtering steps, we also show the potential to generalise the RNF to similar predictive tasks – as seen from improvements in multistep prediction using extracted state transition encoders.

REFERENCES

[1] J. V. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. New York, NY, USA: Wiley-Interscience, 2009.

[2] A. J. Haug, *Bayesian estimation and tracking: a practical guide*. Hoboken, NJ: John Wiley & Sons, 2012.

[3] T. D. Barfoot, *State Estimation for Robotics*. New York, NY, USA: Cambridge University Press, 2017.

[4] H. Ghosh, B. Gurung, and Prajneshu, "Kalman filter-based modelling and forecasting of stochastic volatility with threshold," *Journal of Applied Statistics*, vol. 42, no. 3, pp. 492–507, 2015.

[5] R. Sukkar, E. Katz, Y. Zhang, D. Raunig, and B. T. Wyman, "Disease progression modeling using hidden Markov models," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2012, pp. 2845–2848.

[6] S. Sarkka, *Bayesian Filtering and Smoothing*. New York, NY, USA: Cambridge University Press, 2013.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[8] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[9] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," vol. 3068, 1997.

[10] A. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1991.

[11] A. C. Harvey and R. G. Pierse, "Estimating missing observations in economic time series," *Journal of the American Statistical Association*, vol. 79, no. 385, pp. 125–131, 1984.

[12] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in Neural Information Processing Systems 28 (NIPS 2016)*, 2015.

[13] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep Kalman Filters," *ArXiv e-prints*, 2015. [Online]. Available: http://arxiv.org/abs/1511.05121

[14] R. G. Krishnan, U. Shalit, and D. Sontag, "Structured inference networks for nonlinear state space models," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 2017.

[15] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt, "Deep variational Bayes filters: unsupervised learning of state space models from raw data," in *International Conference on Learning Representations (ICLR 2017)*, 2017.

[16] S. Narayanaswamy, T. B. Paige, J.-W. van de Meent, A. Desmaison, N. Goodman, P. Kohli, F. Wood, and P. Torr, "Learning disentangled representations with semi-supervised deep generative models," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, pp. 5925–5935.

[17] H. Kim and A. Mnih, "Disentangling by factorising," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[18] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf, "Learning independent causal mechanisms," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[19] V. Thomas, E. Bengio, W. Fedus, J. Pondard, P. Beaudoin, H. Larochelle, J. Pineau, D. Precup, and Y. Bengio, "Disentangling the independently controllable factors of variation by interacting with the world," in *NIPS 2017 Workshop on Learning Disentangled Representations*, 2018.

[20] F. Locatello, S. Bauer, M. Lucic, S. Gelly, B. Schölkopf, and O. Bachem, "Challenging common assumptions in the unsupervised learning of disentangled representations," *CoRR*, vol. abs/1811.12359, 2018. [Online]. Available: http://arxiv.org/abs/1811.12359

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations (ICLR 2017)*, 2017.

[22] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[23] M. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta, "Composing graphical models with neural networks for structured representations and fast inference," in *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.

[24] L. Ralaivola and F. d'Alche Buc, "Time series filtering, smoothing and learning using the kernel Kalman filter," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, vol. 3, July 2005, pp. 1449–1454.

[25] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[26] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 2018.

[27] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[29] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," *CoRR*, vol. abs/1901.02860, 2019. [Online]. Available: http://arxiv.org/abs/1901.02860

[30] V. Flunkert, D. Salinas, and J. Gasthaus, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *CoRR*, vol. abs/1704.04110, 2017. [Online]. Available: http://arxiv.org/abs/1704.04110

[31] K. Choromanski, C. Downey, and B. Boots, "Initialization matters: Orthogonal predictive state recurrent neural networks," in *International Conference on Learning Representations (ICLR 2018)*, 2018.

[32] C. Downey, A. Hefny, B. Boots, G. J. Gordon, and B. Li, "Predictive state recurrent neural networks," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[33] A. Venkatraman, N. Rhinehart, W. Sun, L. Pinto, M. Hebert, B. Boots, K. Kitani, and J. Bagnell, "Predictive-state decoders: Encoding the future into recurrent networks," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[34] R. Turner, M. Deisenroth, and C. Rasmussen, "State-space inference and learning with Gaussian processes," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, 2010, pp. 868–875.

[35] H. Nickisch, A. Solin, and A. Grigorevskiy, "State space Gaussian processes with non-Gaussian likelihood," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018, pp. 3789–3798.

[36] A. Doerr, C. Daniel, M. Schiegg, N.-T. Duy, S. Schaal, M. Toussaint, and T. Sebastian, "Probabilistic recurrent state-space models," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[37] S. Sarkka, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, July 2013.

[38] B. Pérez Orozco, G. Abbati, and S. Roberts, "MOrdReD: Memory-based Ordinal Regression Deep Neural Networks for Time Series Forecasting," *CoRR*, vol. arXiv:1803.09704, 2018. [Online]. Available: http://arxiv.org/abs/1803.09704

[39] R. Wen and K. T. B. M. Narayanaswamy, "A multi-horizon quantile recurrent forecaster," in *NIPS 2017 Time Series Workshop*, 2017.

[40] H. Takahashi, T. Iwata, Y. Yamanaka, M. Yamada, and S. Yagi, "Variational autoencoder with implicit optimal priors," *CoRR*, vol. abs/1809.05284, 2018. [Online]. Available: https://arxiv.org/abs/1809.05284

[41] Tomczak and Welling, "VAE with a VampPrior," in *Proceedings of the 21st Internation Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[42] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," *CoRR*, vol. abs/1511.06349, 2015. [Online]. Available: http://arxiv.org/abs/1511.06349

[43] A. van den Oord, O. Vinyals, and k. kavukcuoglu, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017, pp. 6306–6315.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[45] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *International Conference on Learning Representations (ICLR 2016)*, 2016.

[46] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR 2014)*, 2014.

[47] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository – individual household electric power consumption data set," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets

[48] T. Andersen, T. Bollerslev, F. Diebold, and P. Labys, "Modeling and forecasting realized volatility," *Econometrica*, vol. 71, no. 2, pp. 579–625, 2003.

[49] A. Yadav, P. Awasthi, N. Naik, and M. R. Ananthasayanam, "A constant gain kalman filter approach to track maneuvering targets," in *2013 IEEE International Conference on Control Applications (CCA)*, 2013.

[50] T. G. Andersen and T. Bollerslev, "Intraday periodicity and volatility persistence in financial markets," *Journal of Empirical Finance*, vol. 4, no. 2, pp. 115 – 158, 1997.

[51] A. Todd, R. Hayes, P. Beling, and W. Scherer, "Micro-price trading in an order-driven market," in *2014 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*, March 2014, pp. 294–297.

[52] lvaro Cartea, R. Donnelly, and S. Jaimungal, "Enhancing trading strategies with order book signals," *Applied Mathematical Finance*, vol. 25, no. 1, pp. 1–35, 2018.

APPENDIX

A. VRNF Priors and Derivation of KL Term

Defining a prior distribution for the VRNF starts with the specification of a model for the distribution of hidden state $\boldsymbol{x}_t$, conditioned on the amount of available information at each encoder to achieve alignment with the VRNF stages. Per the generative model of Equation (2), we model $\boldsymbol{x}_t$ as a multivariate normal distribution with a mean and covariance that varies with time and the information present at each encoder, based on the notation below:

Propagation:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t-1}) \sim N(\tilde{\boldsymbol{\beta}}_t^{'}, \tilde{\boldsymbol{\nu}}_t^{'}), \tag{26}$$

Input Dynamics:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) \sim N(\tilde{\boldsymbol{\beta}}_t, \tilde{\boldsymbol{\nu}}_t), \tag{27}$$

Error Correction:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) \sim N(\boldsymbol{\beta}_t, \boldsymbol{\nu}_t). \tag{28}$$

For the various priors defined in this section, we adopt the use of diagonal covariance matrices for the inputs, defined as:

$$\boldsymbol{\nu}_t = diag(\boldsymbol{\gamma}_t \odot \boldsymbol{\gamma}_t), \tag{29}$$

where $\boldsymbol{\gamma}_t \in \mathbb{R}^J$ is a vector of standard deviation parameters.

This approximation helps to reduce the computational complexity associated with the matrix multiplications using full covariance matrices, and the $O(J^2T)$ memory requirements from storing full covariances matrices for an RNN unrolled across T timesteps.

KL Divergence Term

Considering the application of the input dynamics encoder alone (i.e. $\alpha_x = \alpha_y = 0$), the KL divergence between independent conditional multivariate Gaussians at each time step can be hence expressed analytically as:

$$KL_{\text{Input}}\big(q(\boldsymbol{x}_{1:T}) \;||\; p(\boldsymbol{x}_{1:T})\big) \tag{30}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T})}\Bigg[\log \frac{p(\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\tilde{\boldsymbol{s}}_1)} + \sum_{t=2}^{T} \log \frac{p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_{1:t}, \boldsymbol{y}_{1:t-1})}{q(\boldsymbol{x}_t|\tilde{\boldsymbol{s}}_t)}\Bigg] \tag{31}$$

$$= \sum_{t=1}^{T}\sum_{j=1}^{J}\Bigg\{\log \frac{\tilde{\gamma}_t(j)}{\sigma(j, \tilde{\boldsymbol{s}}_t)} + \frac{\sigma(j, \tilde{\boldsymbol{s}}_t)^2 + (m(j, \tilde{\boldsymbol{s}}_t) - \tilde{\beta}_t(j))^2}{2\tilde{\gamma}_t(j)^2} - \frac{1}{2}\Bigg\}, \tag{32}$$

where $m(j, \tilde{\boldsymbol{s}}_t), \sigma(j, \tilde{\boldsymbol{s}}_t)$ are j-th elements of $m(\tilde{\boldsymbol{s}}_t), \sigma(\tilde{\boldsymbol{s}}_t)$ as defined in Equations (14) and (15) respectively.

The KL divergence terms are defined similarly for the propagation and error correction encoders, using the means and standard deviations defined above.

Kalman Prior (VRNF-KF)

The use Kalman filter relies on the definition of a linear Gaussian state space model, which we specify below:

$$\boldsymbol{y}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{e}_t, \tag{33}$$

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{\epsilon}_t, \tag{34}$$

where $\boldsymbol{H}, \boldsymbol{A}, \boldsymbol{B}$ are constant matrices, and $\boldsymbol{e}_t \sim N(0, \boldsymbol{R}), \boldsymbol{\epsilon}_t \sim N(0, \boldsymbol{Q})$ are noise terms with constant noise covariances $\boldsymbol{R}$ and $\boldsymbol{Q}$.

a) Propagation: Assuming that inputs $\boldsymbol{u}_t$ is unknown at time t, predictive distributions can still be computed for the hidden state if we have a model for $\boldsymbol{u}_t$. In the simplest case, this can be a standard normal distribution, i.e. $\boldsymbol{u}_t \sim N(\boldsymbol{c}, \boldsymbol{D})$ – where $\boldsymbol{c}$ is a constant mean vector and $\boldsymbol{D}$ a constant covariance matrix. Under this model, predictive distributions can be computed as below:

$$\tilde{\boldsymbol{\beta}}_t^{'} = \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{B}\boldsymbol{c} = \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{c}^{'}, \tag{35}$$

$$\tilde{\boldsymbol{\nu}}_t^{'} = \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^T + \boldsymbol{Q} = \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{Q}^{'}, \tag{36}$$

with $\boldsymbol{c}^{'}, \boldsymbol{Q}^{'}$ collapsing constant terms together into a single parameters.

b) Input Dynamics: When inputs are known, the forecasting equations take on a similar form:

$$\tilde{\boldsymbol{\beta}}_t = \boldsymbol{A}\boldsymbol{\beta}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t \tag{37}$$

$$\tilde{\boldsymbol{\nu}}_t = \boldsymbol{A}\boldsymbol{\nu}_{t-1}\boldsymbol{A}^T + \boldsymbol{Q} \tag{38}$$

Comparing this with the forecasting equations of the propagation step, we can express also the above as functions $\tilde{\boldsymbol{\beta}}_t^{'}$ and $\tilde{\boldsymbol{\nu}}_t^{'}$, i.e.:

$$\tilde{\boldsymbol{\beta}}_t = \tilde{\boldsymbol{\beta}}_t^{'} + \boldsymbol{B}\boldsymbol{u}_t - \boldsymbol{c}^{'}, \tag{39}$$

$$\tilde{\boldsymbol{\nu}}_t = \tilde{\boldsymbol{\nu}}_t^{'} - \boldsymbol{Q}^{'} + \boldsymbol{Q}. \tag{40}$$

c) Error Correction: Upon receipt of a new observation, the Kalman filter computes a Kalman Gain $\boldsymbol{K}_t$, using it to correct the belief state as below:

$$\boldsymbol{\beta}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\,\tilde{\boldsymbol{\beta}}_t - \boldsymbol{K}_t\boldsymbol{H}\boldsymbol{y}_t, \tag{41}$$

$$\boldsymbol{\nu}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\,\tilde{\boldsymbol{\nu}}_t, \tag{42}$$

where $\boldsymbol{I}$ is an identity matrix and $\boldsymbol{K}_t = \tilde{\boldsymbol{\nu}}_t\boldsymbol{H}^T\left(\boldsymbol{H}\tilde{\boldsymbol{\nu}}_t\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1}$.

Approximations for Efficiency

To avoid the complex memory and space requirements associated with full matrix computations, we make the following approximations in our Kalman Filter equations.

d) Constant Kalman Gain: Firstly, as noted in [49], Kalman gain values in stable filters usually tend towards a steady state value after a initial transient period. We hence fix the Kalman gain at a constant value, and collapse constant coefficients in the error correction equations to give:

$$\boldsymbol{\beta}_t = \boldsymbol{K}^{'}\tilde{\boldsymbol{\beta}}_t - \boldsymbol{H}^{'}\boldsymbol{y}_t, \tag{43}$$

$$\boldsymbol{\nu}_t = \boldsymbol{K}^{'}\tilde{\boldsymbol{\nu}}_t, \tag{44}$$

Where $\boldsymbol{K}^{'} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})$ and $\boldsymbol{H}^{'} = \boldsymbol{K}\boldsymbol{H}$.

e) Independent Hidden State Dimensions: Next, we assume that hidden state dimensions are independent of one another, which effectively diagonalising state related coefficients $\boldsymbol{A} = diag(\boldsymbol{a})$ and $\boldsymbol{Q} = diag(\boldsymbol{q})$.

f) Diagonalising $\boldsymbol{Q}^{'}, \boldsymbol{K}^{'}$: Finally, to allow us to diagonal covariance matrices throughout our equations, we also diagonalise $\boldsymbol{Q}^{'} = diag(\boldsymbol{q}^{'})$ and $\boldsymbol{K}^{'} = diag(\boldsymbol{k}^{'})$.

Prior Definition

Using the above definitions and approximations, the Kalman filter prior can hence be expressed in vector form using the equations below:

Propagation:

$$\tilde{\boldsymbol{\beta}}_t^{'} = \boldsymbol{a} \odot m(\boldsymbol{s}_{t-1}) + \boldsymbol{c}^{'}, \tag{45}$$

$$\tilde{\boldsymbol{\nu}}_t^{'} = \boldsymbol{a} \odot V(\boldsymbol{s}_{t-1}) \odot \boldsymbol{a} + \boldsymbol{q}^{'}. \tag{46}$$

Input Dynamics:

$$\tilde{\boldsymbol{\beta}}t = \boldsymbol{a} \odot m(\boldsymbol{s}_{t-1}) + \boldsymbol{B}\boldsymbol{u}_t, \tag{47}$$

$$\tilde{\boldsymbol{\nu}}_t = \boldsymbol{a} \odot V(\boldsymbol{s}_{t-1}) \odot \boldsymbol{a} + \boldsymbol{q}. \tag{48}$$

Error Correction:

$$\boldsymbol{\beta}_t = \boldsymbol{k}^{'} \odot m(\tilde{\boldsymbol{s}}_t) - \boldsymbol{H}^{'}\boldsymbol{y}_t, \tag{49}$$

$$\boldsymbol{\nu}_t = \boldsymbol{k}^{'} \odot V(\tilde{\boldsymbol{s}}_t). \tag{50}$$

All constant standard deviation are implemented as coefficients wrapped in a softmax layer (e.g. $\boldsymbol{a} = \text{softplus}(\phi)$)) to prevent the optimiser from converging on invalid negative numbers.

In addition, we note that the form input dynamics prior is not conditioned on the propagation encoder outputs, although we could in theory express it terms of its statistics (i.e. Equations (39) and (40)). This to avoid converging on negative values for variances, which can be obtained from the subtraction of positive constant $\boldsymbol{Q}^{'}$, although we revisit this in this form in the next section.

Neural Network Prior (VRNF-NN)

Despite the convenient tractable from of the Kalman filtering equations, this relies on the use of linear state space assumptions which might not be suitable for complex datasets. As such, we also consider the use of multilayer perceptrons (MLP($\cdot$)) to approximate the equations described in the previous section, conditioning it on the previous active encoder stage, i.e.:

Propagation:

$$\tilde{\boldsymbol{\beta}}_t^{'} = \text{MLP}_{\tilde{\boldsymbol{\beta}}'}(m(\boldsymbol{s}_{t-1})) \tag{51}$$

$$\tilde{\boldsymbol{\nu}}_t^{'} = \text{MLP}_{\tilde{\boldsymbol{\nu}}'}(V(\boldsymbol{s}_{t-1})) \tag{52}$$

Input Dynamics:

$$\tilde{\boldsymbol{\beta}}_t = \text{MLP}_{\tilde{\boldsymbol{\beta}}}(m(\tilde{\boldsymbol{s}}_t), \boldsymbol{u}_t) \tag{53}$$

$$\tilde{\boldsymbol{\nu}}_t = \text{MLP}_{\tilde{\boldsymbol{\nu}}}(V(\tilde{\boldsymbol{s}}_t), \boldsymbol{u}_t) \tag{54}$$

Error Correction:

$$\boldsymbol{\beta}_t = \text{MLP}_{\boldsymbol{\beta}}(m(\boldsymbol{s}_t), \boldsymbol{y}_t) \tag{55}$$

$$\boldsymbol{\nu}_t = \text{MLP}_{\boldsymbol{\nu}}(V(\boldsymbol{s}_t), \boldsymbol{y}_t) \tag{56}$$

Similar to the state transition functions used in [13], this can be interpreted as using MLPs to approximate the true Kalman filter functions for linear datasets, while also permitting the learning of more sophisticated non-linear models. All MLPs defined here use an ELU activation function for their hidden layer, fixing the hidden state size to be J. Furthermore, we use linear output layers for $\boldsymbol{\beta}$ MLPs, while passing that of $\boldsymbol{\nu}$ MLPs through a softplus activation function to maintain positivity.

B. Training Procedure for RNF

a) Training Details: During network calibration, trajectories were partitioned into segments of 50 time steps each – which were randomly combined to form minibatches during training. Also, networks were trained for up to a maximum of 100 epochs or convergence. For the electricity and volatility datasets, 50 iterations of random search were performed, using the grid found in Table V. 20 iterations of random search were used for the quote dataset, as the significantly larger dataset led to longer training times for a given set of hyperparameters.

TABLE V
RANDOM SEARCH GRID FOR HYPERPARAMETER OPTIMISATION

	Hyperparameter Ranges
Dropout Rate	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
State Size	5, 10, 25, 50, 100, 150
Minibatch Size	256, 512, 1024
Learning Rate	0.0001, 0.001, 0.01, 0.1, 1.0
Max Gradient Norm	0.0001, 0.001, 0.01, 0.1, 1.0, 10.0
Missing Rate	0.25, 0.5, 0.75

b) State Sizes: To ensure that consistency across all models used, we constrain both the memory state of the RNN and the latent variable modelled to have the same dimensionality – i.e. $J = \dim(\boldsymbol{s}_t) = \dim(\boldsymbol{x}_t)$ for the RNF. The exception is the DSSM, as the full covariance matrix of the Kalman filter would result in a prohibitive J^2 memory requirement if left unchecked. As such, we use the constraint where both the RNN and the Kalman filter to have the same memory capacity for the DSSM – i.e. $J = \dim(\boldsymbol{s}_t) = \dim(\boldsymbol{x}_t) + \dim(\boldsymbol{x}_t)^2$.

c) Dropout Application: Across all benchmarks, dropout was applied only onto the memory state of the RNNs ($\boldsymbol{h}_t$) in the standard fashion and not to latent states $\boldsymbol{x}_t$. For the LSTM, DeepAR Model and RNF, this corresponds to applying dropout masks to the outputs and internal states of the network. For the VRNN, DKF and DSSM, we apply dropout only to the inputs of the network – in line with [14] to maintain comparability to the encoder skipping in the VRNFs.

d) Artificial Missingness: Encoder skipping is restricted to only the VRNFs and standard RNF, controlled by the missing rates defined above.

e) Sample Generation: At prediction time, latent states for the VRNN, DKF and VRNFs are sampled as per the standard VAE case – using $L = 1$ during training, $L = 30$ for our validation error and $L = 100$ for at test time. Predictions from the DeepAR Model, DSSM and standard RNF, however, were obtained directly from the mean estimates, with that of the DSSM computed analytically using the Kalman filtering equations. While this differs slightly from the original paper [26], it also leads to improvements in the performance DSSM by avoiding sampling errors.

C. Description of Datasets

For the experiments in Section VI, we focus on the use of 3 real-world time series datasets, each containing over a million time steps per dataset. These use-cases help us evaluate performance for scenarios in which real-time predictions with RNNs are most beneficial – i.e. when the underlying dynamics is highly non-linear and trajectories are long.

Summary

Electricity: The UCI Individual Household Electric Power Consumption Dataset [47] is a time series of 7 different power consumption metrics measured at 1-min intervals for a single household between December 2006 and November 2010 – coming to a total of 2,075,259 time steps over 4 years. In our experiments, we treated active power as the main observation of interest, taking the remainder to be exogenous inputs into the RNNs.

Intraday Volatility: We compute 30-min realised variances [48] for a universe of 30 different stock indices – derived using 1-min index returns subsampled from Thomson Reuters Tick History Level 1 (TRTH L1) quote data. On the whole, the entire dataset contains 1,706,709 measurements across all indices, with each trajectory spanning 17 years on average. Given the strong evidence for the intraday periodicity of returns volatility [50], we also include the time-of-day as an additional exogenous input.

High-Frequency Stock Quotes: This dataset consists of extracted features from TRTH L1 stock quote data for Barclays (BARC.L) – specifically forecasting microprice returns [51] using volume imbalance as an input predictor – comprising a total of 29,321,946 time steps between 03 January 2017 to 29 December 2017. From [52], volume imbalance in the limit order book is a good predictor of the direction (sign) of the next liquidity taking order, and the price changes immediately after the arrival of a liquidity-taking order.

Electricity

a) Data Processing: The full trajectory was segmented into 3 portions, with the earliest 60% of measurements for training, the next 20% as a validation set, and the final 20% as an independent test set – with results reported in Section VI. All data sets were normalised to have zero mean and unit standard deviations, with normalising constants computed using the training set alone.

b) Summary Statistics: A list of summary statistics can be seen in Table VI.

TABLE VI
SUMMARY STATISTICS FOR ELECTRICITY DATASET

	Mean	S.D.	Min	Max
Active Power*	1.11	1.12	0.08	11.12
Reactive Power	0.12	0.11	0.00	1.39
Intensity	4.73	4.70	0.20	48.40
Voltage	240.32	3.33	223.49	252.72
Sub Metering 1	1.17	6.31	0.00	82.00
Sub Metering 2	1.42	6.26	0.00	78.00
Sub Metering 3	6.04	8.27	0.00	31.00

Intraday Volatility

c) Data Processing: From the 1-min index returns, realised variances were computed as:

$$r_k = \ln p_k - \ln p_{k-1}$$
$$y(t, 30) = \sum_{k=t-30}^{t} r_k^2, \tag{57}$$

where r_k is the 1-min index return at time k, $\ln p_k$ is the log price at k, and $y(t, 30)$ is the 30-min realised variance at time t.

Before computation, the data was cleaned by only considering prices during exchange hours to avoid spurious jumps. In addition, realised variances greater than 10 times the 200-step rolling standard deviation were removed and replaced by its previous value – so as to reduce the impact of outliers.

For the experiments in Section VI, data across all stock indices were grouped together for training and testing – using data prior to 2014 for training, data between 2014-2016 for validation and data from 2016 to 4 July 2018 for independent testing. Min-max normalisation was applied to the datasets, with time normalised by the maximum trading window of each exchange and realised variances by the max and min values of the training dataset.

d) Stock Index Identifiers (RICs):: AEX, AORD, BFX, BSESN, BVLG, BVSP, DJI, FCHI, FTMIB, FTSE, GDAXI, GSPTSE, HSI, IBEX, IXIC, KS11, KSE, MXX, N225, NSEI, OMXC20, OMXHPI, OMXSPI, OSEAX, RUT, SMSI, SPX, SSEC, SSMI, STOXX50E

e) Summary Statistics: A table of summary statistics can be found in Table VII and give an indication of the general ranges of trajectories.

TABLE VII
SUMMARY STATISTICS FOR VOLATILITY DATASET

	Mean	S. D.	Min	Max
Realised Variance*	0.0007	0.0017	0.0000	0.1013
Normalised Time	0.43	0.27	0.00	0.97

High-Frequency Stock Quotes

f) Input/Output Definitions: Microprice returns y_t are defined as:

$$p_t = \frac{V_a(t)p_b(t) + V_b(t)p_a(t)}{V_a(t) + V_b(t)}$$
$$y_t = \frac{p_t - p_{t-1}}{p_{t-1}}$$

Where $V_b(t)$ and $V_a(t)$ are the bid and ask volumes at time t respectively, $p_b(t)$ and $p_a(t)$ are the bid/ask prices, and p_t the microprice.

Volume imbalance I_t is then defined as:

$$I_t = \frac{V_b(t) - V_a(t)}{V_b(t) + V_a(t)}$$

g) Data Processing: From the raw Level 1 (best bid and ask prices and volumes) data from TRTH, we isolate measurements between 08.30 to 16.00 UK time, avoiding the effects of opening and closing auctions in our forecasts. Furthermore, microprice returns were also normalised using an exponentially weighting moving standard deviation with a half-life of 10,000 steps. We note that volume imbalance by definition is restricted to be $I_t \in [-1, 1]$, and hence does not require additional normalisation. Finally, the data was partitioned with training data from January to June, validation data from June to September, and the remainder for independent testing.

h) Summary Statistics: Basic statistics can be found in Table VIII, and give an indication of the range of different variables.

TABLE VIII
SUMMARY STATISTICS FOR QUOTE DATASET

	Mean	S. D.	Min	Max
Normalised Returns*	0.00	0.80	-117.72	117.13
Volume Imbalance	0.02	0.48	-1.00	1.00

Chapter 4

Incorporating Domain Knowledge With Hybrid Models

Publications Included

- **B. Lim**, S. Zohren, S. Roberts. *Enhancing Time Series Momentum Strategies Using Deep Neural Networks*. Journal of Financial Data Science (JFDS), 2019. Weblink: `https://arxiv.org/abs/1904.04912`.
- **B. Lim**, M. van der Schaar. *Disease-Atlas: Navigating Disease Trajectories with Deep Learning*. Proceedings of the Machine Learning for Healthcare Conference (MLHC), 2018. Weblink: `https://arxiv.org/abs/1803.10254`.

Enhancing Time Series Momentum Strategies Using Deep Neural Networks

Bryan Lim, Stefan Zohren, Stephen Roberts

***Abstract*—While time series momentum [1] is a well-studied phenomenon in finance, common strategies require the explicit definition of both a trend estimator and a position sizing rule. In this paper, we introduce Deep Momentum Networks – a hybrid approach which injects deep learning based trading rules into the volatility scaling framework of time series momentum. The model also simultaneously learns both trend estimation and position sizing in a data-driven manner, with networks directly trained by optimising the Sharpe ratio of the signal. Back-testing on a portfolio of 88 continuous futures contracts, we demonstrate that the Sharpe-optimised LSTM improved traditional methods by more than two times in the absence of transactions costs, and continue outperforming when considering transaction costs up to 2-3 basis points. To account for more illiquid assets, we also propose a turnover regularisation term which trains the network to factor in costs at run-time.**

I. Introduction

Momentum as a risk premium in finance has been extensively documented in the academic literature, with evidence of persistent abnormal returns demonstrated across a range of asset classes, prediction horizons and time periods [2, 3, 4]. Based on the philosophy that strong price trends have a tendency to persist, time series momentum strategies are typically designed to increase position sizes with large directional moves and reduce positions at other times. Although the intuition underpinning the strategy is clear, specific implementation details can vary widely between signals – with a plethora of methods available to estimate the magnitude of price trends [5, 6, 4] and map them to actual traded positions [7, 8, 9].

In recent times, deep neural networks have been increasingly used for time series prediction, outperforming traditional benchmarks in applications such as demand forecasting [10], medicine [11] and finance [12]. With the development of modern architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) [13], deep learning models have been favoured for their ability to build representations of a given dataset [14] – capturing temporal dynamics and cross-sectional relationships in a purely data-driven manner. The adoption of deep neural networks has also been facilitated by powerful open-source frameworks such as `TensorFlow` [15] and `PyTorch` [16] – which use automatic differentiation to compute gradients for backpropagation without having to explicitly derive them in advance. In turn, this flexibility has allowed deep neural networks to go beyond standard classification and regression models. For instance, the creation of hybrid methods that combine traditional time-series models with neural network components have been observed to outperform pure methods in either category [17] – e.g. the exponential smoothing RNN [18], autoregressive CNNs [19] and Kalman filter variants [20, 21] – while also making outputs easier to interpret by practitioners. Furthermore, these frameworks have also enabled the development of new loss functions for training neural networks, such as adversarial loss functions in generative adversarial networks (GANs) [22].

While numerous papers have investigated the use of machine learning for financial time series prediction, they typically focus on casting the underlying prediction problem as a standard regression or classification task [23, 24, 25, 12, 26, 19, 27] – with regression models forecasting expected returns, and classification models predicting the direction of future price movements. This approach, however, could lead to suboptimal performance in the context time-series momentum for several reasons. Firstly, sizing positions based on expected returns alone does not take risk characteristics into account – such as the volatility or skew of the predictive

B. Lim, S. Zohren and S. Roberts are with the Department of Engineering Science and the Oxford-Man Institute of Quantitative Finance, University of Oxford, Oxford, United Kingdom (email: bryan.lim@eng.ox.ac.uk, zohren@robots.ox.ac.uk, sjrob@robots.ox.ac.uk).

returns distribution — which could inadvertently expose signals to large downside moves. This is particularly relevant as raw momentum strategies without adequate risk adjustments, such as volatility scaling [7], are susceptible to large crashes during periods of market panic [28, 29]. Furthermore, even with volatility scaling – which leads to positively skewed returns distributions and long-option-like behaviour [30, 31] – trend following strategies can place more losing trades than winning ones and still be profitable on the whole – as they size up only into large but infrequent directional moves. As such, [32] argue that the fraction of winning trades is a meaningless metric of performance, given that it cannot be evaluated independently from the trading style of the strategy. Similarly, high classification accuracies may not necessarily translate into positive strategy performance, as profitability also depends on the magnitude of returns in each class. This is also echoed in betting strategies such as the Kelly criterion [33], which requires both win/loss probabilities and betting odds for optimal sizing in binomial games. In light of the deficiencies of standard supervised learning techniques, new loss functions and training methods would need to be explored for position sizing – accounting for trade-offs between risk and reward.

In this paper, we introduce a novel class of hybrid models that combines deep learning-based trading signals with the volatility scaling framework used in time series momentum strategies [8, 1] – which we refer to as the *Deep Momentum Networks* (DMNs). This improves existing methods from several angles. Firstly, by using deep neural networks to directly generate trading signals, we remove the need to manually specify both the trend estimator and position sizing methodology – allowing them to be learnt directly using modern time series prediction architectures. Secondly, by utilising automatic differentiation in existing backpropagation frameworks, we explicitly optimise networks for risk-adjusted performance metrics, i.e. the Sharpe ratio [34], improving the risk profile of the signal on the whole. Lastly, retaining a consistent framework with other momentum strategies also allows us to retain desirable attributes from previous works – specifically volatility scaling, which plays a critical role in the positive performance of time series momentum strategies [9]. This consistency also helps when making comparisons to existing methods, and facilitates the interpretation of different components of the overall signal by practitioners.

II. Related Works

A. Classical Momentum Strategies

Momentum strategies are traditionally divided into two categories – namely (multivariate) cross sectional momentum [35, 24] and (univariate) time series momentum [1, 8]. Cross sectional momentum strategies focus on the relative performance of securities against each other, buying relative winners and selling relative losers. By ranking a universe of stocks based on their past return and trading the top decile against the bottom decile, [35] find that securities that recently outperformed their peers over the past 3 to 12 months continue to outperform on average over the next month. The performance of cross sectional momentum has also been shown to be stable across time [36], and across a variety of markets and asset classes [4].

Time series momentum extends the idea to focus on an asset's own past returns, building portfolios comprising all securities under consideration. This was initially proposed by [1], who describe a concrete strategy which uses volatility scaling and trades positions based on the sign of returns over the past year – demonstrating profitability across 58 different liquid instruments *individually* over 25 years of data. Since then, numerous trading rules have been proposed – with various trend estimation techniques and methods map them to traded positions. For instance, [6] documents a wide range of linear and non-linear filters to measure trends and a statistic to test for its significance – although methods to size positions with these estimates are not directly discussed. [8] adopt a similar approach to [1], regressing the log price over the past 12 months against time and using the regression coefficient t-statistics to determine the direction of the traded position. While Sharpe ratios were comparable between the two, t-statistic based trend estimation led to a 66% reduction in portfolio turnover and consequently trading costs. More sophisticated trading rules are proposed in [4] and [37], taking volatility-normalised moving average convergence divergence (MACD) indicators

as inputs. Despite the diversity of options, few comparisons have been made between the trading rules themselves, offering little clear evidence or intuitive reasoning to favour one rule over the next. We hence propose the use of deep neural networks to generate these rules directly, avoiding the need for explicit specification. Training them based on risk-adjusted performance metrics, the networks hence learn optimal training rules directly from the data itself.

B. Deep Learning in Finance

Machine learning has long been used for financial time series prediction, with recent deep learning applications studying mid-price prediction using daily data [26], or using limit order book data in a high frequency trading setting [25, 12, 38]. While a variety of CNN and RNN models have been proposed, they typically frame the forecasting task as a classification problem, demonstrating the improved accuracy of their method in predicting the direction of the next price movement. Trading rules are then manually defined in relation to class probabilities – either by using thresholds on classification probabilities to determine when to initiate positions [26], or incorporating these thresholds into the classification problem itself by dividing price movements into buy, hold and sell classes depending on magnitude [12, 38]. In addition to restricting the universe of strategies to those which rely on high accuracy, further gains might be made by learning trading rules directly from the data and removing the need for manual specification – both of which are addressed in our proposed method.

Deep learning regression methods have also been considered in cross-sectional strategies [23, 24], ranking assets on the basis of expected returns over the next time period. Using a variety of linear, tree-based and neural network models [23] demonstrate the outperformance of non-linear methods, with deep neural networks – specifically 3-layer multilayer perceptrons (MLPs) – having the best out-of-sample predictive R^2. Machine learning portfolios were then built by ranking stocks on a monthly basis using model predictions, with the best strategy coming from a 4-layer MLP that trades the top decile against the bottom decile of predictions. In other works, [24] adopt a similar approach using autoencoder and denoising autoencoder architectures, incorporating volatility scaling into their model as well. While the results with basic deep neural networks are promising, they do not consider more modern architectures for time series prediction, such as the LSTM [39] and WaveNet [40] architectures which we evaluate for the DMN. Moreover, to the best of our knowledge, our paper is the first to consider the use of deep learning within the context of time series momentum strategies – opening up possibilities in an alternate class of signals.

Popularised by success of DeepMind's AlphaGo Zero [41], deep reinforcement learning (RL) has also gained much attention in recent times – prized for its ability to recommend path-dependent actions in dynamic environments. RL is particularly of interest within the context of optimal execution and automated hedging [42, 43] for example, where actions taken can have an impact on future states of the world (e.g. market impact). However, deep RL methods generally require a realistic simulation environment (for Q-learning or policy gradient methods), or model of the world (for model-based RL) to provide feedback to agents during training – both of which are difficult to obtain in practice.

III. Strategy Definition

Adopting the terminology of [8], the combined returns of a time series momentum (TSMOM) strategy can be expressed as below – characterised by a trading rule or signal $X_t \in [-1, 1]$:

$$r_{t,t+1}^{TSMOM} = \frac{1}{N_t} \sum_{i=1}^{N_t} X_t^{(i)} \frac{\sigma_{\text{tgt}}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)}. \tag{1}$$

Here $r_{t,t+1}^{TSMOM}$ is the realised return of the strategy from day t to $t+1$, N_t is the number of included assets at t, and $r_{t,t+1}^{(i)}$ is the one-day return of asset i. We set the annualised volatility target σ_{tgt} to be 15% and scale asset returns with an ex-ante volatility estimate $\sigma_t^{(i)}$ – computed using an exponentially weighted moving standard deviation with a 60-day span on $r_{t,t+1}^{(i)}$.

A. Standard Trading Rules

In traditional financial time series momentum strategies, the construction of a trading signal X_t is typically divided into two steps: 1) estimating

future trends based on past information, and 2) computing the actual positions to hold. We illustrate this in this section using two examples from the academic literature [1, 4], which we also include as benchmarks into our tests.

Moskowitz et al. 2012 [1]: In their original paper on time series momentum, a simple trading rule is adopted as below:

$$\textbf{Trend Estimation:} \quad Y_t^{(i)} = r_{t-252,t}^{(i)} \tag{2}$$

$$\textbf{Position Sizing:} \quad X_t^{(i)} = \text{sgn}(Y_t^{(i)}) \tag{3}$$

This broadly uses the past year's returns as a trend estimate for the next time step - taking a maximum long position when the expected trend is positive (i.e. $\text{sgn}(r_{t-252,t}^{(i)})$) and a maximum short position when negative.

Baz et al. 2015 [4]: In practice, more sophisticated methods can be used to compute $Y_t^{(i)}$ and $X_t^{(i)}$ – such as the model of [4] described below:

$$\textbf{Trend Estimation:} \quad Y_t^{(i)} = \frac{q_t^{(i)}}{\text{std}(z_{t-252:t}^{(i)})} \tag{4}$$

$$q_t^{(i)} = \text{MACD}(i,t,S,L) \ / \ \text{std}(p_{t-63:t}^{(i)}) \tag{5}$$

$$\text{MACD}(i,t,S,L) = m(i,S) - m(i,L). \tag{6}$$

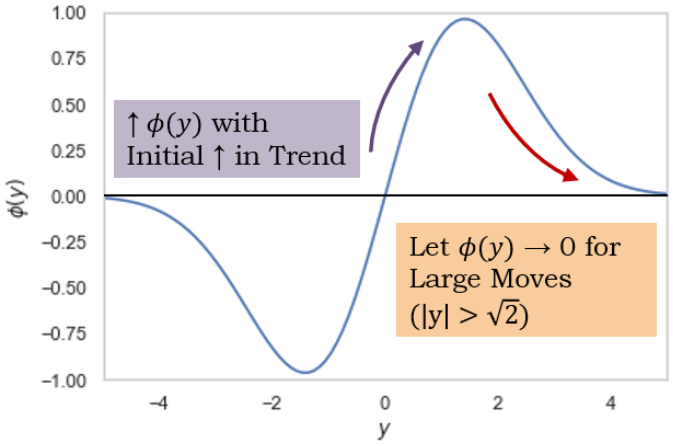
Here $\text{std}(p_{t-63:t}^{(i)})$ is the 63-day rolling standard deviation of asset i prices $p_{t-63:t}^{(i)} = [p_{t-63}^{(i)}, \ldots, p_t^{(i)}]$, $m(i,S)$ is the exponentially weighted moving average of asset i prices with a time-scale S that translates into a half-life of $HL = \log(0.5)/\log(1-\frac{1}{S})$. The moving average crossover divergence (MACD) signal is defined in relation to a short and a long time-scale S and L respectively.

The volatility-normalised MACD signal hence measures the strength of the trend, which is then translated in to a position size as below:

$$\textbf{Position Sizing:} \quad X_t^{(i)} = \phi(Y_t^{(i)}), \tag{7}$$

where $\phi(y) = \frac{y \exp(\frac{-y^2}{4})}{0.89}$. Plotting $\phi(y)$ in Exhibit 1, we can see that positions are increased until $|Y_t^{(i)}| = \sqrt{2} \approx 1.41$, before decreasing back to zero for larger moves. This allows the signal to reduces positions in instances where assets are overbought or oversold – defined to be when $|q_t^{(i)}|$ is observed to be larger than 1.41 times its past year's standard deviation.

Exhibit 1: Position Sizing Function $\phi(y)$



Increasing the complexity even further, multiple signals with different times-scales can also be averaged to give a final position:

$$\tilde{Y}_t^{(i)} = \sum_{k=1}^{3} Y_t^{(i)}(S_k, L_k), \tag{8}$$

where $Y_t^{(i)}(S_k, L_k)$ is as per Equation (4) with explicitly defined short and long time-scales – using $S_k \in \{8, 16, 32\}$ and $L_k \in \{24, 48, 96\}$ as defined in [4].

B. Machine Learning Extensions

As can be seen from Section III-A, many explicit design decisions are required to define a sophisticated time series momentum strategy. We hence start by considering how machine learning methods can be used to learn these relationships directly from data – alleviating the need for manual specification.

Standard Supervised Learning: In line with numerous previous works (see Section II-B), we can cast trend estimation as a standard regression or binary classification problem, with outputs:

$$\textbf{Trend Estimation:} \quad Y_t^{(i)} = f\left(\mathbf{u}_t^{(i)}; \boldsymbol{\theta}\right), \tag{9}$$

where $f(\cdot)$ is the output of the machine learning model, which takes in a vector of input features $\mathbf{u}_t^{(i)}$ and model parameters $\boldsymbol{\theta}$ to generate predictions. Taking volatility-normalised returns as targets, the following mean-squared error and binary cross-entropy losses can be used for training:

$$\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{\Omega}\left(Y_t^{(i)} - \frac{r_{t,t+1}^{(i)}}{\sigma_t^{(i)}}\right)^2 \quad (10)$$

$$\mathcal{L}_{\text{binary}}(\boldsymbol{\theta}) = -\frac{1}{M}\sum_{\Omega}\left\{\mathbb{I}\ \log\left(Y_t^{(i)}\right) + (1-\mathbb{I})\log\left(1-Y_t^{(i)}\right)\right\}, \quad (11)$$

where $\Omega = \left\{\left(Y_1^{(1)}, r_{1,2}^{(1)}/\sigma_t^{(1)}\right), \ \ldots\ , \left(Y_{T-1}^{(N)}, r_{T-1,T}^{(N)}/\sigma_{T-1}^{(N)}\right)\right\}$ is the set of all M possible prediction and target tuples across all N assets and T time steps. For the binary classification case, $\mathbb{I}$ is the indicator function $\mathbb{I}\left(r_{t,t+1}^{(i)}/\sigma_t^{(i)} > 0\right)$ – making $Y_t^{(i)}$ the estimated probability of a positive return.

This still leaves us to specify how trend estimates map to positions, and we do so using a similar form to Equation 3:

Position Sizing:

$$\text{Regression} \quad X_t^{(i)} = \text{sgn}(Y_t^{(i)}) \quad (12)$$

$$\text{Classification} \quad X_t^{(i)} = \text{sgn}(Y_t^{(i)} - 0.5) \quad (13)$$

As such, we take a maximum long position when the expected returns are positive in the regression case, or when the probability of a positive return is greater than 0.5 in the classification case.

Direct Outputs: An alternative approach is to use machine learning models to generate positions directly – simultaneously learning both trend estimation and position sizing in the same function, i.e.:

$$\textbf{Direct Outputs:} \quad X_t^{(i)} = f\left(\mathbf{u}_t^{(i)}; \boldsymbol{\theta}\right). \quad (14)$$

Given the lack of direct information on the optimal positions to hold at each step – which is required to produce labels for standard regression and classification models – calibration would hence need to be performed by directly optimising performance metrics. Specifically, we focus on optimising the average return and the annualised Sharpe ratio via the loss functions below:

$$\begin{aligned}\mathcal{L}_{\text{returns}}(\boldsymbol{\theta}) &= -\mu_R \\ &= -\frac{1}{M}\sum_{\Omega} R(i,t) \\ &= -\frac{1}{M}\sum_{\Omega} X_t^{(i)}\ \frac{\sigma_{\text{tgt}}}{\sigma_t^{(i)}}\ r_{t,t+1}^{(i)} \quad (15)\end{aligned}$$

$$\mathcal{L}_{\text{sharpe}}(\boldsymbol{\theta}) = -\frac{\mu_R \times \sqrt{252}}{\sqrt{\left(\sum_{\Omega} R(i,t)^2\right)/M - \mu_R^2}} \quad (16)$$

where μ_R is the average return over Ω, and $R(i,t)$ is the return captured by the trading rule for asset i at time t.

IV. Deep Momentum Networks

In this section, we examine a variety of architectures that can be used in Deep Momentum Networks – all of which can be easily reconfigured to generate the predictions described in Section III-B. This is achieved by implementing the models using the `Keras` API in `Tensorflow` [15], where output activation functions can be flexibly interchanged to generate the predictions of different types (e.g. expected returns, binary probabilities, or direct positions). Arbitrary loss functions can also be defined for direct outputs, with gradients for backpropagation being easily computed using the built-in libraries for automatic differentiation.

A. Network Architectures

Lasso Regression: In the simplest case, a standard linear model could be used to generate predictions as below:

$$Z_t^{(i)} = g\left(\mathbf{w}^T \mathbf{u}_{t-\tau:t}^{(i)} + b\right), \quad (17)$$

where $Z_t^{(i)} \in \left\{X_t^{(i)}, Y_t^{(i)}\right\}$ depending on the prediction task, $\mathbf{w}$ is a weight vector for the linear model, and b is a bias term. Here $g(\cdot)$ is a activation function which depends on the specific prediction type – linear for standard regression, sigmoid for binary classification, and tanh-function for direct outputs.

Additional regularisation is also provided during training by augmenting the various loss functions to include an additional L_1 regulariser as below:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \alpha||\mathbf{w}||_1, \tag{18}$$

where $\mathcal{L}(\boldsymbol{\theta})$ corresponds to one of the loss functions described in Section III-B, $||\mathbf{w}||_1$ is the L_1 norm of $\mathbf{w}$, and α is a constant term which we treat as an additional hyperparameter. To incorporate recent history into predictions as well, we concatenate inputs over the past τ-days into a single input vector – i.e. $\mathbf{u}^{(i)}_{t-\tau:t} = [\mathbf{u}^{(i)\ T}_{t-\tau}, \ldots, \mathbf{u}^{(i)\ T}_{t}]^T$. This was fixed to be $\tau = 5$ days for tests in Section V.

Multilayer Perceptron (MLP): Increasing the degree of model complexity slightly, a 2-layer neural network can be used to incorporated non-linear effects:

$$\mathbf{h}^{(i)}_t = \tanh\left(\mathbf{W}_h \mathbf{u}^{(i)}_{t-\tau:t} + \mathbf{b}_h\right) \tag{19}$$

$$Z^{(i)}_t = g\left(\mathbf{W}_z \mathbf{h}^{(i)}_t + \mathbf{b}_z\right), \tag{20}$$

where $\mathbf{h}^{(i)}_t$ is the hidden state of the MLP using an internal tanh activation function, $\tanh(\cdot)$, and $\mathbf{W}_{\cdot}$ and $\mathbf{b}_{\cdot}$ are layer weight matrices and biases respectively.

WaveNet: More modern techniques such as convolutional neural networks (CNNs) have been used in the domain of time series prediction – particularly in the form of autoregressive architectures e.g. [19]. These typically take the form of 1D causal convolutions, sliding convolutional filters across time to extract useful representations which are then aggregated in higher layers of the network. To increase the size of the receptive field – or the length of history fed into the CNN – dilated CNNs such as WaveNet [40] have been proposed, which skip over inputs at intermediate levels with a predetermined dilation rate. This allows it to effectively increase the amount of historical information used by the CNN without a large increase in computational cost. Let us consider a dilated convolutional layer with residual connections take the form below:

$$\psi(\mathbf{u}) = \underbrace{\tanh(\mathbf{W}\mathbf{u}) \odot \sigma(\mathbf{V}\mathbf{u})}_{\text{Gated Activation}} + \underbrace{\mathbf{A}\mathbf{u} + \mathbf{b}}_{\text{Skip Connection}} . \tag{21}$$

Here $\mathbf{W}$ and $\mathbf{V}$ are weight matrices associated with the gated activation function, and $\mathbf{A}$ and $\mathbf{b}$ are the weights and biases used to transform the $\mathbf{u}$ to match dimensionality of the layer outputs for the skip connection. The equations for WaveNet architecture used in our investigations can then be expressed as:

$$\mathbf{s}^{(i)}_{\text{weekly}}(t) = \psi(\mathbf{u}^{(i)}_{t-5:t}) \tag{22}$$

$$\mathbf{s}^{(i)}_{\text{monthly}}(t) = \psi\left(\begin{bmatrix} \mathbf{s}^{(i)}_{\text{weekly}}(t) \\ \mathbf{s}^{(i)}_{\text{weekly}}(t-5) \\ \mathbf{s}^{(i)}_{\text{weekly}}(t-10) \\ \mathbf{s}^{(i)}_{\text{weekly}}(t-15) \end{bmatrix}\right) \tag{23}$$

$$\mathbf{s}^{(i)}_{\text{quarterly}}(t) = \psi\left(\begin{bmatrix} \mathbf{s}^{(i)}_{\text{monthly}}(t) \\ \mathbf{s}^{(i)}_{\text{monthly}}(t-21) \\ \mathbf{s}^{(i)}_{\text{monthly}}(t-42) \end{bmatrix}\right). \tag{24}$$

Here each intermediate layer $\mathbf{s}^{(i)}_{\cdot}(t)$ aggregates representations at weekly, monthly and quarterly frequencies respectively. Intermediate layers are then concatenated at each layer before passing through a 2-layer MLP to generate outputs, i.e.:

$$\mathbf{s}^{(i)}_t = \begin{bmatrix} \mathbf{s}^{(i)}_{\text{weekly}}(t) \\ \mathbf{s}^{(i)}_{\text{monthly}}(t) \\ \mathbf{s}^{(i)}_{\text{quarterly}}(t) \end{bmatrix} \tag{25}$$

$$\mathbf{h}^{(i)}_t = \tanh(\mathbf{W}_h \mathbf{s}^{(i)}_t + \mathbf{b}_h) \tag{26}$$

$$Z^{(i)}_t = g\left(\mathbf{W}_z \mathbf{h}^{(i)}_t + \mathbf{b}_z\right). \tag{27}$$

State sizes for each intermediate layers $\mathbf{s}^{(i)}_{\text{weekly}}(t)$, $\mathbf{s}^{(i)}_{\text{monthly}}(t)$, $\mathbf{s}^{(i)}_{\text{quarterly}}(t)$ and the MLP hidden state $\mathbf{h}^{(i)}_t$ are fixed to be the same, allowing us to use a single hyperparameter to define the architecture. To independently evaluate the performance of CNN and RNN architectures, the above also excludes the LSTM block (i.e. the context stack) described in [40], focusing purely on the merits of the dilated CNN model.

Long Short-term Memory (LSTM): Traditionally used in sequence prediction for natural language processing, recurrent neural networks – specifically long short-term memory (LSTM) architectures [39] – have been increasing used in time series prediction

tasks. The equations for the LSTM in our model are provided below:

$$\mathbf{f}_t^{(i)} = \sigma(\mathbf{W}_f \mathbf{u}_t^{(i)} + \mathbf{V}_f \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_f) \quad (28)$$

$$\mathbf{i}_t^{(i)} = \sigma(\mathbf{W}_i \mathbf{u}_t^{(i)} + \mathbf{V}_i \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_i) \quad (29)$$

$$\mathbf{o}_t^{(i)} = \sigma(\mathbf{W}_o \mathbf{u}_t^{(i)} + \mathbf{V}_o \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_o) \quad (30)$$

$$\mathbf{c}_t^{(i)} = \mathbf{f}_t^{(i)} \odot \mathbf{c}_{t-1}^{(i)} + \mathbf{i}_t^{(i)} \odot \tanh(\mathbf{W}_c \mathbf{u}_t^{(i)} + \mathbf{V}_c \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_c) \quad (31)$$

$$\mathbf{h}_t^{(i)} = \mathbf{o}_t^{(i)} \odot \tanh(\mathbf{c}_t^{(i)}) \quad (32)$$

$$Z_t^{(i)} = g\left(\mathbf{W}_z \mathbf{h}_t^{(i)} + \mathbf{b}_z\right), \quad (33)$$

where $\odot$ is the Hadamard (element-wise) product, $\sigma(.)$ is the sigmoid activation function, $\mathbf{W}_.$ and $\mathbf{V}_.$ are weight matrices for the different layers, $\mathbf{f}_t^{(i)}, \mathbf{i}_t^{(i)}, \mathbf{o}_t^{(i)}$ correspond to the forget, input and output gates respectively, $\mathbf{c}_t^{(i)}$ is the cell state, and $\mathbf{h}_t^{(i)}$ is the hidden state of the LSTM. From these equations, we can see that the LSTM uses the cell state as a compact summary of past information, controlling memory retention with the forget gate and incorporating new information via the input gate. As such, the LSTM is able to learn representations of long-term relationships relevant to the prediction task – sequentially updating its internal memory states with new observations at each step.

B. Training Details

Model calibration was undertaken using minibatch stochastic gradient descent with the `Adam` optimiser [44], based on the loss functions defined in Section III-B. Backpropagation was performed up to a maximum of 100 training epochs using 90% of a given block of training data, and the most recent 10% retained as a validation dataset. Validation data is then used to determine convergence – with early stopping triggered when the validation loss has not improved for 25 epochs – and to identify the optimal model across hyperparameter settings. Hyperparameter optimisation was conducted using 50 iterations of random search, with full details provided in Appendix B. For additional information on the deep neural network calibration, please refer to [13].

Dropout regularisation [45] was a key feature to avoid overfitting in the neural network models – with dropout rates included as hyperparameters during training. This was applied to the inputs and hidden state for the MLP, as well as the inputs, Equation (22), and outputs, Equation (26), of the convolutional layers in the WaveNet architecture. For the LSTM, we adopted the same dropout masks as in [46] – applying dropout to the RNN inputs, recurrent states and outputs.

V. Performance Evaluation

A. Overview of Dataset

The predictive performance of the different architectures was evaluated via a backtest using 88 ratio-adjusted continuous futures contracts downloaded from the Pinnacle Data Corp CLC Database [47]. These contracts spanned across a variety of asset classes – including commodities, fixed income and currency futures – and contained prices from 1990 to 2015. A full breakdown of the dataset can be found in Appendix A.

B. Backtest Description

Throughout our backtest, the models were recalibrated from scratch every 5 years – re-running the entire hyperparameter optimisation procedure using all data available up to the recalibration point. Model weights were then fixed for signals generated over the next 5 year period, ensuring that tests were performed out-of-sample.

For the Deep Momentum Networks, we incorporate a series of useful features adopted by standard time series momentum strategies in Section III-A to generate predictions at each step:

1) *Normalised Returns* – Returns over the past day, 1-month, 3-month, 6-month and 1-year periods are used, normalised by a measure of daily volatility scaled to an appropriate time scale. For instance, normalised annual returns were taken to be $r_{t-252,t}^{(i)}/(\sigma_t^{(i)}\sqrt{252})$.
2) *MACD Indicators* – We also include the MACD indicators – i.e. trend estimates $Y_t^{(i)}$ – as in Equation (4), using the same short time-scales $S_k \in \{8, 16, 32\}$ and long time-scales $L_k \in \{24, 48, 96\}$.

For comparisons against traditional time series momentum strategies, we also incorporate the following reference benchmarks:

1) Long Only with Volatility Scaling $(X_t^{(i)} = 1)$
2) Sgn(Returns) – Moskowitz et al. 2012 [1]
3) MACD Signal – Baz et al. 2015 [4]

Finally, performance was judged based on the following metrics:

1) *Profitability* – Expected returns ($\mathbb{E}$[Returns]) and the percentage of positive returns observed across the test period.
2) *Risk* – Daily volatility (Vol.), downside deviation and the maximum drawdown (MDD) of the overall portfolio.
3) *Performance Ratios* – Risk adjusted performance was measured by the Sharpe ratio $\left(\frac{\mathbb{E}[\text{Returns}]}{\text{Vol.}}\right)$, Sortino ratio $\left(\frac{\mathbb{E}[\text{Returns}]}{\text{Downside Deviation}}\right)$ and Calmar ratio $\left(\frac{\mathbb{E}[\text{Returns}]}{\text{MDD}}\right)$, as well as the average profit over the average loss $\left(\frac{\text{Ave. P}}{\text{Ave. L}}\right)$.

C. Results and Discussion

Aggregating the out-of-sample predictions from 1995 to 2015, we compute performance metrics for both the strategy returns based on Equation (1) (Exhibit 2), as well as that for portfolios with an additional layer of volatility scaling – which brings overall strategy returns to match the 15% volatility target (Exhibit 3). Given the large differences in returns volatility seen in Table 2, this rescaling also helps to facilitate comparisons between the cumulative returns of different strategies – which are plotted for various loss functions in Exhibit 4. We note that strategy returns in this section are computed in the absence of transaction costs, allowing us to focus on the raw predictive ability of the models themselves. The impact of transaction costs is explored further in Section VI, where we undertake a deeper analysis of signal turnover. More detailed results can also be found in Appendix C, which echo the findings below.

Focusing on the raw signal outputs, the Sharpe ratio-optimised LSTM outperforms all benchmarks as expected, improving the best neural network model (Sharpe-optimised MLP) by 44% and the best reference benchmark (Sgn(Returns)) by more than two times. In conjunction with Sharpe ratio improvements to both the linear and MLP models, this highlights the benefits of using models which capture non-linear relationships, and have access to more time history via an internal memory state. Additional model complexity, however, does not necessarily lead to better predictive performance, as demonstrated by the underperformance of WaveNet compared to both the reference benchmarks and simple linear models. Part of this can be attributed to the difficulties in tuning models with multiple design parameters - for instance, better results could possibly achieved by using alternative dilation rates, number of convolutional layers, and hidden state sizes in Equations (22) to (24) for the WaveNet. In contrast, only a single design parameter is sufficient to specify the hidden state size in both the MLP and LSTM models. Analysing the relative performance within each model class, we can see that models which directly generate positions perform the best – demonstrating the benefits of simultaneous learning both trend estimation and position sizing functions. In addition, with the exception of a slight decrease in the MLP, Sharpe-optimised models outperform returns-optimised ones, with standard regression and classification benchmarks taking third and fourth place respectively.

From Exhibit 3, while the addition of volatility scaling at the portfolio level improved performance ratios on the whole, it had a larger beneficial effect on machine learning models compared to the reference benchmarks – propelling Sharpe-optimised MLPs to outperform returns-optimised ones, and even leading to Sharpe-optimised linear models beating reference benchmarks. From a risk perspective, we can see that both volatility and downside deviation also become a lot more comparable, with the former hovering close to 15.5% and the latter around 10%. However, Sharpe-optimised LSTMs still retained the lowest MDD across all models, with superior risk-adjusted performance ratios across the board. Referring to the cumulative returns plots for the rescaled portfolios in Exhibit 4, the benefits of direct outputs with Sharpe ratio optimisation can also be observed – with larger cumulative returns observed for linear, MLP and LSTM models compared to the reference benchmarks. Furthermore, we note the general underperformance of models which use standard regression and classification methods for trend estimation – hinting at the difficulties faced in selecting an appropriate position sizing function, and in optimising models to generate positions without accounting for risk. This is particularly relevant for binary classification

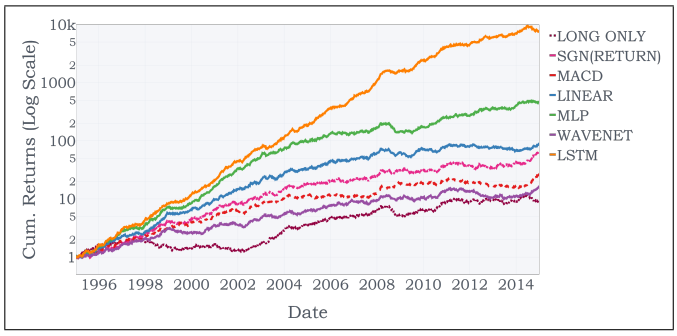
Exhibit 2: Performance Metrics – Raw Signal Outputs

	E[Return]	Vol.	Downside Deviation	MDD	Sharpe	Sortino	Calmar	% of +ve Returns	$\frac{\text{Ave. P}}{\text{Ave. L}}$
Reference									
Long Only	0.039	0.052	0.035	0.167	0.738	1.086	0.230	53.8%	0.970
Sgn(Returns)	0.054	0.046	0.032	0.083	1.192	1.708	0.653	54.8%	1.011
MACD	0.030	0.031	0.022	0.081	0.976	1.356	0.371	53.9%	1.015
Linear									
Sharpe	0.041	0.038	0.028	0.119	1.094	1.462	0.348	54.9%	0.997
Ave. Returns	0.047	0.045	0.031	0.164	1.048	1.500	0.287	53.9%	1.022
MSE	0.049	0.047	0.032	0.164	1.038	1.522	0.298	54.3%	1.000
Binary	0.013	0.044	0.030	0.167	0.295	0.433	0.078	50.6%	1.028
MLP									
Sharpe	0.044	0.031	0.025	0.154	1.383	1.731	0.283	56.0%	1.024
Ave. Returns	**0.064***	0.043	0.030	0.161	1.492	2.123	0.399	55.6%	1.031
MSE	0.039	0.046	0.032	0.166	0.844	1.224	0.232	52.7%	1.035
Binary	0.003	0.042	0.028	0.233	0.080	0.120	0.014	50.8%	0.981
WaveNet									
Sharpe	0.030	0.035	0.026	0.101	0.854	1.167	0.299	53.5%	1.008
Ave. Returns	0.032	0.040	0.028	0.113	0.788	1.145	0.281	53.8%	0.980
MSE	0.022	0.042	0.028	0.134	0.536	0.786	0.166	52.4%	0.994
Binary	0.000	0.043	0.029	0.313	0.011	0.016	0.001	50.2%	0.995
LSTM									
Sharpe	0.045	**0.016***	**0.011***	**0.021***	**2.804***	**3.993***	**2.177***	**59.6%***	**1.102***
Ave. Returns	0.054	0.046	0.033	0.164	1.165	1.645	0.326	54.8%	1.003
MSE	0.031	0.046	0.032	0.163	0.669	0.959	0.189	52.8%	1.003
Binary	0.012	0.039	0.026	0.255	0.300	0.454	0.046	51.0%	1.012

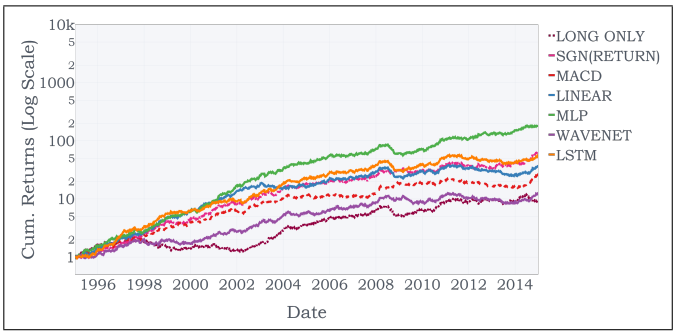
Exhibit 3: Performance Metrics – Rescaled to Target Volatility

	E[Return]	Vol.	Downside Deviation	MDD	Sharpe	Sortino	Calmar	% of +ve Returns	$\frac{\text{Ave. P}}{\text{Ave. L}}$
Reference									
Long Only	0.117	0.154	0.102	0.431	0.759	1.141	0.271	53.8%	0.973
Sgn(Returns)	0.215	0.154	0.102	0.264	1.392	2.108	0.815	54.8%	1.041
MACD	0.172	0.155	0.106	0.317	1.111	1.622	0.543	53.9%	1.031
Linear									
Sharpe	0.232	0.155	0.103	0.303	1.496	2.254	0.765	54.9%	1.056
Ave. Returns	0.189	0.154	0.100	0.372	1.225	1.893	0.507	53.9%	1.047
MSE	0.186	0.154	**0.099***	0.365	1.211	1.889	0.509	54.3%	1.025
Binary	0.051	0.155	0.103	0.558	0.332	0.496	0.092	50.6%	1.033
MLP									
Sharpe	0.312	0.154	0.102	0.335	2.017	3.042	0.930	56.0%	1.104
Ave. Returns	0.266	0.154	**0.099***	0.354	1.731	2.674	0.752	55.6%	1.065
MSE	0.156	0.154	**0.099***	0.371	1.017	1.582	0.422	52.7%	1.062
Binary	0.017	0.154	0.102	0.661	0.108	0.162	0.025	50.8%	0.986
WaveNet									
Sharpe	0.148	0.155	0.103	0.349	0.956	1.429	0.424	53.5%	1.018
Ave. Returns	0.136	0.154	0.101	0.356	0.881	1.346	0.381	53.8%	0.993
MSE	0.084	**0.153***	0.101	0.459	0.550	0.837	0.184	52.4%	0.995
Binary	0.007	0.155	0.103	0.779	0.045	0.068	0.009	50.2%	1.001
LSTM									
Sharpe	**0.451***	0.155	0.105	**0.209***	**2.907***	**4.290***	**2.159***	**59.6%***	**1.113***
Ave. Returns	0.208	0.154	0.102	0.365	1.349	2.045	0.568	54.8%	1.028
MSE	0.121	0.154	0.100	0.362	0.791	1.211	0.335	52.8%	1.020
Binary	0.075	0.155	**0.099***	0.682	0.486	0.762	0.110	51.0%	1.043

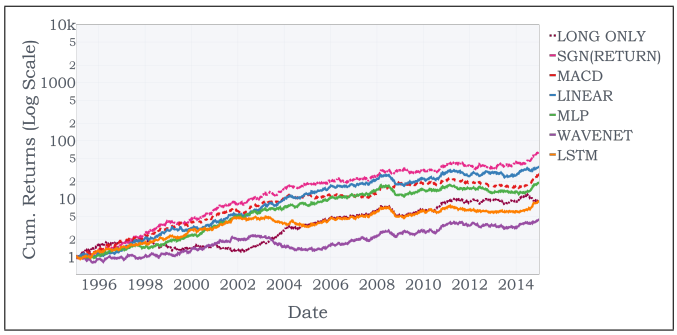
Exhibit 4: Cumulative Returns - Rescaled to Target Volatility



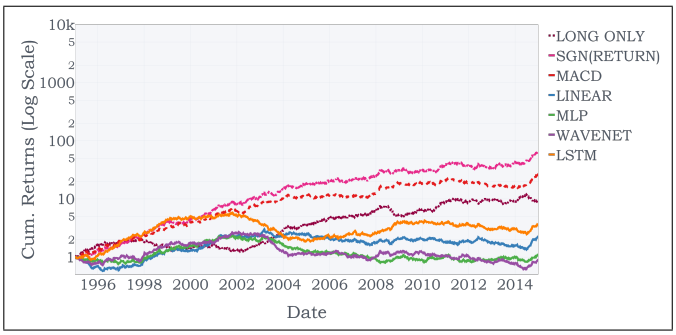
(a) Sharpe Ratio



(b) Average Returns



(c) MSE

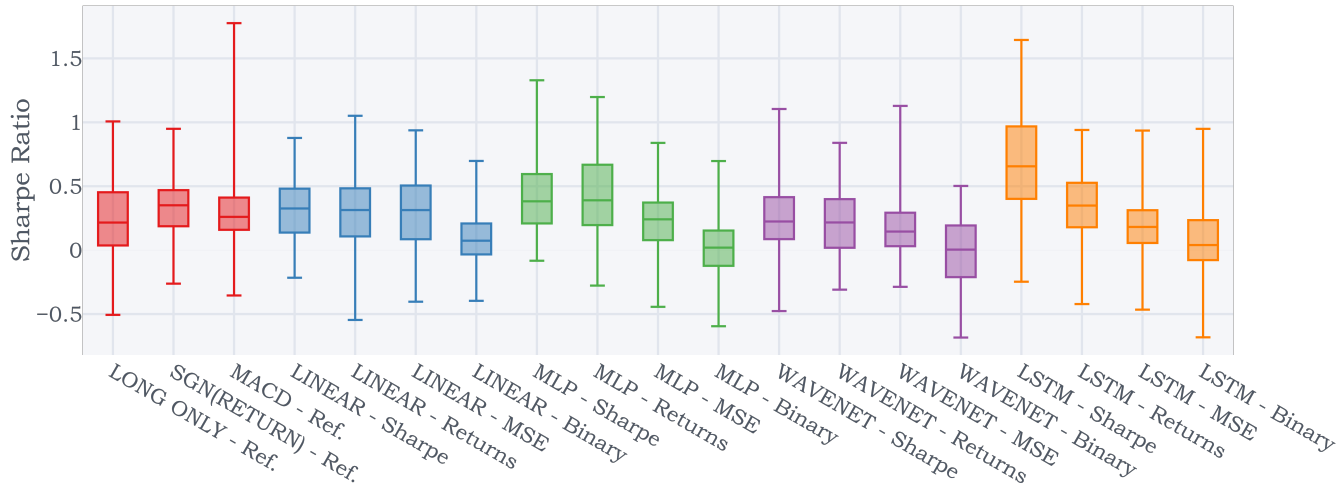


(d) Binary

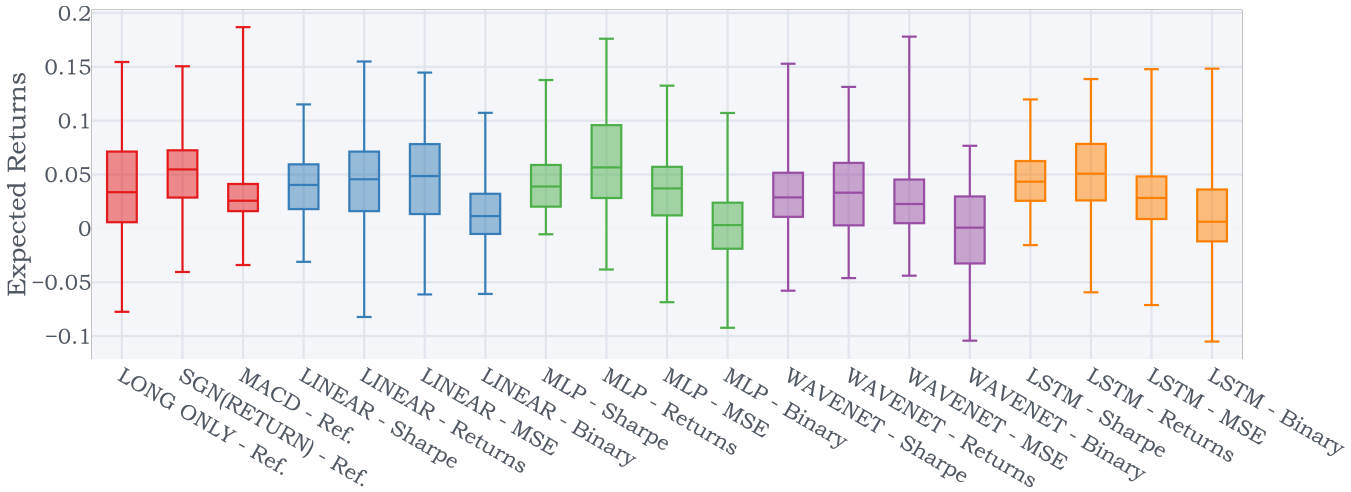
methods, which produce relatively flat equity lines and underperform reference benchmarks in general. Some of these poor results can be explained by the implicit decision threshold adopted. From the percentage of positive returns captured in Exhibit 3, most binary classification models have about a 50% accuracy which, while expected of a classifier with a 0.5 probability threshold, is far below the accuracies seen in other benchmarks. Furthermore, performance is made worse by the fact that the model's magnitude of gains versus losses $\left(\frac{\text{Ave. P}}{\text{Ave. L}}\right)$ is much smaller than competing methods – with average loss magnitudes even outweighing profits for the MLP classifier $\left(\frac{\text{Ave. P}}{\text{Ave. L}} = 0.986\right)$. As such, these observations lend support to the direct generation of positions sizes with machine learning methods, given the multiple considerations (e.g. decision thresholds and profit/loss magnitudes) that would be required to incorporate standard supervising learning methods into a profitable trading strategy.

Strategy performance could also be aided by diversification across a range of assets, particularly when the correlation between signals is low. Hence, to evaluate the raw quality of the underlying signal, we investigate the performance constituents of the time series momentum portfolios – using box plots for a variety of performance metrics, plotting the minimum, lower quartile, median, upper quartile, and maximum values across individual futures contracts. We present in Exhibit 5 plots of one metric per category in Section V-B, although similar results can be seen for other performance ratios are documented in Appendix C. In general, the Sharpe ratio plots in Exhibit 5a echo previous findings, with direct output methods performing better than indirect trend estimation models. However, as seen in Exhibit 5c, this is mainly attributable to significant reduction in signal volatility for the Sharpe-optimised methods, despite a comparable range of average returns in Exhibit 5b. The benefits of retaining the volatility scaling can also be observed, with individual signal volatility capped near the target across all methods – even with a naive $\text{sgn}(.)$ position sizer. As such, the combination of volatility scaling, direct outputs and Sharpe ratio optimisation were all key to performance gains in Deep Momentum Networks.

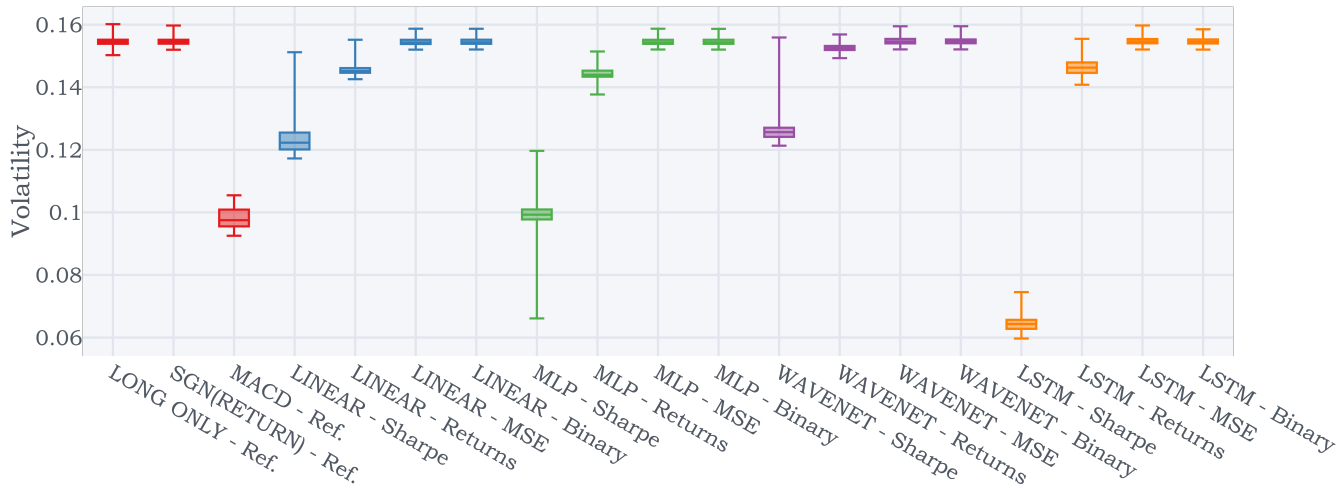
Exhibit 5: Performance Across Individual Assets



(a) Sharpe Ratio



(b) Average Returns



(c) Volatility

VI. Turnover Analysis

To investigate how transaction costs affect strategy performance, we first analyse the daily position changes of the signal – characterised for asset i by daily turnover $\zeta_t^{(i)}$ as defined in [8]:

$$\zeta_t^{(i)} = \sigma_{\text{tgt}} \left| \frac{X_t^{(i)}}{\sigma_t^{(i)}} - \frac{X_{t-1}^{(i)}}{\sigma_{t-1}^{(i)}} \right| \tag{34}$$

Which is broadly proportional to the volume of asset i traded on day t with reference to the updated portfolio weights.

Exhibit 6a shows the average strategy turnover across all assets from 1995 to 2015, focusing on positions generated by the raw signal outputs. As the box plots are charted on a logarithm scale, we note that while the machine learning-based models have a similar turnover, they also trade significantly more than the reference benchmarks – approximately 10 times more compared to the Long Only benchmark. This is also reflected in Exhibit 6a which compares the average daily returns against the average daily turnover – with ratios from machine learning models lying close to the x-axis.

To concretely quantify the impact of transaction costs on performance, we also compute the ex-cost Sharpe ratios – using the rebalancing costs defined in [8] to adjust our returns for a variety of transaction cost assumptions . For the results in Exhibit 7, the top of each bar chart marks the maximum cost-free Sharpe ratio of the strategy, with each coloured block denoting the Sharpe ratio reduction for the corresponding cost assumption. In line with the turnover analysis, the reference benchmarks demonstrate the most resilience to high transaction costs (up to 5bps), with the profitability across most machine learning models persisting only up to 4bps. However, we still obtain higher cost-adjusted Sharpe ratios with the Sharpe-optimised LSTM for up to 2-3 bps, demonstrating its suitability for trading more liquid instruments.

A. Turnover Regularisation

One simple way to account for transaction costs is to use cost-adjusted returns $\tilde{r}_{t,t+1}^{TSMOM}$ directly during training, augmenting the strategy returns defined in Equation (1) as below:

$$\tilde{r}_{t,t+1}^{TSMOM} = \frac{\sigma_{\text{tgt}}}{N_t} \sum_{i=1}^{N_t} \left(\frac{X_t^{(i)}}{\sigma_t^{(i)}} \, r_{t,t+1}^{(i)} - c \left| \frac{X_t^{(i)}}{\sigma_t^{(i)}} - \frac{X_{t-1}^{(i)}}{\sigma_{t-1}^{(i)}} \right| \right), \tag{35}$$

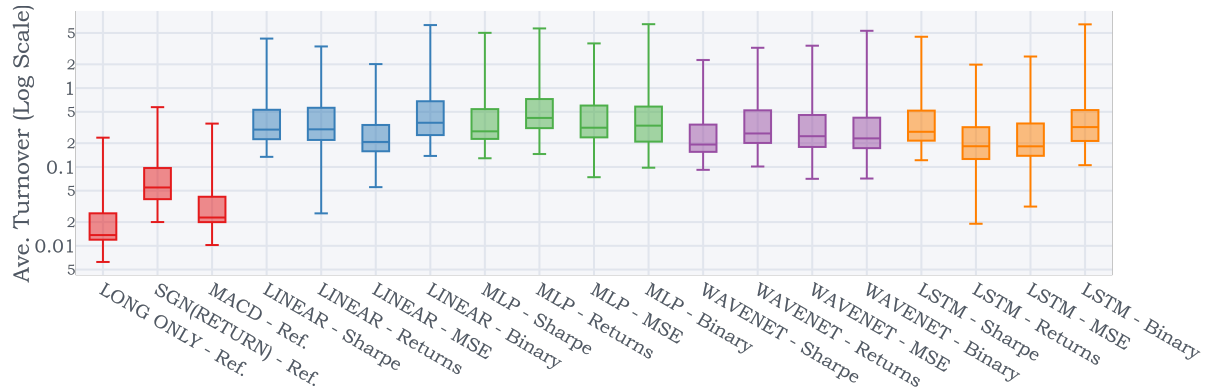
where c is a constant reflecting transaction cost assumptions. As such, using $\tilde{r}_{t,t+1}^{TSMOM}$ in Sharpe ratio loss functions during training corresponds to optimising the ex-cost risk-adjusted returns, and $c \left| \frac{X_t^{(i)}}{\sigma_t^{(i)}} - \frac{X_{t-1}^{(i)}}{\sigma_{t-1}^{(i)}} \right|$ can also be interpreted as a regularisation term for turnover.

Given that the Sharpe-optimised LSTM is still profitable in the presence of small transactions costs, we seek to quantify the effectiveness of turnover regularisation when costs are prohibitively high – considering the extreme case where $c = 10$bps in our investigation. Tests were focused on the Sharpe-optimised LSTM with and without the turnover regulariser (LSTM + Reg. for the former) – including the additional portfolio level volatility scaling to bring signal volatilities to the same level. Based on the results in Exhibit 8, we can see that the turnover regularisation does help improve the LSTM in the presence of large costs, leading to slightly better performance ratios when compared to the reference benchmarks.

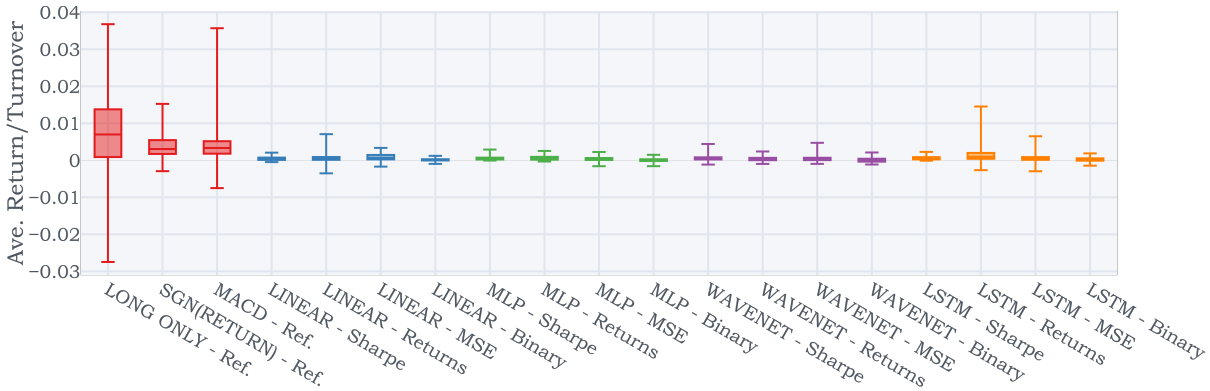
VII. Conclusions

We introduce Deep Momentum Networks – a hybrid class of deep learning models which retain the volatility scaling framework of time series momentum strategies while using deep neural networks to output position targeting trading signals. Two approaches to position generation were evaluated here. Firstly, we cast trend estimation as a standard supervised learning problem – using machine learning models to forecast the expected asset returns or probability of a positive return at the next time step – and apply a simple maximum long/short trading rule based on the direction of the next return. Secondly, trading rules were directly generated as outputs from the model, which we calibrate by maximising the Sharpe ratio or average strategy return. Testing this on a universe of continuous futures contracts, we demonstrate clear improvements in risk-adjusted performance by calibrating models with the Sharpe

Exhibit 6: Turnover Analysis



(a) Average Strategy Turnover



(b) Average Returns / Average Turnover

Exhibit 7: Impact of Transaction Costs on Sharpe Ratio

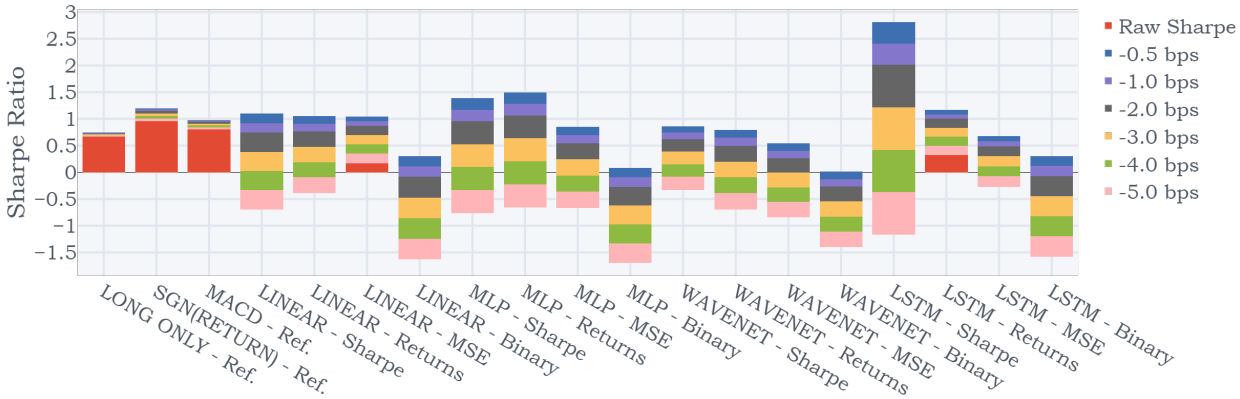


Exhibit 8: Performance Metrics with Transaction Costs ($c = 10$bps)

	E[Return]	**Vol.**	**Downside Deviation**	**MDD**	**Sharpe**	**Sortino**	**Calmar**	**% of +ve Returns**	**Ave. P / Ave. L**
Long Only	0.097	**0.154***	0.103	0.482	0.628	0.942	0.201	53.3%	0.970
Sgn(Returns)	0.133	**0.154***	**0.102***	0.373	0.861	1.296	0.356	53.3%	1.011
MACD	0.111	0.155	0.106	0.472	0.719	1.047	0.236	52.5%	**1.020***
LSTM	-0.833	0.157	0.114	1.000	-5.313	-7.310	-0.833	33.9%	0.793
LSTM + Reg.	**0.141***	**0.154***	**0.102***	**0.371***	**0.912***	**1.379***	**0.379***	**53.4%***	1.014

ratio – where the LSTM model achieved best results. Incorporating transaction costs, the Sharpe-optimised LSTM outperforms benchmarks up to 2-3 basis points of costs, demonstrating its suitability for trading more liquid assets. To accommodate high costs settings, we introduce a turnover regulariser to use during training, which was shown to be effective even in extreme scenarios (i.e. $c = 10$bps).

Future work includes extensions of the framework presented here to incorporate ways to deal better with non-stationarity in the data, such as using the recently introduced Recurrent Neural Filters [48]. Another direction of future work focuses on the study of time series momentum at the microstructure level.

VIII. Acknowledgements

We would like to thank Anthony Ledford, James Powrie and Thomas Flury for their interesting comments as well the Oxford-Man Institute of Quantitative Finance for financial support.

References

[1] T. J. Moskowitz, Y. H. Ooi, and L. H. Pedersen, "Time series momentum," *Journal of Financial Economics*, vol. 104, no. 2, pp. 228 – 250, 2012, Special Issue on Investor Sentiment.

[2] B. Hurst, Y. H. Ooi, and L. H. Pedersen, "A century of evidence on trend-following investing," *The Journal of Portfolio Management*, vol. 44, no. 1, pp. 15–29, 2017.

[3] Y. Lempérière, C. Deremble, P. Seager, M. Potters, and J.-P. Bouchaud, "Two centuries of trend following," *Journal of Investment Strategies*, vol. 3, no. 3, pp. 41–61, 2014.

[4] J. Baz, N. Granger, C. R. Harvey, N. Le Roux, and S. Rattray, "Dissecting investment strategies in the cross section and time series," *SSRN*, 2015. [Online]. Available: https://ssrn.com/abstract=2695101

[5] A. Levine and L. H. Pedersen, "Which trend is your friend," *Financial Analysts Journal*, vol. 72, no. 3, 2016.

[6] B. Bruder, T.-L. Dao, J.-C. Richard, and T. Roncalli, "Trend filtering methods for momentum strategies," *SSRN*, 2013. [Online]. Available: https://ssrn.com/abstract=2289097

[7] A. Y. Kim, Y. Tse, and J. K. Wald, "Time series momentum and volatility scaling," *Journal of Financial Markets*, vol. 30, pp. 103 – 124, 2016.

[8] N. Baltas and R. Kosowski, "Demystifying time-series momentum strategies: Volatility estimators, trading rules and pairwise correlations," *SSRN*, 2017. [Online]. Available: https://ssrn.com/abstract=2140091

[9] C. R. Harvey, E. Hoyle, R. Korgaonkar, S. Rattray, M. Sargaison, and O. van Hemert, "The impact of volatility targeting," *SSRN*, 2018. [Online]. Available: https://ssrn.com/abstract=3175538

[10] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," in *Time Series Workshop – International Conference on Machine Learning (ICML)*, 2017.

[11] B. Lim and M. van der Schaar, "Disease-atlas: Navigating disease trajectories using deep learning," in *Proceedings of the 3rd Machine Learning for Healthcare Conference (MLHC)*, ser. Proceedings of Machine Learning Research, vol. 85, 2018, pp. 137–160.

[12] Z. Zhang, S. Zohren, and S. Roberts, "DeepLOB: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, 2019.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[14] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[15] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Autodiff Workshop – Conference on Neural Information Processing (NIPS)*, 2017.

[17] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The M4 competition: Results, findings, conclusion and way forward," *International Journal of Forecasting*, vol. 34, no. 4, pp. 802 – 808, 2018.

[18] S. Smyl, J. Ranganathan, , and A. Pasqua. (2018) M4 forecasting competition: Introducing a new hybrid es-rnn model. [Online]. Available: https://eng.uber.com/m4-forecasting-competition/

[19] M. Binkowski, G. Marti, and P. Donnat, "Autoregressive convolutional neural networks for asynchronous time series," in

Proceedings of the 35th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 580–589.

[20] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018.

[21] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.

[23] S. Gu, B. T. Kelly, and D. Xiu, "Empirical asset pricing via machine learning," *Chicago Booth Research Paper No. 18-04; 31st Australasian Finance and Banking Conference 2018*, 2017. [Online]. Available: https://ssrn.com/abstract=3159577

[24] S. Kim, "Enhancing the momentum strategy through deep regression," *Quantitative Finance*, vol. 0, no. 0, pp. 1–13, 2019.

[25] J. Sirignano and R. Cont, "Universal features of price formation in financial markets: Perspectives from deep learning," *SSRN*, 2018. [Online]. Available: https://ssrn.com/abstract=3141294

[26] S. Ghoshal and S. Roberts, "Thresholded ConvNet ensembles: Neural networks for technical forecasting," in *Data Science in Fintech Workshop – Conference on Knowledge Discover and Data Mining (KDD)*, 2018.

[27] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLOS ONE*, vol. 12, no. 7, pp. 1–24, 2017.

[28] P. Barroso and P. Santa-Clara, "Momentum has its moments," *Journal of Financial Economics*, vol. 116, no. 1, pp. 111 – 120, 2015.

[29] K. Daniel and T. J. Moskowitz, "Momentum crashes," *Journal of Financial Economics*, vol. 122, no. 2, pp. 221 – 247, 2016.

[30] R. Martins and D. Zou, "Momentum strategies offer a positive point of skew," *Risk Magazine*, 2012.

[31] P. Jusselin, E. Lezmi, H. Malongo, C. Masselin, T. Roncalli, and T.-L. Dao, "Understanding the momentum risk premium: An in-depth journey through trend-following strategies," *SSRN*, 2017. [Online]. Available: https://ssrn.com/abstract=3042173

[32] M. Potters and J.-P. Bouchaud, "Trend followers lose more than they gain," *Wilmott Magazine*, 2016.

[33] L. M. Rotando and E. O. Thorp, "The Kelly criterion and the stock market," *The American Mathematical Monthly*, vol. 99, no. 10, pp. 922–931, 1992.

[34] W. F. Sharpe, "The sharpe ratio," *The Journal of Portfolio Management*, vol. 21, no. 1, pp. 49–58, 1994.

[35] N. Jegadeesh and S. Titman, "Returns to buying winners and selling losers: Implications for stock market efficiency," *The Journal of Finance*, vol. 48, no. 1, pp. 65–91, 1993.

[36] ——, "Profitability of momentum strategies: An evaluation of alternative explanations," *The Journal of Finance*, vol. 56, no. 2, pp. 699–720, 2001.

[37] J. Rohrbach, S. Suremann, and J. Osterrieder, "Momentum and trend following trading strategies for currencies revisited - combining academia and industry," *SSRN*, 2017. [Online]. Available: https://ssrn.com/abstract=2949379

[38] Z. Zhang, S. Zohren, and S. Roberts, "BDLOB: Bayesian deep convolutional neural networks for limit order books," in *Bayesian Deep Learning Workshop – Conference on Neural Information Processing (NeurIPS)*, 2018.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[41] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–, 2017.

[42] P. N. Kolm and G. Ritter, "Dynamic replication and hedging: A reinforcement learning approach," *The Journal of Financial Data Science*, vol. 1, no. 1, pp. 159–171, 2019.

[43] H. Bühler, L. Gonon, J. Teichmann, and B. Wood, "Deep Hedging," *arXiv e-prints*, p. arXiv:1802.03042, 2018.

[44] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[46] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

[47] "Pinnacle Data Corp. CLC Database," https://pinnacledata2.com/clc.html.

[48] B. Lim, S. Zohren, and S. Roberts, "Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction," *arXiv e-prints*, p. arXiv:1901.08096, 2019.

APPENDIX

A. Dataset Details

From the full 98 ratio-adjusted continuous futures contracts in the Pinnacle Data Corp CLC Database, we extract 88 which have $< 10\%$ of its data missing – with a breakdown by asset class below:

1) Commodities:

Identifier	Description
BC	BRENT CRUDE OIL, composite
BG	BRENT GASOIL, comp.
BO	SOYBEAN OIL
CC	COCOA
CL	CRUDE OIL
CT	COTTON #2
C_	CORN
DA	MILK III, Comp.
FC	FEEDER CATTLE
GC	GOLD (COMMEX)
GI	GOLDMAN SAKS C. I.
HG	COPPER
HO	HEATING OIL #2
JO	ORANGE JUICE
KC	COFFEE
KW	WHEAT, KC
LB	LUMBER
LC	LIVE CATTLE
LH	LIVE HOGS
MW	WHEAT, MINN
NG	NATURAL GAS
NR	ROUGH RICE
O_	OATS
PA	PALLADIUM
PL	PLATINUM
RB	RBOB GASOLINE
SB	SUGAR #11
SI	SILVER (COMMEX)
SM	SOYBEAN MEAL
S_	SOYBEANS
W_	WHEAT, CBOT
ZA	PALLADIUM, electronic
ZB	RBOB, Electronic
ZC	CORN, Electronic
ZF	FEEDER CATTLE, Electronic
ZG	GOLD, Electronic
ZH	HEATING OIL, electronic
ZI	SILVER, Electronic
ZK	COPPER, electronic
ZL	SOYBEAN OIL, Electronic
ZM	SOYBEAN MEAL, Electronic
ZN	NATURAL GAS, electronic
ZO	OATS, Electronic
ZP	PLATINUM, electronic
ZR	ROUGH RICE, Electronic
ZS	SOYBEANS, Electronic
ZT	LIVE CATTLE, Electronic
ZU	CRUDE OIL, Electronic
ZW	WHEAT, Electronic
ZZ	LEAN HOGS, Electronic

2) Equities:

Identifier	Description
AX	GERMAN DAX INDEX
CA	CAC40 INDEX
EN	NASDAQ, MINI
ER	RUSSELL 2000, MINI
ES	S & P 500, MINI
HS	HANG SENG
LX	FTSE 100 INDEX
MD	S&P 400 (Mini electronic)
SC	S & P 500, composite
SP	S & P 500, day session
XU	DOW JONES EUROSTOXX50
XX	DOW JONES STOXX 50
YM	Mini Dow Jones ($5.00)

3) Fixed Income:

Identifier	Description
AP	AUSTRALIAN PRICE INDEX
DT	EURO BOND (BUND)
FA	T-NOTE, 5yr day session
FB	T-NOTE, 5yr composite
GS	GILT, LONG BOND
TA	T-NOTE, 10yr day session
TD	T-NOTES, 2yr day session
TU	T-NOTES, 2yr composite
TY	T-NOTE, 10yr composite
UA	T-BONDS, day session
UB	EURO BOBL
US	T-BONDS, composite

4) FX:

Identifier	Description
AD	AUSTRALIAN $$, day session
AN	AUSTRALIAN $$, composite
BN	BRITISH POUND, composite
CB	CANADIAN 10YR BOND
CN	CANADIAN $$, composite
DX	US DOLLAR INDEX
FN	EURO, composite
FX	EURO, day session
JN	JAPANESE YEN, composite
MP	MEXICAN PESO
NK	NIKKEI INDEX
SF	SWISS FRANC, day session
SN	SWISS FRANC, composite

To reduce the impact of outliers, we also winsorise the data by capping/flooring it to be within 5 times its exponentially weighted moving (EWM) standard deviations from its EWM average – computed using a 252-day half life.

Exhibit 9: Hyperparameter Search Range

Hyperparameters	**Random Search Grid**	**Notes**
Dropout Rate	0.1, 0.2, 0.3, 0.4, 0.5	Neural Networks Only
Hidden Layer Size	5, 10, 20, 40, 80	Neural Networks Only
Minibatch Size	256, 512, 1024, 2048	
Learning Rate	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}	
Max Gradient Norm	10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}	
L1 Regularisation Weight (α)	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}	Lasso Regression Only

B. Hyperparameter Optimisation

Hyperparameter optimisation was applied using 50 iterations of random search, with the full search grid documented in Exhibit 9, with the models fully recalibrated every 5 years using all available data up to that point. For LSTM-based models, time series were subdivided into trajectories of 63 time steps ($\approx$ 3 months), with the LSTM unrolled across the length of the trajectory during backpropagation.

C. Additional Results

In addition to the selected results in Section V, we also present a full list of results was presented for completeness – echoing the key findings reported in the discuss. Detailed descriptions of the plots and tables can be found below:

1) Cross-Validation Performance: The testing procedure in Section V can also be interpreted as a cross-validation approach – splitting the original dataset into six 5-year blocks (1990-2015), calibrating using an expanding window of data, and testing out-of-sample on the next block outside the training set. As such, for consistency with machine learning literature, we present our results in a cross validation format as well – reporting the average value across all blocks $\pm$ 2 standard deviations. Furthermore, this also gives an indication of how signal performance varies across the various time periods.

- **Exhibit 10** – Cross-validation results for raw signal outputs.
- **Exhibit 11** – Cross-validation results for signals which have been rescaled to target volatility at the portfolio level.

2) Metrics Across Individual Assets: We also provide additional plots on performance of other risk metrics and performance ratios across individual assets, as described below:

- **Exhibit 12** – Full performance ratios (Sharpe, Sortino, Calmar).
- **Exhibit 13** – Box plots including additional risk metrics (volatility, downside deviation and maximum drawdown). We note that our findings for other risk metrics are similar to volatility – with Sharpe-optimised models lowering risk across different methods.

Exhibit 10: Cross-Validation Performance – Raw Signal Outputs

	E[Return]	**Vol.**	**Downside Deviation**	**MDD**
Reference				
Long Only	0.043 ± 0.028	0.054 ± 0.016	0.037 ± 0.013	0.116 ± 0.091
Sgn(Returns)	0.047 ± 0.051	0.046 ± 0.012	0.032 ± 0.007	0.067 ± 0.041
MACD	0.026 ± 0.032	0.032 ± 0.008	0.023 ± 0.007	0.054 ± 0.048
Linear				
Sharpe	0.034 ± 0.030	0.039 ± 0.028	0.028 ± 0.020	0.072 ± 0.096
Ave. Returns	0.033 ± 0.031	0.046 ± 0.025	0.031 ± 0.018	0.110 ± 0.114
MSE	0.047 ± 0.038	0.049 ± 0.019	0.033 ± 0.013	0.100 ± 0.121
Binary	0.012 ± 0.028	0.045 ± 0.011	0.031 ± 0.009	0.109 ± 0.045
MLP				
Sharpe	0.038 ± 0.027	0.030 ± 0.041	0.021 ± 0.028	0.062 ± 0.160
Ave. Returns	**0.056 ± 0.046***	0.044 ± 0.024	0.030 ± 0.017	0.075 ± 0.150
MSE	0.037 ± 0.051	0.048 ± 0.021	0.032 ± 0.015	0.109 ± 0.134
Binary	-0.004 ± 0.028	0.042 ± 0.007	0.028 ± 0.006	0.111 ± 0.079
WaveNet				
Sharpe	0.030 ± 0.030	0.038 ± 0.019	0.027 ± 0.015	0.069 ± 0.055
Ave. Returns	0.034 ± 0.043	0.042 ± 0.003	0.030 ± 0.003	0.088 ± 0.062
MSE	0.024 ± 0.046	0.043 ± 0.010	0.030 ± 0.010	0.102 ± 0.056
Binary	-0.009 ± 0.023	0.043 ± 0.008	0.030 ± 0.008	0.159 ± 0.107
LSTM				
Sharpe	0.045 ± 0.030	**0.017 ± 0.004***	**0.012 ± 0.003***	**0.019 ± 0.005***
Ave. Returns	0.045 ± 0.050	0.048 ± 0.018	0.034 ± 0.011	0.104 ± 0.119
MSE	0.023 ± 0.037	0.048 ± 0.022	0.033 ± 0.017	0.116 ± 0.082
Binary	-0.005 ± 0.088	0.042 ± 0.003	0.027 ± 0.006	0.151 ± 0.211

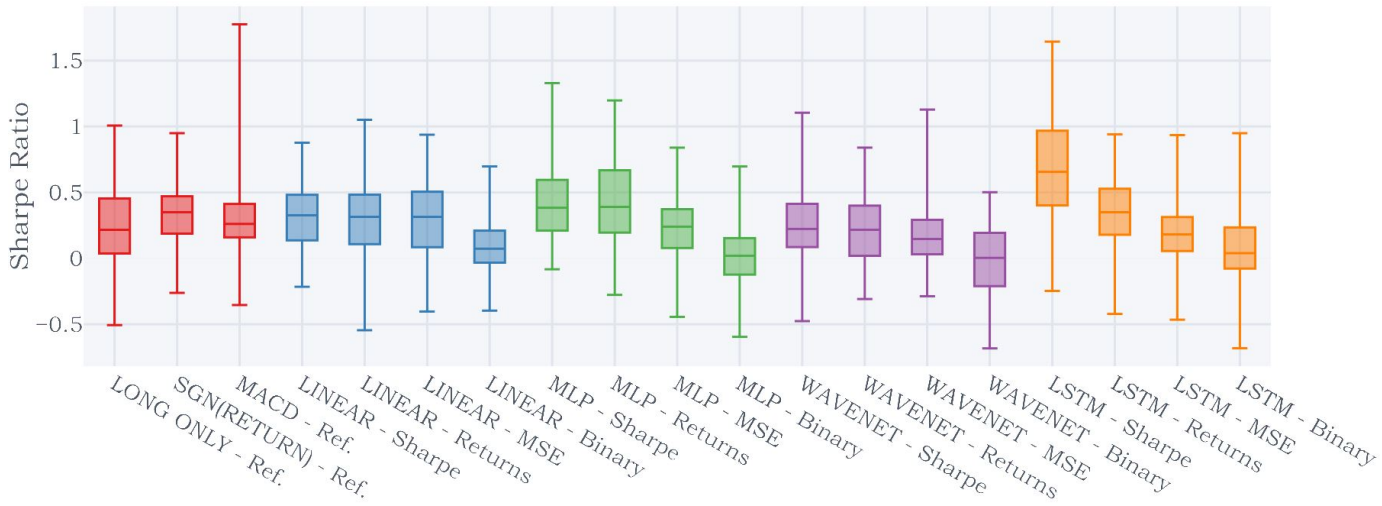
	Sharpe	**Sortino**	**Calmar**	**Fraction of +ve Returns**	**$\frac{\text{Ave. P}}{\text{Ave. L}}$**
Reference					
Long Only	0.839 ± 0.786	1.258 ± 1.262	0.420 ± 0.490	0.546 ± 0.025	0.956 ± 0.135
Sgn(Returns)	1.045 ± 1.230	1.528 ± 1.966	0.864 ± 1.539	0.543 ± 0.061	1.002 ± 0.067
MACD	0.839 ± 1.208	1.208 ± 1.817	0.625 ± 1.033	0.532 ± 0.030	1.016 ± 0.079
Linear					
Sharpe	1.025 ± 1.530	1.451 ± 2.154	0.800 ± 1.772	0.544 ± 0.042	1.000 ± 0.100
Ave. Returns	0.757 ± 0.833	1.150 ± 1.378	0.397 ± 0.686	0.530 ± 0.009	1.005 ± 0.100
MSE	1.012 ± 1.126	1.532 ± 1.811	0.708 ± 1.433	0.540 ± 0.024	1.008 ± 0.096
Binary	0.288 ± 0.729	0.434 ± 1.113	0.123 ± 0.313	0.506 ± 0.027	1.024 ± 0.051
MLP					
Sharpe	1.669 ± 2.332	2.420 ± 3.443	1.665 ± 2.738	0.554 ± 0.063	1.069 ± 0.151
Ave. Returns	1.415 ± 1.781	2.127 ± 2.996	1.520 ± 2.761	0.553 ± 0.043	1.022 ± 0.134
MSE	0.821 ± 1.334	1.270 ± 2.160	0.652 ± 1.684	0.525 ± 0.025	1.036 ± 0.127
Binary	-0.099 ± 0.648	-0.180 ± 0.956	-0.013 ± 0.304	0.500 ± 0.042	0.986 ± 0.064
WaveNet					
Sharpe	0.780 ± 0.538	1.118 ± 0.854	0.477 ± 0.610	0.535 ± 0.022	0.990 ± 0.094
Ave. Returns	0.809 ± 1.113	1.160 ± 1.615	0.501 ± 1.036	0.543 ± 0.059	0.963 ± 0.069
MSE	0.513 ± 0.991	0.744 ± 1.477	0.276 ± 0.509	0.527 ± 0.033	0.979 ± 0.077
Binary	-0.220 ± 0.523	-0.329 ± 0.768	-0.043 ± 0.145	0.499 ± 0.011	0.969 ± 0.043
LSTM					
Sharpe	**2.781 ± 2.081***	**3.978 ± 3.160***	**2.488 ± 1.921***	**0.593 ± 0.054***	**1.104 ± 0.199***
Ave. Returns	0.961 ± 1.268	1.397 ± 1.926	0.679 ± 1.552	0.547 ± 0.039	0.972 ± 0.118
MSE	0.451 ± 0.526	0.668 ± 0.812	0.184 ± 0.170	0.520 ± 0.026	0.996 ± 0.048
Binary	-0.114 ± 2.147	-0.191 ± 3.435	0.227 ± 1.241	0.495 ± 0.077	1.002 ± 0.077

Exhibit 11: Cross-Validation Performance – Rescaled to Target Volatility

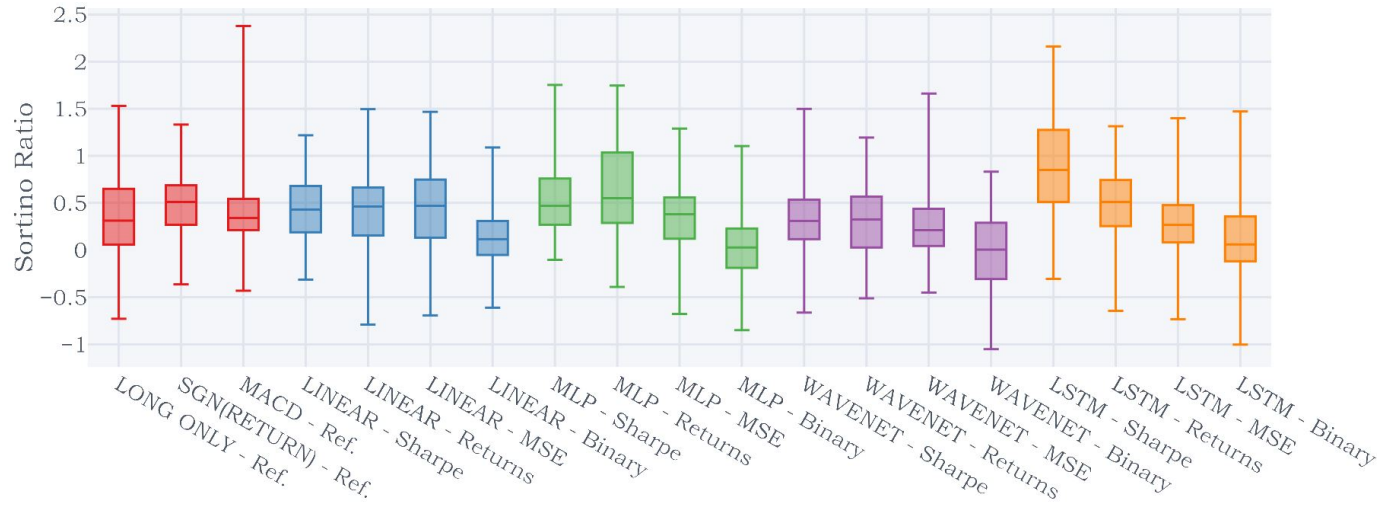
	E[Return]	**Vol.**	**Downside Deviation**	**MDD**
Reference				
Long Only	0.131 ± 0.142	0.154 ± 0.001	0.104 ± 0.014	0.304 ± 0.113
Sgn(Returns)	0.186 ± 0.184	0.154 ± 0.002	0.101 ± 0.012	0.194 ± 0.126
MACD	0.140 ± 0.166	0.154 ± 0.002	0.105 ± 0.010	0.243 ± 0.129
Linear				
Sharpe	0.182 ± 0.273	0.155 ± 0.003	0.105 ± 0.007	0.232 ± 0.175
Ave. Returns	0.127 ± 0.141	0.154 ± 0.003	0.101 ± 0.009	0.318 ± 0.177
MSE	0.170 ± 0.189	0.154 ± 0.003	**0.099 ± 0.006***	0.256 ± 0.221
Binary	0.049 ± 0.170	0.155 ± 0.002	0.104 ± 0.013	0.351 ± 0.114
MLP				
Sharpe	0.271 ± 0.375	0.154 ± 0.008	0.104 ± 0.000	0.186 ± 0.259
Ave. Returns	0.233 ± 0.270	0.154 ± 0.003	0.101 ± 0.010	0.194 ± 0.277
MSE	0.148 ± 0.178	0.154 ± 0.003	0.100 ± 0.009	0.268 ± 0.262
Binary	-0.011 ± 0.117	0.154 ± 0.002	0.102 ± 0.018	0.377 ± 0.221
WaveNet				
Sharpe	0.131 ± 0.103	0.154 ± 0.002	0.104 ± 0.009	0.254 ± 0.164
Ave. Returns	0.142 ± 0.196	0.154 ± 0.002	0.103 ± 0.003	0.262 ± 0.204
MSE	0.087 ± 0.150	**0.153 ± 0.003***	0.101 ± 0.009	0.307 ± 0.247
Binary	-0.030 ± 0.099	0.155 ± 0.001	0.105 ± 0.006	0.485 ± 0.283
LSTM				
Sharpe	**0.435 ± 0.342***	0.155 ± 0.002	0.108 ± 0.012	**0.164 ± 0.077***
Ave. Returns	0.157 ± 0.202	**0.153 ± 0.002***	0.102 ± 0.011	0.285 ± 0.196
MSE	0.087 ± 0.091	0.154 ± 0.003	0.100 ± 0.006	0.310 ± 0.130
Binary	-0.008 ± 0.332	0.155 ± 0.002	0.100 ± 0.009	0.428 ± 0.495

	Sharpe	**Sortino**	**Calmar**	**Fraction of +ve Returns**	$\frac{\text{Ave. P}}{\text{Ave. L}}$
Reference					
Long Only	0.847 ± 0.915	1.287 ± 1.475	0.445 ± 0.579	0.546 ± 0.025	0.958 ± 0.164
Sgn(Returns)	1.213 ± 1.205	1.856 ± 1.944	1.098 ± 1.658	0.543 ± 0.061	1.028 ± 0.070
MACD	0.911 ± 1.086	1.361 ± 1.733	0.643 ± 0.958	0.532 ± 0.030	1.023 ± 0.074
Linear					
Sharpe	1.176 ± 1.772	1.752 ± 2.615	1.060 ± 2.376	0.544 ± 0.042	1.025 ± 0.139
Ave. Returns	0.826 ± 0.914	1.287 ± 1.504	0.471 ± 0.777	0.530 ± 0.009	1.016 ± 0.116
MSE	1.101 ± 1.220	1.729 ± 2.037	0.890 ± 1.787	0.540 ± 0.024	1.022 ± 0.116
Binary	0.321 ± 1.105	0.509 ± 1.720	0.169 ± 0.585	0.506 ± 0.027	1.031 ± 0.127
MLP					
Sharpe	1.757 ± 2.405	2.623 ± 3.626	2.091 ± 3.474	0.554 ± 0.063	1.085 ± 0.176
Ave. Returns	1.516 ± 1.764	2.336 ± 2.923	1.771 ± 2.889	0.553 ± 0.043	1.038 ± 0.141
MSE	0.960 ± 1.163	1.510 ± 1.927	0.864 ± 1.999	0.525 ± 0.025	1.059 ± 0.103
Binary	-0.071 ± 0.756	-0.140 ± 1.133	0.000 ± 0.372	0.500 ± 0.042	0.991 ± 0.072
WaveNet					
Sharpe	0.849 ± 0.663	1.270 ± 1.060	0.575 ± 0.680	0.535 ± 0.022	1.000 ± 0.121
Ave. Returns	0.920 ± 1.271	1.376 ± 1.915	0.738 ± 1.591	0.543 ± 0.059	0.979 ± 0.066
MSE	0.565 ± 0.972	0.854 ± 1.513	0.364 ± 0.665	0.527 ± 0.033	0.986 ± 0.081
Binary	-0.196 ± 0.641	-0.298 ± 0.946	-0.044 ± 0.206	0.499 ± 0.011	0.974 ± 0.067
LSTM					
Sharpe	**2.803 ± 2.195***	**4.084 ± 3.469***	**2.887 ± 3.030***	**0.593 ± 0.054***	**1.106 ± 0.216***
Ave. Returns	1.023 ± 1.312	1.564 ± 2.131	0.706 ± 1.440	0.547 ± 0.039	0.980 ± 0.127
MSE	0.563 ± 0.580	0.865 ± 0.901	0.284 ± 0.269	0.520 ± 0.026	1.014 ± 0.016
Binary	-0.050 ± 2.152	-0.122 ± 3.381	0.190 ± 1.152	0.495 ± 0.077	1.012 ± 0.048

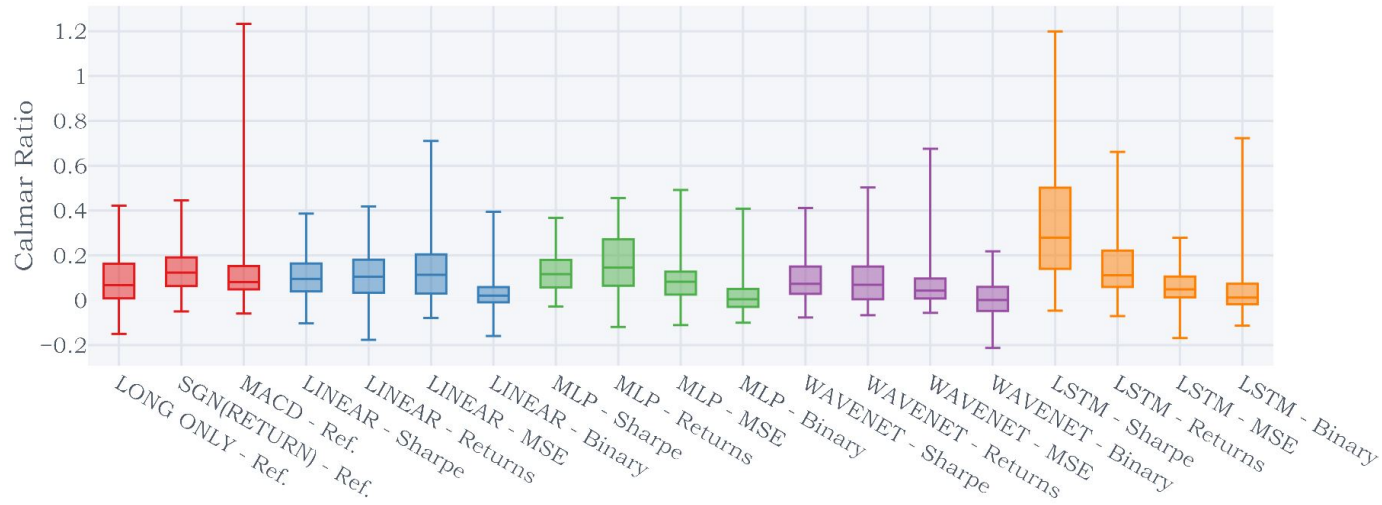
Exhibit 12: Performance Ratios Across Individual Assets



(a) Sharpe Ratio

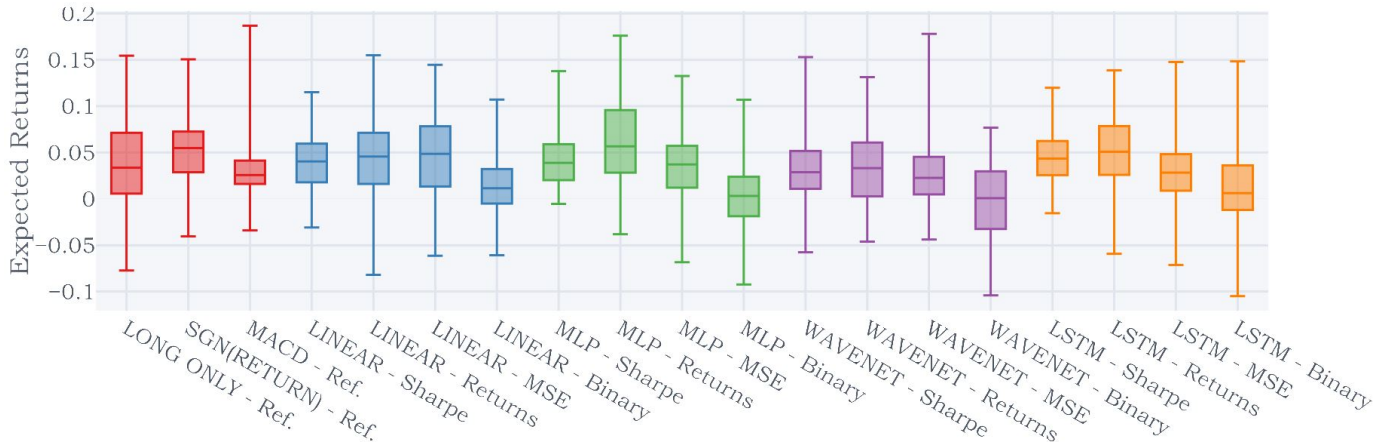


(b) Sortino Ratio

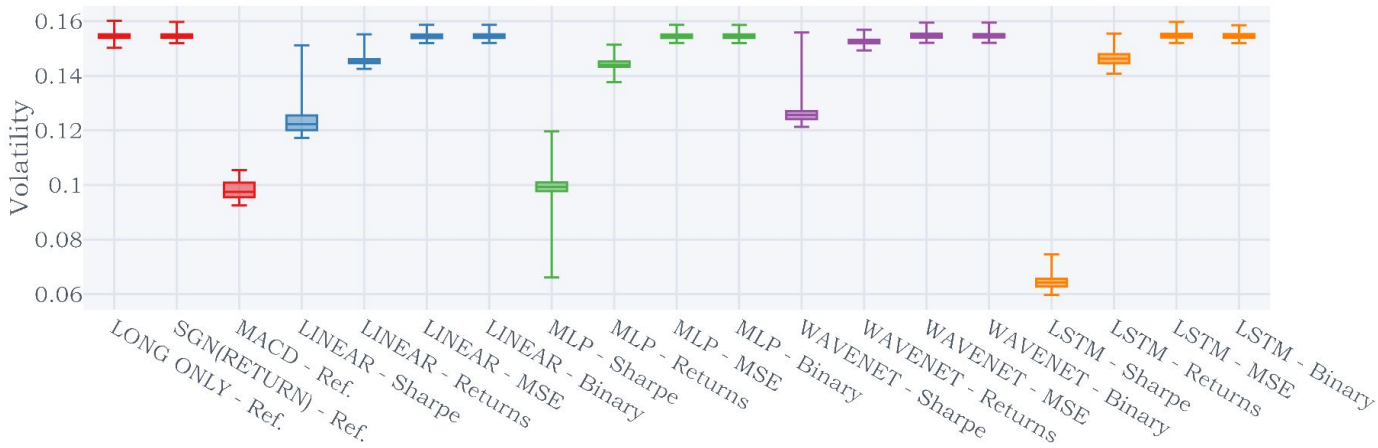


(c) Calmar Ratio

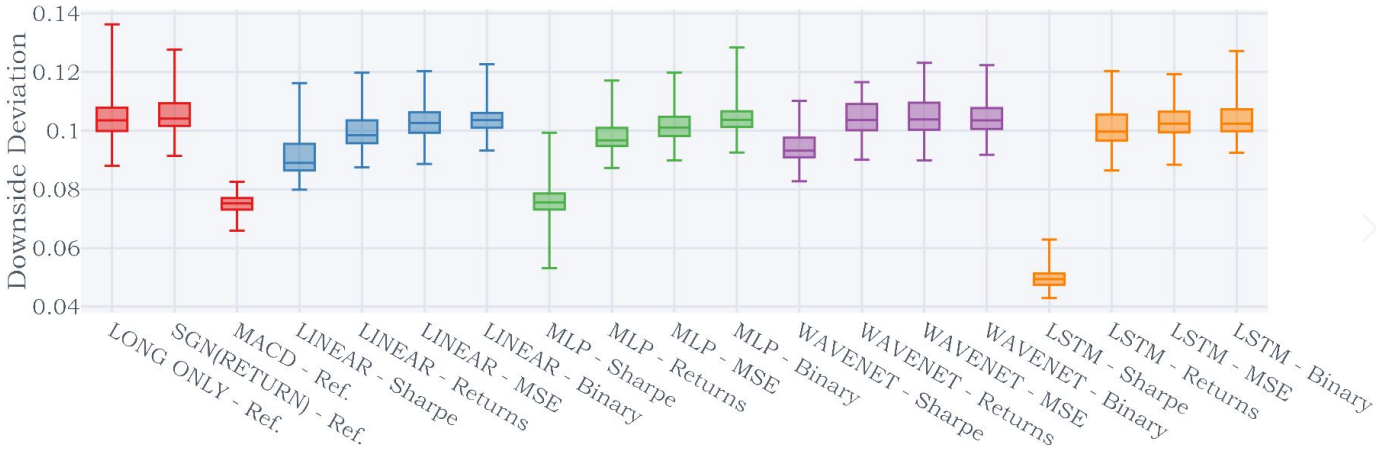
Exhibit 13: Reward vs Risk Across Individual Assets



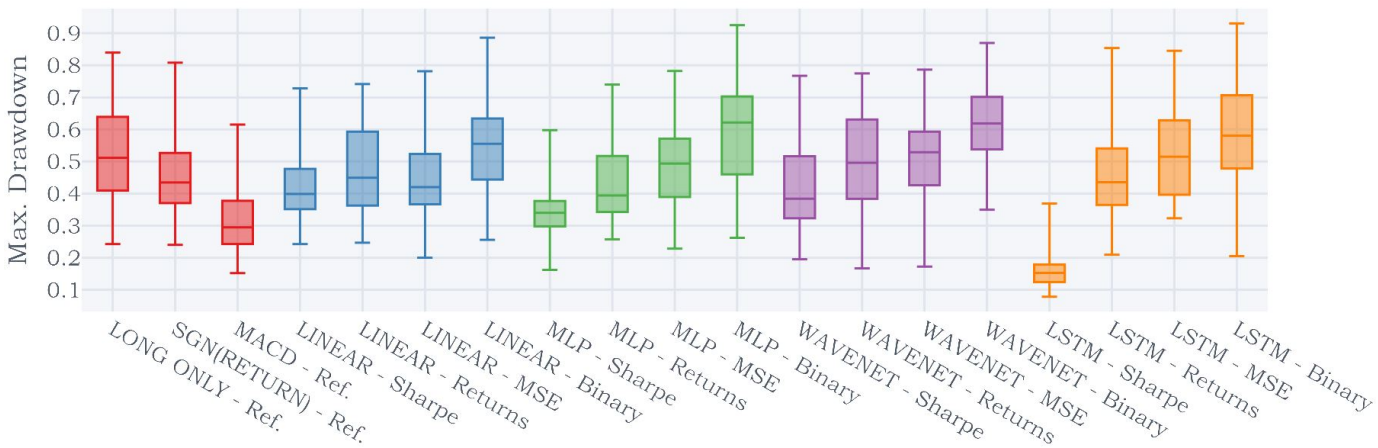
(a) Expected Returns



(b) Volatility



(c) Downside Deviation



(d) Max. Drawdown

Proceedings of Machine Learning Research 85:1–23, 2018 Machine Learning for Healthcare

Disease-Atlas: Navigating Disease Trajectories using Deep Learning

Bryan Lim BRYAN.LIM@ENG.OX.AC.UK
University of Oxford, Oxford, UK

Mihaela van der Schaar MIHAELA.VANDERSCHAAR@ENG.OX.AC.UK
University of Oxford, Oxford, UK
Alan Turing Institute, London, UK

Abstract

Joint models for longitudinal and time-to-event data are commonly used in longitudinal studies to forecast disease trajectories over time. While there are many advantages to joint modeling, the standard forms suffer from limitations that arise from a fixed model specification and computational difficulties when applied to high-dimensional datasets. In this paper, we propose a deep learning approach to address these limitations, enhancing existing methods with the inherent flexibility and scalability of deep neural networks while retaining the benefits of joint modeling. Using longitudinal data from the UK Cystic Fibrosis Trust, we demonstrate improvements in performance and scalability, as well as robustness in the presence of irregularly sampled data.

1. Introduction

Building a Disease Atlas for clinicians involves the dynamic forecasting of medical conditions based on clinically relevant variables collected over time, and guiding them in charting a course of action. This includes the simultaneous prediction of survival probabilities, risks of developing related diseases, and relevant biomarker trajectories at different stages of disease progression. While prognosis, i.e. survival prediction, is usually the main area of focus (van Houwelingen and Putter, 2011; Rizopoulos et al., 2017), a growing area in precision medicine is the forecasting of personalized disease trajectories, using patterns in temporal correlations and associations between related diseases to predict their evolution over time. (Jensen et al., 2014; Kannan et al., 2017). Dynamic prediction methods that account for these interactions are particularly relevant in multimorbidity management, as patients with one chronic disease typically develop other long-term conditions over time (Farmer et al., 2016). With the mounting evidence on the prevalence of multimorbidity in aging populations around the world (Xu et al., 2017), the ability to jointly forecast multiple clinical variables would be beneficial in providing clinicians with a fuller picture of a patient's medical condition.

1.1. Clinical Relevance

A substantial portion of machine learning literature investigates predictions with time-series data, typically focusing on patients in the hospital. In this setting, patients are tracked for a relatively short period of time, spanning from a few days to weeks, with measurements

collected every few hours. This leads to the collection of numerous measurements, potentially with a high degree of missingness. Given the length of the monitoring period, in-hospital predictions are usually narrow in their scope, focusing on detecting the rapid onset of critical events, such as ICU admission, and not considering the prediction of comorbidities which can take years to develop.

With chronic diseases however, such as cystic fibrosis or diabetes, patients are followed up over the span of years, usually as part of regular physical examinations. This differs significantly from the in-hospital setting as measurements are collected infrequently, e.g. once every few years and possibly at irregular intervals, leading to relatively few observations per patient. The state of the patient also evolves slowly, allowing for the development of related comorbidities over time. Additional comorbidities in turn affect key biomarkers which reflect a patient's clinical state and rate of deterioration, such as lung function scores (e.g. FEV1) in cystic fibrosis or brain scan measurements in Alzheimer's disease. As such, the ability to jointly forecast comorbidity and biomarker trajectories, in addition to survival, allows for early intervention by clinicians to prevent the development of other related diseases and forestall further deterioration. This would allow for an improved quality of life for the patient even if immediate improvements to survival might be small. Hence, the development of new machine learning methods to combine longitudinal predictions with dynamic survival (time-to-event) analysis, where events-of-interest can include heart failure, respiratory failure or the onset of dementia in addition to death, would allow for a more holistic management of long-term conditions, going above and beyond the short-term survival prediction usually seen in hospital settings.

1.2. Technical Significance

Traditionally, joint models for longitudinal and time-to-event data have been commonly used in clinical studies when there is prior knowledge indicating an association between longitudinal trajectories and survival. Using individual models for each data trajectory as building blocks, such as linear mixed models for longitudinal data and the Cox proportional hazard model for survival, joint models add a common association structure on top of them, e.g. through shared-random effects or frailty models (Hickey et al., 2016). From a dynamic prediction perspective, joint models have been shown to lead to a reduced bias in estimation (Ibrahim et al., 2010) and improved predictive accuracy (Hogan and Laird, 1998). However, standard joint models face severe computational challenges when applied to large datasets, which arise when increasing the dimensionality of the random effects component (Hickey et al., 2016).

To overcome the limitations of traditional joint models, we introduce Disease-Atlas - a scalable deep learning approach to forecasting disease trajectories over time. Our main contributions are as follows:

Deep Learning for Joint Models We provide a novel conception of the joint modeling framework using deep learning, capturing the relationships between trajectories through shared representations learned directly from data, and improving scalability as a whole. The network outputs parameters of predictive distributions for longitudinal and time-to-event data that take a similar form to the sub-models used in joint modeling. To the best of our

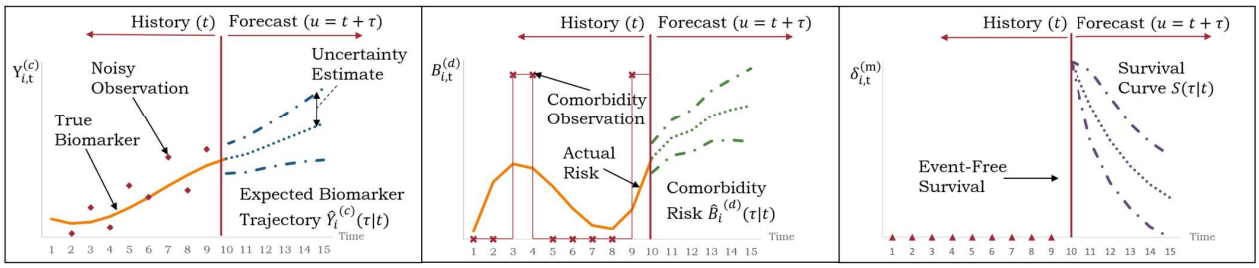


Figure 1: Illustration of Disease-Atlas Predictions over Time

knowledge, this paper is the first to investigate the use of deep learning in joint models for longitudinal and survival trajectories.

Robustness to Irregular Sampling via Multitask Learning Observations in longitudinal studies are very rarely aligned at every time step, as measurements can be collected at different sampling frequencies. Hence, training a multioutput neural network would require the imputation of the target labels as a pre-processing step, so as to artificially align the dataset prior to calibration. This could lead to poorer predictions if imputation quality is low and a high degree of missingess is present, as the network is biased to simply learn the imputation mechanism. To mitigate this issue, we formulate joint model calibration as a multitask learning problem, grouping variables - which are measured at the same time and with similar sampling frequencies - together into tasks, and training the network using only actual observations as target labels.

Incorporating Medical History into Forecasts While deep learning for medicine has gained popularity in recent times, the majority of methods, such as (Alaa and van der Schaar, 2017; Ranganath et al., 2016), only use covariates at a single time point in making predictions. However, a patient's medical history could also be informative of her future clinical outcomes, and predictions could be improved by incorporating past information. We integrate historical information into our network using a Recurrent Neural Network (RNN) in the base layer, which contains a memory state that updates over time as new observations come in.

2. Related Work

While the utility of joint models has been demonstrated by its popularity in longitudinal studies, numerous modeling choices exist, each containing its own advantages and limitations (see (Hickey et al., 2016) for a full overview). (Rizopoulos et al., 2014), for example, highlight the sensitivity of predictions to the association structures used, adopting a Bayesian model averaging approach instead to aggregate the outputs of different models over time. In this respect, the flexibility of deep learning has the potential to enhance dynamic predictions with joint models, by directly learning variable relationships from the data itself, and completely removing the need for explicit model specification. In addition, (Hickey et al., 2016; Barrett et al., 2015; Waldmann et al., 2017; Futoma et al., 2016) note performance limitations when applying standard joint models to high-dimensional datasets. These are typically estimated

using Expectation Maximization (EM) or Markov Chain Monte Carlo (MCMC) sampling methods, which rapidly grow in complexity with the number of covariates and random effects. As such, most studies and software packages often focus on modeling a single or a small number of longitudinal measurements, along with a time-to-event of interest. However, the increase in data availability through electronic health records opens up the possibility of using information from multiple trajectories to improve predictions. Recent works have attempted to address this limitation by exploiting special properties of the longitudinal sub-models, such as the multivariate skew-normal structure in (Barrett and Su, 2017), and the combination of variational approximation and dynamic EM-style updates over time in (Futoma et al., 2016). In light of this, the use of deep learning holds much promise in enhancing the performance of joint models, given its inherent ability to scale with large datasets without the need for specific modeling assumptions.

Deep learning has seen increasing use in medical applications, with successes in traditional survival analysis (Ranganath et al., 2016; Luck et al., 2017) survival analysis with competing risks (Lee et al., 2018; Alaa and van der Schaar, 2017) and treatment recommendations (Katzman et al., 2016). In general, these methods focus purely on forecasting survival, do not consider dynamic prediction over time and only use covariates at a single time point in making predictions. Deep Kalman Filters (Krishnan et al., 2017) use a network which does dynamically update its latent states over time but assumes that all outputs follow the same distribution. This prevents it from being applied to heterogeneous datasets, which limits its usage for joint modeling.

RNNs have also been used extensively in making predictions inside the hospital, using frequently sampled measurements as inputs for event detection or automated diagnosis. This includes joint architectures for forecasting follow-up times and clinical diagnoses (Choi et al., 2016; Du et al., 2016) and multitask learning (Harutyunyan et al., 2017; Razavian et al., 2016; Lipton et al., 2016). While these architectures bear resemblance to our network, some fundamental differences exist. Firstly, they focus on dynamic multi-label classification tasks which produce a single label at each point, i.e. the most likely event or diagnosis at the next follow. In contrast, the Disease-Atlas allows for the simultaneous prediction of multiple variables of interest at each time step, which can be either discrete (classification) or continuous (regression). Secondly, the RNNs in existing works typically produce single point forecasts, which do not account for the uncertainty of the model. We address this limitation using the Monte-Carlo Dropout procedure of (Gal and Ghahramani, 2016) (see Section 4.3), producing predictions with uncertainty estimates at each time step. In addition, the event times predicted by (Choi et al., 2016; Du et al., 2016) correspond to the timing of the next follow up, essentially providing additional information on when a diagnosis is expected to be observed. In contrast, the Disease-Atlas allows for the joint forecasting of multiple outcomes of interest, allowing the co-occurrence of diseases and modeling the expected survival time directly. Lastly, the multitask learning framework of the Disease-Atlas allows it benefit from the improved representation learning seen in previous methods, but also extends previous work with modifications to handle irregular sampling in the output.

In addition, Gaussian process (GP) based models have also been studied in the context of joint modeling (Schulam and Saria, 2017; Soleimani et al., 2018). While these also forecast multiple outcomes of different types, inference at each time step can be expensive for long trajectories, with sparse GP approximations having at least $O(M^2T)$ complexity, where T

is the length of the trajectory and M the number of inducing points. In contrast, RNNs update their memory state once per time step, without having to iterate over the entire observation history. In addition, static patient covariates, such as genetic or demographic information which can have significant impact on a patient's clinical trajectory, are either omitted from the model (Schulam and Saria, 2017), or incorporated as a linear additive component to the time-to-event submodel (Soleimani et al., 2018). This is easily added as an additional input to the deep neural network, which can also learn the complex interactions between static and longitudinal variables directly from data.

3. Problem Definition

For a given longitudinal study, let there be N patients with observations made at time t, for $0 \leq t \leq T_{cens}$ where T_{cens} denotes an administrative censoring time [1]. For the i^{th} patient at time t, observations are made for a K-dimensional vector of longitudinal variables $\mathbf{V_{i,t}} = [Y_{i,t}^{(1)}, \ldots, Y_{i,t}^{(C)}, B_{i,t}^{(1)}, \ldots, B_{i,t}^{(D)}]$, where $Y_{i,t}^{(c)}$ and $B_{i,t}^{(d)}$ are continuous and discrete longitudinal measurements respectively, a L-dimensional vector of external covariates $\mathbf{X_{i,t}} = [X_{i,t}^{(1)}, \ldots X_{i,t}^{(L)}]$, and a M-dimensional vector of event occurrences $\delta_{\mathbf{i,t}} = [\delta_{i,t}^{(1)}, \ldots, \delta_{i,t}^{(M)}]$, where $\delta_{i,t}^{(m)} \in \{0, 1\}$ is an indicator variable denoting the presence or absence of the m^{th} event. $T_{i,t}^{(m)}$ is defined to be the first time the event is observed after t, which allows us to model both repeated events and events that lead to censoring (e.g. death). The final observation for patient i occurs at $T_{i,\max} = \min(T_{cens}, T_{i,0}^{(a_1)}, \ldots, T_{i,0}^{(a_{\max})})$, where $\{a_i, \ldots, a_{\max}\}$ is the set of indices for events that censor observations. Furthermore, we introduce a filtration $\mathcal{F}_{i,t}$ to capture the full history of longitudinal variables, external covariates and event occurrences of patient i until time t.

3.1. Joint Modeling

From (Hickey et al., 2016), numerous sub-models for longitudinal measurements exist, each with their own pros and cons. General forms for continuous and binary longitudinal measurements are typically expressed as:

$$Y_{i,u}^{(c)} | \mathcal{F}_{i,t} \sim \mathrm{N}\left(m^{(c)}\left(u, \mathcal{F}_{i,t}; \mathbf{b}_i, \tilde{\mathbf{W}}\right), \sigma_u^{(c)\ 2}\right) \tag{1}$$

$$B_{i,u}^{(d)} | \mathcal{F}_{i,t} \sim \text{Bernoulli}\left(\Phi^{(d)}\left(u, \mathcal{F}_{i,t}; \mathbf{b}_i, \tilde{\mathbf{W}}\right)\right) \tag{2}$$

Where $m^{(c)}(.)$ is a function for the predictive mean of the c-th longitudinal variable, and $\sigma_t^{(c)\ 2}$ its variance. $\Phi^{(d)}(.)$ is a function for the probability of the binary observation, such as the commonly used logit or probit functions, and $\tilde{\mathbf{W}}$ is the vector of static coefficients used by the sub-models.

In both models, $\mathbf{b}_i$ is a vector of association parameters used across trajectories, and define the association structure of the joint model. While the majority of models use subject-specific random effects, this can also refer to time-dependent latent variables as seen in (Ibrahim et al., 2004) or shared spline coefficients in (Barrett and Su, 2017).

1. Administrative censoring refers to the right-censoring that occurs when a study observation period ends.

Event times can be expressed using the general form below:

$$T_{i,t}^{(m)}|\mathcal{F}_{i,t} \sim \mathcal{S}\left(\Lambda^{(m)}\left(t, \mathcal{F}_{i,t}; \mathbf{b}_i, \tilde{\mathbf{W}}\right)\right) \tag{3}$$

Where $\mathcal{S}$ is an appropriate survival distribution (e.g. Exponential, Weibull, etc), and $\Lambda^{(m)}(.)$ is a generic cumulative hazard function. In most joint model applications, this typically takes the form of the Cox proportional hazards model.

The standard linear mixed effects models can be expressed as:

$$\begin{aligned} Y_{i,t}^{(c)} &= \mathbf{X}_{i,t}^{\top}\, \theta_{fix}^{(c)} + \mathbf{R}_{i,t}^{\top}\, \theta_{rand}^{(c)} + \epsilon_{i,t}^{(c)} \\ &= m^{(c)}(t) + \epsilon_t^{(c)} \end{aligned} \tag{4}$$

$$h_{i,t}^{(m)} = h_0(t) \exp\left(\mathbf{B}_i^{\top}\gamma^{(m)} + \beta^{(m)} m^{(c)}(t)\right) \tag{5}$$

Where $\mathbf{X}_{i,t}$ and $\mathbf{R}_{i,t}$ are time-dependent design vectors for fixed effects $\theta_{fix}^{(c)}$ and random effects $\theta_{rand}^{(c)}$, $\epsilon_{i,t}^{(c)} \sim N\left(0, \sigma_t^{(c)\ 2}\right)$ is a random noise term, and $h_{i,t}^{(m)}$ is the hazard rate of the survival process, with patient fixed covariates $\mathbf{B}_i$, and static coefficients $\gamma^{(m)}$ and $\beta^{(m)}$.

In this model, we define the association parameters of the joint model to be those common to both longitudinal and survival processes, i.e $\mathbf{b}_i = [\theta_{fix}^{(c)}, \theta_{rand}^{(c)}]$, and static coefficients which are unique to the separate processes, i.e. $\tilde{\mathbf{W}} = [\gamma^{(m)}, \beta^{(m)}]$.

3.2. Dynamic Prediction

Dynamic prediction in joint models can be defined as the estimation of both the expected values of longitudinal variables and survival probabilities over a specific time window τ in the future:

$$\hat{V}_i^{(k)}(\tau|t) = \mathbb{E}\left[V_{i,t+\tau}^{(k)}|\mathcal{F}_{i,t}; \mathbf{b}_i, \tilde{\mathbf{W}}\right] \tag{6}$$

$$S_i^{(m)}(\tau|t) = P\left(T_{i,0}^{(m)} \geq t + \tau | T_{i,0}^{(m)} \geq t, \mathcal{F}_{i,t}; \mathbf{b}_i, \tilde{\mathbf{W}}\right) \tag{7}$$

Where $V_{i,t}^{(k)}$ is the k^{th} longitudinal variable at time t that can be either continuous or binary. A conceptual illustration of dynamic prediction can be found in Figure 1. For continuous longitudinal variables, such as biomarker predictions, the goal is to forecast its expected value given all the information until the current time step (i.e. $\mathcal{F}_{i,t}$). In the case of binary observations, such as the presence of a comorbidity, the expectation in Equation 6 is the probability (or risk) of developing a comorbidity at time $t + \tau$. The survival curves shown give us the probability of not experiencing an event over various horizons τ given $\mathcal{F}_{i,t}$. While the uncertainty estimates are model dependent, these are usually expressed via confidence intervals in frequentist methods, or using the posterior distributions for $\mathbf{b}_{i,t}$ and $\tilde{\mathbf{W}}$ in Bayesian models.

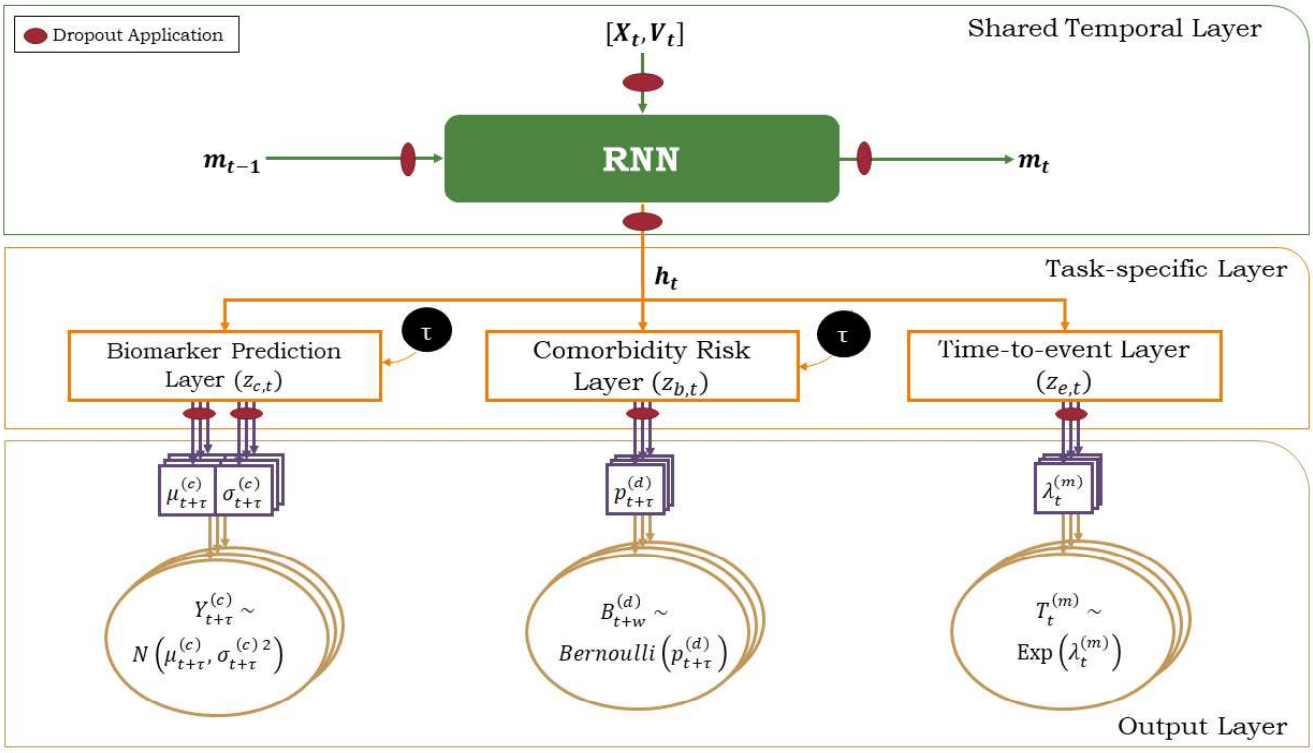


Figure 2: Disease-Atlas Network Architecture

4. Network Design

4.1. Architecture

Disease-Atlas captures the associations within the joint modeling framework, by learning *shared representations* between trajectories at different stages of the network, while retaining the same sub-model distributions captured by joint models. The network, as shown in Figure 2, is conceptually divided into 3 sections: 1) A shared temporal layer to learn the temporal and cross correlations between variables, 2) task-specific layers to learn shared representations between related trajectories, and 3) an output layer which computes parameters for predictive sub-model distributions for use in likelihood loss computations during training and generating predictive distributions at run-time.

The equations for each layer are listed in detail below. For notational convenience, we drop the subscript i for variables in this section, noting that the network is only applied to trajectories from one patient at time.

Shared Temporal Layer We start with an RNN at the base of the network, which incorporates historical information (i.e. $\mathcal{F}_t$) into forecasts by updating its memory state over time. For the tests in Section 5, the usage of both the Simple Recurrent Network (SRN)

and LSTM in this layer was compared.

$$[\mathbf{h_t}, \mathbf{m_t}] = \text{RNN}([\mathbf{X_t}, \mathbf{V_t}], \mathbf{m_{t-1}}) \tag{8}$$

Where $\mathbf{h_t}$ is the output of the RNN and $\mathbf{m_t}$ its memory state. To generate uncertainty estimates for forecasts and retain consistency with joint models, we adopt the MC dropout approach described in (Gal and Ghahramani, 2016). Dropout masks are applied to the inputs, memory states and outputs of the RNN, and are also fixed across time steps. For memory updates, the RNN uses the Exponential Linear Unit (ELU) activation function.

Task-specific Layers For the task-specific layers, variables are grouped according to the sub-model types in Section 3.1, with layer $\mathbf{z_{c,t}}$ for continuous-valued longitudinal variables, $\mathbf{z_{b,t}}$ for binary longitudinal variables and $\mathbf{z_{e,t}}$ for events. Dropout masks are also applied to the outputs of each layer here. At the inputs to the continuous and binary task layers, a prediction horizon τ is also concatenated with the outputs from the RNN. This allows the parameters of the predictive distributions at $t + \tau$ to be computed in the final layer, i.e. $\tilde{\mathbf{h}}_\mathbf{t} = [\mathbf{h}_t, \tau]$.

$$\mathbf{z_{c,t}} = \text{ELU}(\mathbf{W_c}\tilde{\mathbf{h}}_\mathbf{t} + \mathbf{a_c}) \tag{9a}$$
$$\mathbf{z_{b,t}} = \text{ELU}(\mathbf{W_b}\tilde{\mathbf{h}}_\mathbf{t} + \mathbf{a_b}) \tag{9b}$$
$$\mathbf{z_{e,t}} = \text{ELU}(\mathbf{W_e}\mathbf{h_t} + \mathbf{a_e}) \tag{9c}$$

Output Layer The final layer computes the parameter vectors of the predictive distribution, which are used to compute log likelihoods during training and predictions at run-time.

$$\mu_{\mathbf{t}+\tau} = \mathbf{W}_\mu \mathbf{z_{c,t}} + \mathbf{a}_\mu \tag{10a}$$
$$\sigma_{\mathbf{t}+\tau} = \text{Softplus}(\mathbf{W}_\sigma \mathbf{z_{c,t}} + \mathbf{a}_\sigma)) \tag{10b}$$
$$\mathbf{p_{t+\tau}} = \text{Sigmoid}(\mathbf{W_p z_{b,t}} + \mathbf{a_p})) \tag{10c}$$
$$\lambda_t = \text{Softplus}(\mathbf{W}_\lambda \mathbf{z_{e,t}} + \mathbf{a}_\lambda)) \tag{10d}$$

Softplus activation functions are applied to $\sigma_{\mathbf{t}+\tau}$ and $\mathbf{p_{t+\tau}}$ to ensure that we obtain valid (i.e. ≥ 0) standard deviations and binary probabilities. For simplicity, the exponential distribution is selected to model survival times, and predictive distributions can be expressed in a similar manner to that of Section 3.1:

$$Y^{(c)}_{t+\tau} \sim N\left(\mu^{(c)}_{t+\tau}, \sigma^{(c)2}_{t+\tau}\right) \tag{11a}$$
$$B^{(d)}_{t+\tau} \sim \text{Bernoulli}\left(p^{(d)}_{t+\tau}\right) \tag{11b}$$
$$T^{(m)}_{t} \sim \text{Exponential}\left(\lambda^{(m)}_{t}\right) \tag{11c}$$

4.2. Multitask Learning

From the above, the negative log-likelihood of the data given the network is:

$$\mathcal{L}(\mathbf{W}) = \sum_{i,t,w,k_c,k_b,m} -\Bigg[\log f_c\left(Y^{(c)}_{i,t+\tau}|\mu^{(c)}_{t+\tau}, \sigma^{(c)2}_{t+\tau}, \mathbf{W}\right) + \log f_b\left(B^{(d)}_{i,t+\tau}|p^{(d)}_{t+\tau}, \mathbf{W}\right) + \log f_T\left(T^{(m)}_{i,t}|\lambda^{(m)}_t, \mathbf{W}\right)\Bigg] \tag{12}$$

Where $f_c(.)$, $f_b(.)$ are likelihood functions based on Equations 11 and $\mathbf{W}$ collectively represents the weights and biases of the entire network. For survival times, $f_T(.)$ is given as:

$$f_T\left(T^{(m)}_t|\lambda^{(m)}_t, \mathbf{W}\right) = \left(\lambda^{(m)}_t\right)^{\delta_{i,T}} \exp\left(-\lambda^{(m)}_t T^{(m)}_t\right) \tag{13}$$

Which corresponds to event-free survival until time T before encountering the event (Dunteman and Ho, 2006). While the negative log-likelihood can be directly optimized across tasks, the use of multitask learning can yield the following benefits:

Better Survival Representations As shown in (Harutyunyan et al., 2017), multitask learning problems which have one main task of interest can weight the individual loss contributions of each subtask to favor representations for the main problem. For our current architecture, where we group similar tasks into task-specific layers, our loss function corresponds to:

$$L(\mathbf{W}) = -\underbrace{\alpha_c \sum^{i,t,w,c} \log f_c\left(Y^{(c)}_{t+\tau}|\mathbf{W}\right)}_{\text{Continuous Longitudinal Loss } l_c} - \underbrace{\alpha_b \sum^{i,t,w,d} \log f_b\left(B^{(d)}_{t+\tau}|\mathbf{W}\right)}_{\text{Binary Longitudinal Loss } l_b} - \underbrace{\alpha_T \sum^{i,t,m} \log f_T\left(T^{(m)}_t|\mathbf{W}\right)}_{\text{Time-to-event Loss } l_T} \tag{14}$$

Given that survival predictions are the primary focus of many longitudinal studies, we set $\alpha_c = \alpha_b = 1$ and include α_T as an additional hyperparameter to be optimized. To train the network, patient trajectories are subdivided into Q sets of $\Omega_q(i, \rho, \tau) = \{\mathbf{X_{i,0:\rho}}, \mathbf{Y_{i,\rho+\tau}}, \mathbf{T_{max,i}}, \delta_\mathbf{i}\}$, where ρ is the length of the covariate history to use in training trajectories up to a maximum of $\rho_{\max}$. Our procedure follows that of (Collobert and Weston, 2008), as detailed in Algorithm 1.

Handling Irregularly Sampled Data We address issues with irregular sampling by grouping variables that are measured together into the same task, and training the network with multitask learning. For instance, height, weight and BMI measurements are usually taken at the same time during follow-up, and can be grouped together. Given the completeness of the datasets we consider, we assume that task groupings match those defined by the task-specific layer of the network, and multitask learning is performed using Equation 14 and Algorithm 1.

We note, however, that in the extreme case where none of the trajectories are aligned, we can define each variable as a separate task with its own loss function l_*. Algorithm 1

Algorithm 1 Training Disease-Atlas

Input: Data $\Omega = \{\Omega_1, \ldots, \Omega_Q\}$, max iterations $\mathcal{J}$
Output: Calibrated network weights $\mathbf{W}$
for `count`$= 1$ **to** $\mathcal{J}$ **do**
  Get minibatch $\mathcal{M} \sim \gamma$ random samples from Ω
  Sample task loss function $l \sim \{l_c, l_b, l_T\}$
  Update $\mathbf{W} \leftarrow$ `Adam`$(l, \mathcal{M})$, using feed-forward passes with dropout applied
end for

then samples loss functions for one variable at a time, and the network is trained using only actual observations as target labels. This could reduce errors in cases where multiple sample rates exist and simple imputation is used, which might result in the multioutput networks replicating the imputation process instead of making true predictions.

4.3. Forecasting Disease Trajectories

Dynamic prediction involves 2 key elements - 1) calculating the expected longitudinal values and survival curves as described in Section 3.2, and 2) computing uncertainty estimates. To obtain these measures, we apply the Monte-Carlo dropout approach of (Gal and Ghahramani, 2016) by approximating the posterior over network weights as:

$$p(V_{t+\tau}^{(k)}|\mathcal{F}_t) \approx \frac{1}{J}\sum_{j=1}^{J} p(V_{t+\tau}^{(k)}|\mathcal{F}_t, \hat{\mathbf{W}}_j) \tag{15}$$

Where we draw J samples $\hat{\mathbf{W}}_j$ using feed-forward passes through the network with the same dropout mask applied across time-steps. The samples obtained can then be used to compute expectations and uncertainty intervals for forecasts.

5. Tests on Medical Data

5.1. Overview of Datasets

The UK Cystic Fibrosis (CF) registry contains data obtained for a cohort of 10980 CF patients during annual follow ups between 2008-2015, with a total of 87 variables associated with each patient across all years. In our investigations below, we consider a joint model for 2 continuous lung function scores (FEV1 and Predicted FEV1), 20 comorbidity and infection risks (treated as binary longitudinal observations) as well as death as the event of interest, simultaneously forecasting them all at each time step. We refer the reader to Appendix A for a full breakdown of the dataset.

5.2. Benchmarks and Training Procedure

We compared the Disease-Atlas (DA) against simpler neural networks i.e. LSTM and Multilayer Perceptrons (MLP), and traditional dynamic prediction methods, i.e. landmarking (L) (van Houwelingen and Putter, 2011) and joint models (JM). The data was partitioned into 3 sets: a training set with 60% of the patients, a validation set with 20% and a testing

set with the final 20%. Hyper-parameter optimization was performed using 20 iterations of random search on the validation data, and the test set was reserved for out-of-sample testing. Full details on the training procedure can be found in Appendix A. Models were compared using several metrics: Mean-Squared Error (MSE) for predictions of continuous longitudinal variables, and the area under the Receiver Operating Characteristic (AUROC) and Precision-Recall Curve (AUPRC) for binary variables and the event of interest, i.e. death. Predictions were made via the MC dropout procedure described in Section 4.3, with expectations computed using 300 samples per time step.

Disease-Atlas Through subsequent tests, we demonstrate the performance contributions of the different innovations of Disease-Atlas, namely the usage of multitask learning in the presence of irregular sampling (Section 5.3), and the inclusion of the RNN in the base layer for temporal information (Section 5.4). For the temporal layer, we evaluate the use of both a LSTM (DA-LSTM) or a single ELU layer (DA-NN).

Standard Neural Networks Both the LSTM and MLP were taken to be benchmarks, structured to produce the same output distribution parameters as the Disease-Atlas, and optimized according to the multioutput loss function in Equation 12. This makes the benchmarks equivalent to the DA-LSTM and DA-NN structures without task-specific layers and input τ, and restricts them to making one-step-ahead longitudinal predictions only.

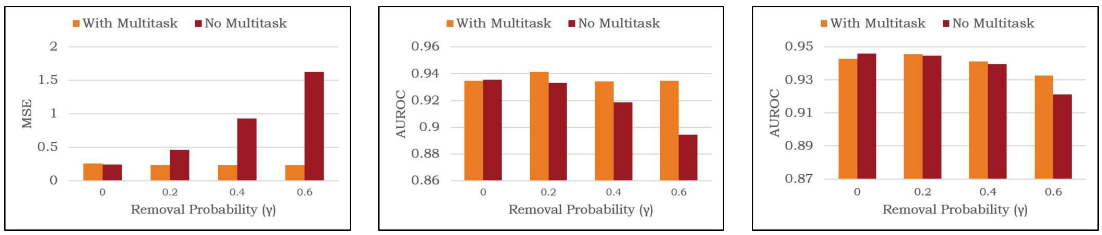
Landmarking For consistency with other studies of Cystic Fibrosis (MacKenzie et al., 2014), we use age as the time variable, and fit separate Cox regression models for patients in different age groups (< 25, $25 - 50$, $50 - 75$ and > 75 years old). As data is left-truncated with respect to age, we use the entry-exit implementation of the Cox proportional hazards model implemented in (Therneau, 2015). To avoid issues with collinearity, we start with a preliminary feature selection step first - performing multi-step Cox regression on the validation dataset, and only retaining features with coefficient p-values < 0.1.

Joint Models With the computational limitations of standard joint models, direct application to our dataset - containing over 6,500 patients in the training set with 87 annual covariate measurements - proved to be infeasible. As such, we used the two-step estimation procedure of (Wu, 2009), fitting the individual linear mixed effects (LME) longitudinal sub-models first, and using the mean estimates in the Cox regression model. For continuous variables, the linear mixed effects models used random intercepts and slopes, this is in line with the FEV1 models in (van Horck et al., 2017), (Van Diemen et al., 2011) and (Stern et al., 2007). For binary variables, we use the logit regression version of the LME model, as implemented in the `lme4` R package (Bates et al., 2015).

5.3. Evaluating Multitask Learning with Irregularly Sampled Data

To evaluate the effectiveness of multitask learning, we simulate irregular sampling by randomly removing all data points across each task (defined in Section 4.2) with a probability γ at every time step. For multivariate prediction, continuous-valued inputs were imputed using the mean value of the training set, while binary variables and indicators of death were set to 0. The networks were trained according to Section 4.2, and then evaluated on the complete test set based on 1) MSE for continuous variables, and 2) AUROC for binary/event predictions.

Figure 3 shows the outperformance of multitask learning, which has a lower MSE for FEV1 and higher AUROCs for comorbidity and mortality predictions as γ increases. The improvements are most pronounced for MSE, as FEV1 has values at every time step, as opposed to binary observations and occurrences of death which are relatively more infrequent in the dataset. This demonstrates the robustness of the model to irregular sampling, and provides a way for joint models to be used on datasets even under such conditions.



(a) FEV1 MSE (b) Comorbidity AUROC (c) Mortality AUROC

Figure 3: Performance Comparison Between Multitask and Multioutput Networks

5.4. Performance vs Benchmarks

As survival analysis is the usually main task of interest, we perform a comprehensive evaluation across all benchmarks, computing a probability of the event of interest, i.e. death, at a given time step using $1 - S_i^{(m)}(\tau|t)$ from Equation 7. Performance was compared on the basis of AUROC and AUPRC of mortality predictions at various horizons τ. To account for the stochastic nature of the MC dropout sampling procedure, performance evaluation was repeated 3 times for each neural network, with the averages and standard deviations of the metrics in reported in Table 1.

Table 1: Results of Mortality Predictions for Cystic Fibrosis (Mean ± S.D. Across 3 Runs)

	τ	DA-LSTM	DA-NN	LSTM	MLP	L	JM
AUROC	1	**0.944(± 0.0004)**	0.943(± 0.0003)	0.943(± 0.0007)	0.941(± 0.0003)	0.824	0.870
	2	**0.924(± 0.0008)**	0.923(± 0.0005)	0.923(± 0.0005)	0.919(± 0.0003)	0.812	0.870
	3	**0.910(± 0.0003)**	0.905(± 0.0002)	0.908(± 0.0002)	0.907(± 0.0002)	0.825	0.851
	4	**0.905(± 0.0003)**	0.902(± 0.0008)	0.904(± 0.0003)	0.904(± 0.0006)	0.776	0.828
	5	**0.895(± 0.0003)**	0.892(± 0.0005)	0.894(± 0.0005)	0.888(± 0.0007)	0.765	0.806
AUPRC	1	**0.278 (± 0.0037)**	0.238 (± 0.0040)	0.230 (± 0.0020)	0.219 (± 0.0036)	0.161	0.119
	2	**0.193 (± 0.0014)**	0.169 (± 0.0033)	0.165 (± 0.0017)	0.186 (± 0.0036)	0.082	0.092
	3	0.103 (± 0.0005)	0.092 (± 0.0007)	0.099 (± 0.0028)	**0.105 (± 0.0001)**	0.085	0.089
	4	**0.109 (± 0.0007)**	0.101 (± 0.0014)	0.095 (± 0.0010)	0.102 (± 0.0006)	0.062	0.068
	5	**0.101 (± 0.0007)**	0.091 (± 0.0008)	0.093 (± 0.0017)	0.100 (± 0.0017)	0.058	0.059

The first thing to note is the vast improvements of neural networks over traditional dynamic prediction models, with the DA-LSTM showing average AUPRC improvements of 75% and 78% across all time steps when compared to landmarking and standard joint models. Among the neural network benchmarks, the strongest gains for the DA-LSTM can be seen over shorter prediction horizons (1-2 years), improving the DA-NN by 17% and the LSTM

by 21% in one-step-ahead AUPRC. This highlights the benefits of both the use of temporal information and the addition of task-specific layers for short-term survival predictions. While the gains in AUROC are indeed smaller, this can be attributed to the large class imbalance present within the dataset, due to the rare occurrence of death in the dataset (seen in 451 of 10275 patients) and the censoring effect it has on a patients trajectory. As shown in (Saito and Rehmsmeier, 2015), ROC metrics can lead to deceptive good performance, as the definition of the false positive rate (false positive / total number of negatives) permits the occurrence of a large number of false positives in imbalanced datasets, and recommend the use of PRC metrics. In addition, we note that the Disease-Atlas architectures also allow for the forecasting of longitudinal variables over arbitrary horizons, which standard neural network architectures are unable to accommodate by default.

Table 2: Results of Longitudinal Predictions for Cystic Fibrosis (Single Run)

		MSE		**AUROC [Mean ± SD]**		**AUPRC [Mean ±SD]**	
	τ	FEV1	Pred. FEV1	Comorbidities	Infections	Comorbidities	Infections
DA-LSTM	1	**0.182**	**121.3**	**0.957 (± 0.025)**	**0.888 (± 0.056)**	**0.680 (±0.261)**	**0.416 (±0.247)**
	2	**0.191**	**139.4**	**0.926 (±0.047)**	**0.850 (± 0.044)**	**0.648 (± 0.244)**	**0.337 (± 0.261)**
	3	**0.275**	**191.3**	**0.882 (±0.048)**	**0.798 (± 0.057)**	**0.555 (± 0.213)**	**0.337 (± 0.261)**
	4	**0.374**	**254.4**	**0.817 (±0.085)**	**0.723 (± 0.068)**	**0.459 (± 0.184)**	**0.309 (± 0.252)**
	5	**0.461**	**308.1**	**0.790 (±0.067)**	**0.669 (± 0.126)**	**0.388 (± 0.169)**	**0.269 (± 0.247)**
JM	1	0.553	368.6	0.699 (± 0.148)	0.673 (± 0.069)	0.176 (± 0.088)	0.161 (± 0.176)
	2	0.593	411.1	0.694 (± 0.139)	0.651 (± 0.060)	0.180 (± 0.089)	0.157 (± 0.181)
	3	0.641	451.8	0.685 (± 0.140)	0.631 (± 0.072)	0.185 (± 0.090)	0.160 (± 0.186)
	4	0.695	490.1	0.681 (± 0.132)	0.607 (± 0.077)	0.187 (± 0.091)	0.159 (± 0.188)
	5	0.750	519.7	0.673 (± 0.130)	0.580 (± 0.082)	0.188 (± 0.093)	0.155 (± 0.186)

We proceed to evaluate the longitudinal forecasting ability of the Disease-Atlas in Table 2, focusing on comparisons between the DA-LSTM and standard joint models. To provide a concise summary of longitudinal prediction results, we use a single set of 300 MC dropout samples to compute performance metrics. The average AUROC/AUPRC for comorbidities and infections are reported separately, along with standard deviations across each group. Similarly, results for FEV1 and Predicted FEV1 are reported for a single evaluation, and a full breakdown of results for each individual binary longitudinal variable is detailed in Appendix A. We can see that the DA-LSTM improves performance across all longitudinal prediction categories, reducing MSEs by 56% on average across all continuous predictions, and improving AUPRCs in comorbidities and infections on average by 199% and 112% respectively. This vast improvement underscores the ability of deep neural networks to learn complex interactions directly from the data, and demonstrates the benefits of using a deep learning approach to joint modeling.

6. Illustrative Use Case for Disease-Atlas: Personalized Screening

While numerous use cases for the Disease-Atlas exist, one possible example is its use in personalized screening, i.e. prescribing testing regimes and follow-up schedules that are tailored to the unique characteristics of a patient. (Ahuja et al., 2017) derive an optimal policy that balances the costs of screening, along with the risks of delaying the screening process. Their paper, however, has two important limitations: 1) it requires the evolution of

the disease to be known, and 2) it requires an analytical expression for the cost of delay, which is often difficult to determine in practice. Disease-Atlas does not suffer from these limitations.

We illustrate how the Disease Atlas can be used for identifying screening profiles for Cystic Fibrosis patients. We use as an exemplar an actual patient from our test set who started to be seen in 2008 and is screened for Diabetes, a very important comorbidity affecting the treatment of patients.

Figure 4 shows how the Disease Atlas can be used to design personalized screening policies. Using the Disease Atlas as applied from 2008-2010, we can record the "smoothed" estimate at each time step, i.e. $\hat{B}_i^{(d)}(0|t)$ as per Equation 6 and plotted in orange. To determine a patient's future risk, we extrapolate in the usual way over various horizons τ, providing both the expected value and an uncertainty interval using the 5th and 95th percentiles of the MC dropout samples. From Figure 6, we see that the patient had a steady 18% increase in risk from 2008 to 2010, and will expect a further increase of 13% over the next 5 years. Informed by the Disease Atlas, the clinician may decide to prescribe additional tests for Diabetes, or increase the frequency of follow-up in the short-term to better monitor risks.

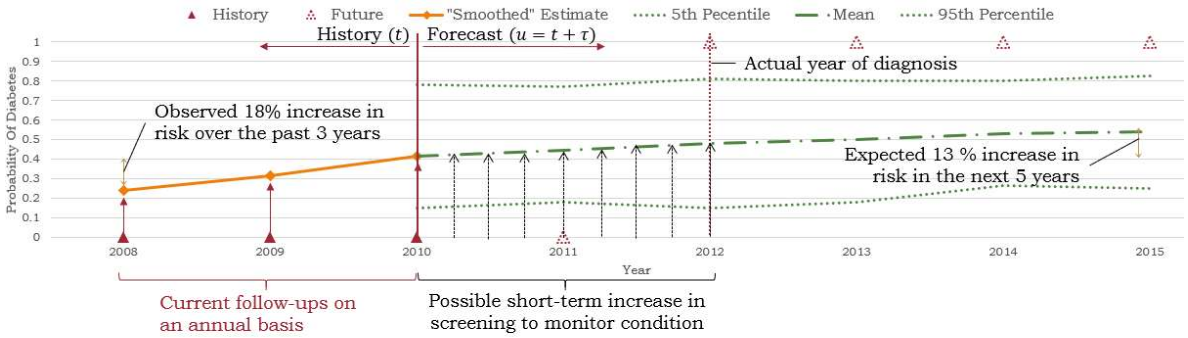


Figure 4: Using Disease-Atlas for Personalized Screening

References

Kartik Ahuja, William Zame, and Mihaela van der Schaar. Dpscreen: Dynamic personalized screening. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, NIPS 2017, pages 1321–1332. 2017.

A. M. Alaa and M. van der Schaar. Deep multi-task gaussian processes for survival analysis with competing risks. In *Advances in Neural Information Processing Systems*, NIPS 2017. 2017.

J Barrett and L. Su. Dynamic predictions using flexible joint models of longitudinal and timetoevent data. *Statistics in Medicine.*, 36(9):1447–1460, April 2017.

Jessica Barrett, Peter Diggle, Robin Henderson, and David Taylor-Robinson. Joint modelling of repeated measurements and time-to-event outcomes: flexible model specification and exact likelihood inference. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(1):131–148, 2015.

Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. doi: 10.18637/jss.v067.i01.

Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F. Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, pages 301–318, 2016.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML 2008, pages 160–167, 2008.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1555–1564, 2016.

George H. Dunteman and Moon-Ho R. Ho. *An Introduction to Generalized Linear Models.* SAGE Publications, Inc., Thousand Oaks, CA, USA, 2006.

Caroline Farmer, Elisabetta Fenu, Norma O'Flynn, and Bruce Guthrie. Clinical assessment and management of multimorbidity: summary of nice guidance. *BMJ*, 354, 2016.

Joseph Futoma, Mark Sendak, C. Blake Cameron, and Katherine Heller. Scalable joint modeling of longitudinal and point process data for disease trajectory prediction and improving management of chronic kidney disease. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'16, pages 222–231, 2016.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, NIPS 2016, 2016.

Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *CoRR*, abs/1703.07771, 2017. URL https://arxiv.org/abs/1703.07771.

Graeme L. Hickey, Pete Philipson, Andrea Jorgensen, and Ruwanthi Kolamunnage-Dona. Joint modelling of time-to-event and multivariate longitudinal outcomes: recent developments and issues. *BMC Medical Research Methodology*, 16(1):117, Sep 2016.

Joseph W Hogan and Nan M Laird. Increasing efficiency from censored survival data by using random effects to model longitudinal covariates. *Statistical Methods in Medical Research*, 7(1):28–48, 1998.

Joseph G. Ibrahim, Ming Hui Chen, and Debajyoti Sinha. Bayesian methods for joint modeling of longitudinal and survival data with applications to cancer vaccine trials. *Statistica Sinica*, 14(3):863–883, 7 2004.

Joseph G. Ibrahim, Haitao Chu, and Liddy M. Chen. Basic concepts and methods for joint models of longitudinal and survival data. *Journal of Clinical Oncology*, 28(16):27962801, 2010.

Anders Boeck Jensen, Pope L. Moseley, Tudor I. Oprea1, Sabrina Gade Ellese, Robert Eriksson, Henriette Schmock, Peter Bjdstrup Jensen, Lars Juh, Jensen, and Sren Brunak. Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients. *Nature Communications.*, 5(4022), 2014.

Venkateshan Kannan, Narsis A. Kiani, Fredrik Piehl, and Jesper Tegner. A minimal unified model of disease trajectories captures hallmarks of multiple sclerosis. *Mathematical Biosciences*, 289:1 – 8, 2017.

Jared Katzman, Uri Shaham, Jonathan Bates, Alexander Cloninger, Tingting Jiang, and Yuval Kluger. Deepsurv: Personalized treatment recommender system using a cox proportional hazards deep neural network. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML 2016, 2016.

Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *CoRR*, abs/1511.05121, 2017. URL https://arxiv.org/abs/1511.05121.

C. Lee, W. R. Zame, J. Yoon, and M. van der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *AAAI*, 2018.

Zachary C. Lipton, David C. Kale, Charles Elkan, and Randall Wetzel. Learning to diagnose with lstm recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Margaux Luck, Tristan Sylvain, Héloïse Cardinal, Andrea Lodi, and Yoshua Bengio. Deep learning for patient-specific kidney graft survival analysis. *CoRR*, abs/1705.10245, 2017. URL http://arxiv.org/abs/1705.10245.

Todd MacKenzie, Alex H. Gifford, Kathryn A. Sabadosa, Hebe B. Quinton, Emily A. Knapp, Christopher H. Goss, and Bruce C. Marshall. Longevity of patients with cystic fibrosis in 2000 to 2010 and beyond: Survival analysis of the cystic fibrosis foundation patient registry. *I Annals of internal medicine*, 161(4):233–241, 2014.

Rajesh Ranganath, Adler Perotte, Nomie Elhadad, and David Blei. Deep survival analysis. In *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, pages 101–114, 18–19 Aug 2016.

Narges Razavian, Jake Marcus, and David Sontag. Multi-task prediction of disease onsets from longitudinal laboratory tests. In *Proceedings of the 1st Machine Learning for Healthcare Conference (MLHC)*, volume 56 of *Proceedings of Machine Learning Research*, pages 73–100, Children's Hospital LA, Los Angeles, CA, USA, 18–19 Aug 2016.

Dimitris Rizopoulos, Laura A. Hatfield, Bradley P. Carlin, and Johanna J. M. Takkenberg. Combining dynamic predictions from joint models for longitudinal and time-to-event data using bayesian model averaging. *Journal of the American Statistical Association*, 109 (508):1385–1397, 2014.

Dimitris Rizopoulos, Geert Molenberghs, and Emmanuel M.E.H. Lesaffre. Dynamic predictions with time-dependent covariates in survival analysis using joint modeling and landmarking. *Biometrical Journal*, 59(6):1521–4036, 2017.

Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS One*, 10(23), 2015.

Peter Schulam and Suchi Saria. Reliable decision support using counterfactual models. In *Proceedings of the thirty-first Conference on Neural Information Processing Systems, (NIPS)*, 2017.

H. Soleimani, J. Hensman, and S. Saria. Scalable joint models for reliable uncertainty-aware event prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40 (8):1948–1963, 2018.

D. A. Stern, W. J. Morgan, A. L. Wright, S. Guerra, and F. D. Martinez. Poor airway function in early infancy and lung function by age 22 years: a non-selective longitudinal cohort study. *Lancet*, 370, 2007.

Terry M Therneau. *A Package for Survival Analysis in S*, 2015. URL https://CRAN.R-project.org/package=survival. version 2.38.

C Van Diemen, D Postma, M Siedlinski, A Blokstra, H Smit, and H. Boezen. Genetic variation in timp1 but not mmps predict excess fev1 decline in two general population-based cohorts. *Respiratory Research.*, 12(1), 2011.

Marieke van Horck, Bjorn Winkens, Geertjan Wesseling, Dillys van Vliet, Kim van de Kant, Sanne Vaassen, Karin de Winter-de Groot, Ilja de Vreede, Quirijn Jbsis, and Edward Dompeling. Early detection of pulmonary exacerbations in children with cystic fibrosis by electronic home monitoring of symptoms and lung function. *Scientific Reports*, 7(1), 2017.

Hans van Houwelingen and Hein Putter. *Dynamic Prediction in Clinical Survival Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2011.

Elisabeth Waldmann, David Taylor-Robinson, Nadja Klein, Thomas Kneib, Tania Pressler, Matthias Schmid, and Andreas Mayr. Boosting joint models for longitudinal and time-to-event data. *Biometrical Journal*, 59(6):1104–112, 2017.

Lang Wu. *Mixed Effects Models for Complex Data*. Chapman & Hall/CRC, Hoboken, NJ , USA, 2009.

Xiaolin Xu, Gita D. Mishra, and Mark Jones. Evidence on multimorbidity from definition to intervention: An overview of systematic reviews. *Ageing Research Reviews*, 37:53 – 68, 2017.

Appendix for Disease-Atlas

Appendix A. Tests on Cystic Fibrosis Dataset

A.1. Details on Dataset

The UK Cystic Fibrosis (CF) registry contains data obtained for a cohort of 10980 CF patients during annual follow ups between 2008-2015, with a total of 87 variables that were associated with each patient across all years. This includes demographic information (e.g. age, height, weight, BMI), genetic information, treatments received, metrics for lung function (FEV1 and Predicted FEV1), comorbidities observed, and any bacterial infections developed. In our investigations, we consider a joint model for the 2 continuous lung function scores (FEV1 and Predicted FEV1), 20 binary longitudinal variables of comorbidity and infection, along with death as the event of interest.

A full description of the jointly-modeled longitudinal and time-to-event datasets can be found in Table 3.

A.2. Hyperparameter Optimization

Hyperparameter optimization was conducted using 20 iterations of random search, with the search space documented in Table 4. Please note that the RNN state size was defined relative to the number of input features (L). The task specific layers were also size in relation to the RNN state size, and defined to be (state size + task output size) /2. This was done to ensure that we had a principled way of sizing the task-specific layer relative to state size and outputs, without having to add on too many additional hyper-parameters (i.e. one per task-specific layer). In addition, α_T was defined in relation to the number of longitudinal variables (K). All neural networks were trained to convergence, as determined by the survival log-likelihoods evaluated on the validation data, or up to a maximum of 50 epochs.

The final parameters obtained for each network can be found in Table 5.

A.3. Additional Results

To supplement the results in the test section of the main report, a detailed breakdown of the prediction results for binary longitudinal variables can be found in Table 6 and 7 for DA-LSTM and JM respectively. For the DA-LSTM, due to the randomness present in the MC dropout procedure, the performance evaluation was repeated 3 times with the means and standard deviations reported in the tables. The results also demonstrate the outperformance of the deep neural network over standard benchmarks in both AUROC and AUPRC terms.

Appendix B. Acknowledgments

This work is supported by the UK Cystic Fibrosis Trust and the Oxford-Man Institute. We would also like to thank the UK Cystic Fibrosis Trust for providing us access to the data from the CF registry, which was used extensively in the analysis performed in this report.

Table 3: Description of Longitudinal and Time-to-event Data for CF

		Type	% Patients	Mean	S.D.	Min	Max
Event	Death	Binary (Event)	4.70%	0.008	0.087	0.000	1.000
Biomarkers	FEV1	Continuous	100.00%	2.176	0.914	0.090	6.250
	Predicted FEV1	Continuous	100.00%	72.109	22.404	8.950	197
Comorbidities	Liver Disease	Binary	20.80%	0.128	0.334	0.000	1.000
	Asthma	Binary	22.96%	0.146	0.353	0.000	1.000
	Arthropathy	Binary	9.50%	0.050	0.218	0.000	1.000
	Bone fracture	Binary	1.94%	0.007	0.081	0.000	1.000
	Raised Liver Enzymes	Binary	23.91%	0.114	0.318	0.000	1.000
	Osteopenia	Binary	20.37%	0.114	0.318	0.000	1.000
	Osteoporosis	Binary	9.58%	0.051	0.219	0.000	1.000
	Hypertension	Binary	3.30%	0.020	0.139	0.000	1.000
	Diabetes	Binary	24.56%	0.167	0.373	0.000	1.000
Bacterial Infections	Burkholderia Cepacia	Binary	5.59%	0.034	0.181	0.000	1.000
	Pseudomonas Aeruginosa	Binary	65.18%	0.407	0.491	0.000	1.000
	Haemophilus Influenza	Binary	30.55%	0.091	0.288	0.000	1.000
	Aspergillus	Binary	29.29%	0.110	0.313	0.000	1.000
	NTM	Binary	6.38%	0.019	0.136	0.000	1.000
	Ecoli	Binary	5.32%	0.012	0.111	0.000	1.000
	Klebsiella Pneumoniae	Binary	4.93%	0.010	0.101	0.000	1.000
	Gram-Negative	Binary	3.78%	0.008	0.089	0.000	1.000
	Xanthomonas	Binary	13.18%	0.043	0.202	0.000	1.000
	Staphylococcus Aureus	Binary	52.59%	0.244	0.429	0.000	1.000
	ALCA	Binary	5.06%	0.020	0.138	0.000	1.000

Table 4: Hyper-parameter Selection Range for Random Search

	Hyper-parameter Selection Range
Max Number of Epochs	50
RNN State Size	1L, 2L, 3L, 4L, 5L
α_T	K, 2K, 3K, 4K, 5K
Max Gradient Norm	0.5, 1.0, 1.5, 2.0
Learning Rate	1e-3, 5e-3, 1e-4
Minibatch Size	64, 128, 256
Dropout Rate	0.2, 0.3, 0.4, 0.5

Table 5: Hyper-parameters Selected for CF Tests

	State Size	**Minibatch Size**	**Learning Rate**	**Max Gradient Norm**	**Dropout Rate**	α_T
DA-LSTM	1L	256	1.00E-04	0.5	0.3	3K
DA-NN	5L	256	1.00E-04	2.0	0.3	4K
Standard LSTM	1L	32	1.00E-03	1.5	0.4	4K
Standard NN	1.5L	64	1.00E-03	1	0.45	1K

Table 6: AUROC for Comorbidity and Infection Predictions for CF Dataset (Mean ± S.D. Across 3 Runs)

DA-LSTM	**1**	**2**	**3**	**4**	**5**
Comorbidities					
Liver Disease	0.975 (± 0.0001)	0.948 (± 0.0001)	0.897 (± 0.0003)	0.826 (± 0.0005)	0.767 (± 0.0001)
Asthma	0.979 (± 0.0001)	0.946 (± 0.0002)	0.906 (± 0.0005)	0.843 (± 0.0004)	0.784 (± 0.0003)
Arthropathy	0.975 (± 0.0004)	0.950 (± 0.0002)	0.911 (± 0.0004)	0.888 (± 0.0007)	0.833 (± 0.0004)
Bone fracture	0.891 (± 0.0024)	0.791 (± 0.0015)	0.789 (± 0.0014)	0.610 (± 0.0029)	0.721 (± 0.0013)
Raised Liver Enzymes	0.937 (± 0.0006)	0.909 (± 0.0002)	0.798 (± 0.0004)	0.741 (± 0.0006)	0.685 (± 0.0002)
Osteopenia	0.961 (± 0.0005)	0.942 (± 0.0003)	0.897 (± 0.0006)	0.873 (± 0.0005)	0.850 (± 0.0004)
Osteoporosis	0.956 (± 0.0006)	0.943 (± 0.0008)	0.912 (± 0.0006)	0.877 (± 0.0008)	0.854 (± 0.0003)
Hypertension	0.977 (± 0.0004)	0.955 (± 0.0006)	0.936 (± 0.0009)	0.879 (± 0.0006)	0.869 (± 0.0009)
Diabetes	0.963 (± 0.0002)	0.942 (± 0.0002)	0.918 (± 0.0001)	0.873 (± 0.0003)	0.844 (± 0.0004)
Infections					
Burkholderia Cepacia	0.960 (± 0.0011)	0.929 (± 0.0009)	0.922 (± 0.0008)	0.876 (± 0.0004)	0.846 (± 0.0011)
Pseudomonas Aeruginosa	0.898 (± 0.0004)	0.878 (± 0.0002)	0.850 (± 0.0002)	0.818 (± 0.0003)	0.796 (± 0.0005)
Haemophilus Influenza	0.848 (± 0.0007)	0.835 (± 0.0002)	0.798 (± 0.0002)	0.737 (± 0.0007)	0.738 (± 0.0012)
Aspergillus	0.873 (± 0.0007)	0.798 (± 0.0004)	0.789 (± 0.0007)	0.673 (± 0.0010)	0.665 (± 0.0006)
NTM	0.897 (± 0.0008)	0.802 (± 0.0013)	0.824 (± 0.0014)	0.675 (± 0.0008)	0.633 (± 0.0002)
Ecoli	0.929 (± 0.0018)	0.894 (± 0.0013)	0.733 (± 0.0044)	0.677 (± 0.0043)	0.340 (± 0.0066)
Klebsiella Pneumoniae	0.950 (± 0.0012)	0.904 (± 0.0007)	0.815 (± 0.0044)	0.630 (± 0.0005)	0.725 (± 0.0046)
Gram-Negative	0.745 (± 0.0052)	0.793 (± 0.0045)	0.690 (± 0.0007)	0.698 (± 0.0020)	0.587 (± 0.0027)
Xanthomonas	0.894 (± 0.0009)	0.831 (± 0.0005)	0.770 (± 0.0013)	0.716 (± 0.0007)	0.654 (± 0.0006)
Staphylococcus Aureus	0.908 (± 0.0002)	0.856 (± 0.0004)	0.784 (± 0.0005)	0.699 (± 0.0003)	0.649 (± 0.0005)
ALCA	0.863 (± 0.0008)	0.831 (± 0.0010)	0.800 (± 0.0008)	0.758 (± 0.0010)	0.725 (± 0.0015)
JM	**1**	**2**	**3**	**4**	**5**
Comorbidities					
Liver Disease	0.634	0.622	0.619	0.607	0.602
Asthma	0.701	0.671	0.649	0.622	0.597
Arthropathy	0.761	0.755	0.755	0.757	0.749
Bone Fracture	0.344	0.379	0.378	0.413	0.411
Raised Liver Enzymes	0.622	0.608	0.589	0.584	0.594
Osteopenia	0.774	0.767	0.761	0.758	0.746
Osteoporosis	0.801	0.794	0.781	0.772	0.76
Hypertension	0.894	0.891	0.893	0.893	0.889
Diabetes	0.76	0.754	0.739	0.723	0.709
Infections					
Burkholderia Cepacia	0.636	0.638	0.63	0.622	0.613
Pseudomonas Aeruginosa	0.744	0.735	0.727	0.713	0.692
Haemophilus Influenza	0.698	0.712	0.722	0.715	0.685
Aspergillus	0.716	0.689	0.662	0.622	0.603
NTM	0.709	0.686	0.678	0.633	0.586
Ecoli	0.729	0.604	0.507	0.444	0.389
Klebsiella Pneumoniae	0.777	0.73	0.706	0.67	0.624
Gram-Negative	0.533	0.55	0.53	0.518	0.489
Xanthomonas	0.628	0.613	0.604	0.611	0.603
Staphylococcus Aureus	0.607	0.589	0.577	0.561	0.542
ALCA	0.624	0.613	0.598	0.572	0.552

Table 7: AUPRC for Comorbidity and Infection Predictions for CF Dataset (Mean ± S.D. Across 3 Runs)

DA-LSTM	**1**	**2**	**3**	**4**	**5**
Comorbidities					
Liver Disease	0.862 (± 0.0006)	0.825 (± 0.0007)	0.709 (± 0.0022)	0.616 (± 0.0009)	0.513 (± 0.0012)
Asthma	0.904 (± 0.0006)	0.845 (± 0.0015)	0.773 (± 0.0016)	0.642 (± 0.0024)	0.544 (± 0.0019)
Arthropathy	0.799 (± 0.0036)	0.760 (± 0.0027)	0.621 (± 0.0011)	0.500 (± 0.0038)	0.347 (± 0.0026)
Bone fracture	0.064 (± 0.0020)	0.043 (± 0.0012)	0.052 (± 0.0011)	0.032 (± 0.0011)	0.031 (± 0.0018)
Raised Liver Enzymes	0.784 (± 0.0015)	0.726 (± 0.0019)	0.536 (± 0.0016)	0.409 (± 0.0024)	0.338 (± 0.0011)
Osteopenia	0.758 (± 0.0018)	0.742 (± 0.0013)	0.648 (± 0.0024)	0.577 (± 0.0009)	0.526 (± 0.0017)
Osteoporosis	0.658 (± 0.0044)	0.644 (± 0.0028)	0.507 (± 0.0017)	0.406 (± 0.0013)	0.322 (± 0.0011)
Hypertension	0.308 (± 0.0069)	0.340 (± 0.0072)	0.309 (± 0.0074)	0.277 (± 0.0031)	0.227 (± 0.0010)
Diabetes	0.850 (± 0.0006)	0.798 (± 0.0019)	0.774 (± 0.0013)	0.663 (± 0.0020)	0.640 (± 0.0034)
Infections					
Burkholderia Cepacia	0.692 (± 0.0029)	0.672 (± 0.0026)	0.639 (± 0.0047)	0.576 (± 0.0058)	0.471 (± 0.0052)
Pseudomonas Aeruginosa	0.840 (± 0.0002)	0.828 (± 0.0010)	0.815 (± 0.0010)	0.800 (± 0.0007)	0.794 (± 0.0017)
Haemophilus Influenza	0.369 (± 0.0006)	0.332 (± 0.0005)	0.265 (± 0.0007)	0.243 (± 0.0009)	0.278 (± 0.0007)
Aspergillus	0.380 (± 0.0038)	0.315 (± 0.0008)	0.337 (± 0.0019)	0.270 (± 0.0011)	0.293 (± 0.0012)
NTM	0.237 (± 0.0008)	0.073 (± 0.0006)	0.181 (± 0.0017)	0.133 (± 0.0024)	0.138 (± 0.0008)
Ecoli	0.506 (± 0.0040)	0.242 (± 0.0030)	0.089 (± 0.0021)	0.036 (± 0.0005)	0.008 (± 0.0009)
Klebsiella Pneumoniae	0.299 (± 0.0039)	0.146 (± 0.0044)	0.060 (± 0.0041)	0.010 (± 0.0000)	0.015 (± 0.0004)
Gram-Negative	0.028 (± 0.0007)	0.038 (± 0.0013)	0.022 (± 0.0004)	0.027 (± 0.0003)	0.022 (± 0.0004)
Xanthomonas	0.298 (± 0.0068)	0.202 (± 0.0037)	0.218 (± 0.0020)	0.180 (± 0.0022)	0.128 (± 0.0019)
Staphylococcus Aureus	0.771 (± 0.0010)	0.706 (± 0.0018)	0.612 (± 0.0014)	0.537 (± 0.0002)	0.497 (± 0.0006)
ALCA	0.153 (± 0.0011)	0.148 (± 0.0024)	0.155 (± 0.0040)	0.144 (± 0.0019)	0.175 (± 0.0025)
JM	**1**	**2**	**3**	**4**	**5**
Comorbidities					
Liver Disease	0.181	0.186	0.197	0.2	0.207
Asthma	0.272	0.261	0.258	0.245	0.24
Arthropathy	0.134	0.142	0.148	0.155	0.154
Bone Fracture	0.006	0.007	0.007	0.009	0.01
Raised Liver Enzymes	0.163	0.16	0.156	0.157	0.172
Osteopenia	0.245	0.255	0.266	0.278	0.28
Osteoporosis	0.144	0.149	0.151	0.146	0.134
Hypertension	0.123	0.13	0.141	0.142	0.142
Diabetes	0.319	0.334	0.342	0.348	0.356
Infections					
Burkholderia Cepacia	0.054	0.058	0.056	0.056	0.062
Pseudomonas Aeruginosa	0.636	0.641	0.65	0.655	0.649
Haemophilus Influenza	0.181	0.204	0.233	0.231	0.202
Aspergillus	0.22	0.22	0.218	0.212	0.216
NTM	0.076	0.068	0.072	0.062	0.041
Ecoli	0.098	0.037	0.025	0.011	0.005
Klebsiella Pneumoniae	0.051	0.037	0.026	0.025	0.027
Gram-Negative	0.009	0.01	0.012	0.012	0.015
Xanthomonas	0.079	0.079	0.087	0.092	0.098
Staphylococcus Aureus	0.336	0.337	0.344	0.347	0.345
ALCA	0.037	0.04	0.037	0.04	0.047

Chapter 5

Learning Time-Dependent Causal Effects

Publications Included

- **B. Lim**, A. Alaa, M. van der Schaar. *Forecasting Treatment Responses Over Time Using Marginal Structural Models*. Advances in Neural Information Processing Systems (NeurIPS), 2018. Weblink: `http://papers.nips.cc/paper/7977-forecasting-treatment-responses-over-time-using-recurrent-marginal-structural-networks`.

Forecasting Treatment Responses Over Time Using Recurrent Marginal Structural Networks

Bryan Lim
Department of Engineering Science
University of Oxford
bryan.lim@eng.ox.ac.uk

Ahmed Alaa
Electrical Engineering Department
University of California, Los Angeles
ahmedmalaa@ucla.edu

Mihaela van der Schaar
University of Oxford
and The Alan Turing Institute
mschaar@turing.ac.uk

Abstract

Electronic health records provide a rich source of data for machine learning methods to learn dynamic treatment responses over time. However, any direct estimation is hampered by the presence of time-dependent confounding, where actions taken are dependent on time-varying variables related to the outcome of interest. Drawing inspiration from marginal structural models, a class of methods in epidemiology which use propensity weighting to adjust for time-dependent confounders, we introduce the Recurrent Marginal Structural Network - a sequence-to-sequence architecture for forecasting a patient's expected response to a series of planned treatments. Using simulations of a state-of-the-art pharmacokinetic-pharmacodynamic (PK-PD) model of tumor growth [12], we demonstrate the ability of our network to accurately learn unbiased treatment responses from observational data – even under changes in the policy of treatment assignments – and performance gains over benchmarks.

1 Introduction

With the increasing prevalence of electronic health records, there has been much interest in the use of machine learning to estimate treatment effects directly from observational data [13, 41, 44, 2]. These records, collected over time as part of regular follow-ups, provide a more cost-effective method to gather insights on the effectiveness of past treatment regimens. While the majority of previous work focuses on the effects of interventions at a single point in time, observational data also captures information on complex time-dependent treatment scenarios, such as where the efficacy of treatments changes over time (e.g. drug resistance in cancer patients [40]), or where patients receive multiple interventions administered at different points in time (e.g. joint prescriptions of chemotherapy and radiotherapy [12]). As such, the ability to accurately estimate treatment effects over time would allow doctors to determine both the treatments to prescribe and the optimal time at which to administer them.

However, straightforward estimation in observational studies is hampered by the presence of time-dependent confounders, arising in cases where interventions are contingent on biomarkers whose value are affected by past treatments. For examples, asthma rescue drugs provide short-term rapid improvements to lung function measures, but are usually prescribed to patients with reduced lung function scores. As such, naïve methods can lead to the incorrect conclusion that the medication reduces lung function scores, contrary to the actual treatment effect [26]. Furthermore, [23] show

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

that the standard adjustments for causal inference, e.g. stratification, matching and propensity scoring [16], can introduce bias into the estimation in the presence of time-dependent confounding.

Marginal structural models (MSMs) are a class of methods commonly used in epidemiology to estimate time-dependent effects of exposure while adjusting for time-dependent confounders [15, 24, 19, 14]. Using the probability of a treatment assignment, conditioned on past exposures and covariate history, MSMs typically adopt inverse probability of treatment weighting (IPTW) to correct for bias in standard regression methods [22], re-constructing a 'pseudo-population' from the observational dataset to similar to that of a randomized clinical trial. However, the effectiveness of bias correction is dependent on a correct specification of the conditional probability of treatment assignment, which is difficult to do in practice given the complexity of treatment planning. In standard MSMs, IPTWs are produced using pooled logistic regression, which makes strong assumptions on the form of the conditional probability distribution. This also requires one separate set of coefficients to be estimated per time-step and many models to be estimated for long trajectories.

In this paper, we propose a new deep learning model - which we refer to as Recurrent Marginal Structural Networks - to directly learn time-dependent treatment responses from observational data, based on in the marginal structural modeling framework. Our key contributions are as follows:

Multi-step Prediction Using Sequence-to-sequence Architecture To forecast treatment responses at multiple time horizons in the future, we propose a new RNN architecture for multi-step prediction based on sequence-to-sequence architectures in natural language processing [36]. This comprises two halves, 1) an encoder RNN which learns representations for the patient's current clinical state, and 2) a decoder which is initialized using the encoder's final memory state and computes forward predictions given the intended treatment assignments. At run time, the R-MSN also allows for prediction horizons to be flexibly adjusted to match the intended treatment duration, by expanding or contracting the number of decoder units in the sequence-to-sequence model.

Scenario Analysis for Complex Treatment Regimens Treatment planning in clinical settings is often based on the interaction of numerous variables - including 1) the desired outcomes for a patient (e.g. survival improvement or comorbidity risk reduction), 2) the treatments to assign (e.g. binary interventions or continuous dosages), and 3) the length of treatment affected by both number and duration of interventions. The R-MSN naturally encapsulates this by using multi-input/output RNNs, which can be configured to have multiples treatments and targets of different forms (e.g. continuous or discrete). Different sequences of treatments can also be evaluated using the sequence-to-sequence architecture of the network. Moreover, given the susceptibility of IPTWs to model misspecification, the R-MSN uses Long-short Term Memory units (LSTMs) to compute the probabilities required for propensity weighting. Combining these aspects together, the R-MSN is able to help clinicians evaluate the projected outcome of a complex treatment scenario – providing timely clinical decision support and helping them customize a treatment regimen to the patient. A example of scenario analysis for different cancer treatment regimens is shown in Figure 1, with the expected response of tumor growth to no treatment, chemotherapy and radiotherapy shown.

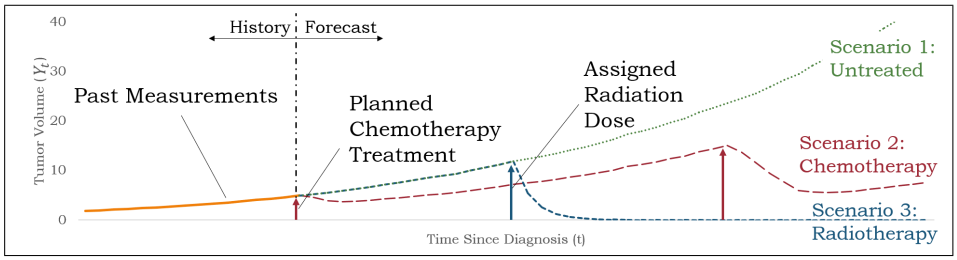


Figure 1: Forecasting Tumor Growth Under Multiple Treatment Scenarios

2 Related Works

Given the diversity of literature on causal inference, we focus on works associated with time-dependent treatment responses and deep learning here, with a wider survey in Appendix A.

G-computation and Structural Models. Counterfactual inference under time-dependent confounding has been extensively studied in the epidemiology literature, particularly in the seminal works of Robins [30, 31, 16]. Methods in this area can be categorized into 3 groups: models based on the G-computation formula, structural nested mean models, and marginal structural models [8]. While all these models provide strong theoretical foundations on the adjustments for time-dependent confounding, their prediction models are typically based on linear or logistic regression. These models would be misspecified when either the outcomes or the treatment policy exhibit complex dependencies on the covariate history.

Potential Outcomes with Longitudinal Data. Bayesian nonparametric models have been proposed to estimate the effects of both single [32, 33, 43, 34] and joint treatment assignments [35] over time. These methods use Gaussian processes (GPs) to model the baseline progression, which can estimate the treatment effects at multiple points in the future. However, some limitations do exist. Firstly, to aid in calibration, most Bayesian methods make strong assumptions on model structure - such as 1) independent baseline progression and treatment response components [43, 35], and 2) the lack of heterogeneous effects, by either omitting baseline covariates (e.g. genetic or demographic information) [34, 33] or incorporating them as linear components [43, 35]. Recurrent neural networks (RNNs) avoid the need for any explicit model specifications, with the networks learning these relationships directly from the data. Secondly, inference with Bayesian models can be computationally complex, making them difficult to scale. This arises from the use of Markov Chain-Monte Carlo sampling for g-computation, and the use of sparse GPs that have at least $O(NM^2)$ complexity, where N and M are the number of observations and inducing points respectively [39]. From this perspective, RNNs have the benefit of scalability and update their internal states with new observations as they arrive. Lastly, apart from [35] which we evaluate in Section 5, existing models do not consider treatment responses for combined interventions and multiple targets. This is handled naturally in our network by using multi-input/multi-output RNN architectures.

Deep Learning for Causal Inference. Deep learning has also been used to estimate individualized treatment effects for a single intervention at a fixed time, using instrumental variable approaches [13], generative adversarial networks [44] and multi-task architectures [3]. To the best of our knowledge, ours is the first deep learning method for time-dependent effects and establishes a framework to use existing RNN architectures for treatment response estimation.

3 Problem Definition

Let $\mathbf{Y}_{t,i} = [Y_{t,i}(1), \ldots, Y_{t,i}(\Omega_y)]$ be a vector of Ω_y observed outcomes for patient i at time t, $\mathbf{A}_{t,i} = [A_{t,i}(1), \ldots, A_{t,i}(\Omega_a)]$ a vector of actual treatment administered, $\mathbf{L}_{t,i} = [L_{t,i}(1), \ldots, L_{t,i}(\Omega_l)]$ time-dependent covariates and $\mathbf{X}_i = [X_i(1), \ldots, X_i(\Omega_v)]$ patient-specific static features. For notational simplicity, we will omit the subscript i going forward unless explicitly required.

Treatment Responses Over Time Determining an individual's response to a prescribed treatment can be characterized as learning a function $g(.)$ for the expected outcomes over a prediction horizon τ, given an intended course of treatment and past observations, i.e.:

$$\mathbb{E}\left[\mathbf{Y}_{t+\tau} | a(t, \tau-1), \bar{\mathbf{H}}_t\right] = g(\tau, a(t, \tau-1), \bar{\mathbf{H}}_t) \tag{1}$$

where $g(.)$ represents a generic, possibly non-linear, function, $a(t, \tau-1) = (\mathbf{a}_t, \ldots \mathbf{a}_{t+\tau-1})$ is an *intended* sequence of treatments $\mathbf{a}_k$ from the current time until just before the outcome is observed, and $\bar{\mathbf{H}}_t = (\bar{\mathbf{L}}_t, \bar{\mathbf{A}}_{t-1}, \mathbf{X})$ is the patient's history with covariates $\bar{\mathbf{L}}_t = (\mathbf{L}_1, \ldots, \mathbf{L}_t)$ and actions $\bar{\mathbf{A}}_{t-1} = (\mathbf{A}_1, \ldots \mathbf{A}_{t-1})$.

Inverse Probability of Treatment Weighting Inverse probability of treatment weighting, extensively studied in marginal structural modeling to adjust for time-dependent confounding [22, 16, 15, 24, 26], with extensions to joint treatment assignments [19], censored observations [14] and continuous dosages [10]. We list the key results for our problem below, with a more thorough discussion in Appendix B.

The stabilized weights for joint treatment assignments [21] can be expressed as:

$$\mathbf{SW}(t, \tau) = \prod_{n=t}^{t+\tau} \frac{f(\mathbf{A}_n | \bar{\mathbf{A}}_{n-1})}{f(\mathbf{A}_n | \bar{\mathbf{H}}_n)} = \prod_{n=t}^{t+\tau} \frac{\prod_{k=1}^{\Omega_a} f(A_n(k) | \bar{\mathbf{A}}_{n-1})}{\prod_{k=1}^{\Omega_a} f(A_n(k) | \bar{\mathbf{H}}_n)} \tag{2}$$

where $f(.)$ is the probability mass function for discrete treatment applications, or the probability density function when continuous dosages are used [10]. We also note that $\bar{\mathbf{H}}_n$ contains both past treatments $\bar{\mathbf{A}}_{n-1}$ and potential confounders $\bar{\mathbf{L}}_n$. To account for censoring, we used the additional stabilized weights below:

$$SW^*(t,\tau) = \prod_{n=t}^{t+\tau} \frac{f(C_n = 0|\mathcal{T} > n, \bar{\mathbf{A}}_{\mathbf{n-1}})}{f(C_n = 0|\mathcal{T} > n, \bar{\mathbf{L}}_{n-1}, \bar{\mathbf{A}}_{n-1}, \mathbf{X})} \tag{3}$$

where $C_n = 1$ denotes right censoring of the trajectory, and $\mathcal{T}$ is the time at which censoring occurs.

We also adopt the additional steps for stabilization proposed in [42], truncating stabilized weights at their 1st and 99th percentile values, and normalizing weights by their mean for a fixed prediction horizon, i.e. $\tilde{\mathbf{SW}} = \mathbf{SW}_i(t,\tau) / \left(\sum_{i=1}^{I} \sum_{t=1}^{T_i} \mathbf{SW}_i(t,\tau)/N \right)$ where I is the total number of patients, T_i is the length of the patient's trajectory and N the total number of observations. Stabilized weights are then used to weight the loss contributions of each training observation, expressed in squared-errors terms below for continuous predictions:

$$e(i,t,\tau) = \tilde{\mathbf{SW}}_i(t,\tau-1) \times \tilde{SW}_i^*(t,\tau-1) \times \|\mathbf{Y}_{t+\tau,i} - g(\tau, a(t,\tau-1), \bar{\mathbf{H}}_t)\|^2 \tag{4}$$

4 Recurrent Marginal Structural Networks

An MSM can be subdivided into two submodels, one modeling the IPTWs and the other estimating the treatment response itself. Adopting this framework, we use two sets of deep neural networks to build a Recurrent Marginal Structural Network (R-MSN) - 1) a set propensity networks to compute treatment probabilities used for IPTW, and 2) a prediction network used to determine the treatment response for a given set of planned interventions. Additional details on the algorithm can be found in Appendix E, with the source code uploaded onto GitHub[1].

4.1 Propensity Networks

From Equations 2 and 3, we can see that 4 key probability functions are required to calculate the stabilized weights. In all instances, probabilities are conditioned on the history of past observations ($\bar{\mathbf{A}}_{n-1}$ and $\bar{\mathbf{H}}_n$), making RNNs natural candidates to learn these functions.

Each probability function is parameterized with a different LSTM – collectively referred to as propensity networks – with action probabilities $f\left(\bar{\mathbf{A}}_n| \, . \right)$ generated jointly by a set of multi-target LSTMs and censoring probabilities $f\left(C_n = 0| \, . \right)$ by single output LSTMs. This also accounts for possible correlations between treatment assignments, for instance in treatment regimens where complementary drugs are prescribed together to combat different aspects of the same disease.

The flexibility of RNN architectures also allows for the modeling of treatment assignments with different forms. In simple cases with discrete treatment assignments, a standard LSTM with a sigmoid output layer can be used for binary treatment probabilities or a softmax layer for categorical ones. More complex architectures, such as variational RNNs [6], can be used to compute probabilities when treatments map to continuous dosages. To calculate the binary probabilities in the experiments in Section 5, LSTMs were fitted with tanh state activations and sigmoid outputs.

4.2 Prediction Network

The prediction network focuses on forecasting the treatment response of a patient, with time-dependent confounding accounted for using IPTWs from the propensity networks. Although standard RNNs can be used for one-step-ahead forecasts, actual treatments plans can be considerably more complex, with varying durations and number of interventions depending on the condition of the patient. To remove any restrictions on the prediction horizon or number of planned interventions, we propose the sequence-to-sequence architecture depicted in Figure 4.2. One key difference between our model and standard sequence-to-sequence (e.g.[36]) is that the last unit of the encoder is also used in making predictions for the first time step, in addition to the decoder units at further horizons. This allows the R-MSN to use all available information in making predictions, including the covariates

[1] `https://github.com/sjblim/rmsn_nips_2018`

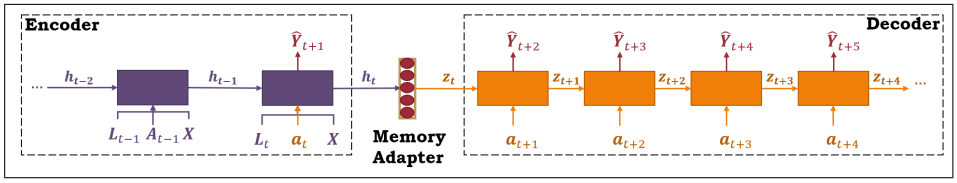


Figure 2: R-MSN Architecture for Multi-step Treatment Response Prediction

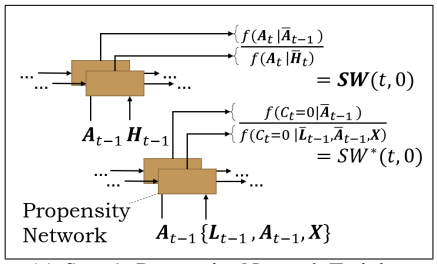
available at the current time step t. For the continuous predictions in Section 5, we used Exponential Linear Unit (ELU [7]) state activations and a linear output layer.

Encoder The goal of the encoder is to learn good representations for the patient's current clinical state, and we do so with a standard LSTM that makes one-step-ahead predictions of the outcome ($\hat{\mathbf{Y}}_{t+1}$) given observations of covariates and actual treatments. At the current follow-up time t, the encoder is also used in forecasting the expected response at $t+1$, as the latest covariate measurements L_t are available to be fed into the LSTM along with the first planned treatment assignment.

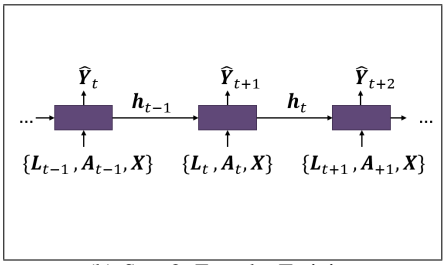
Decoder While multi-step prediction can be performed by recursively feeding outputs into the inputs at the next time step, this would require output predictions for *all* covariates, with a high degree of accuracy to reduce error propagation through the network. Given that often only a small subset treatment outcomes are of interest, it would be desirable to forecast treatment responses on the basis of planned future actions alone. As such, the purpose of the decoder is to propagate the encoder representation forwards in time - using only the proposed treatment assignments and avoiding the need to forecast input covariates. This is achieved by training another LSTM that accepts only actions as inputs, but initializing the internal memory state of the first LSTM in the decoder sequence (z_t) using encoder representations. To allow for different state sizes in the encoder and decoder, encoder internal states (h_t) are passed through a single network layer with ELU activations, i.e. the memory adapter, before being initializing the decoder. As the network is made up of LSTM units, the internal states here refer to the concatenation of the cell and hidden states [17] of the LSTM.

4.3 Training Procedure

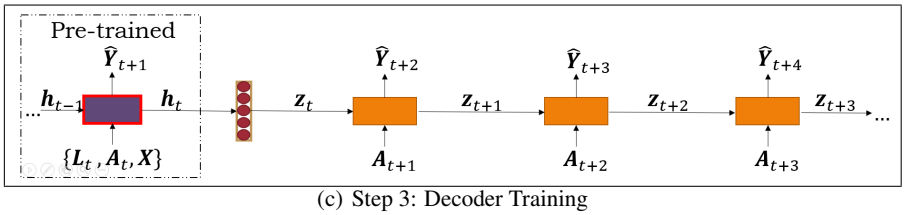
The training procedure for R-MSNs can be subdivided into the 3 training steps shown in Figure 3 - starting with the propensity networks, followed by the encoder, and ending with the decoder.



(a) Step 1: Propensity Network Training



(b) Step 2: Encoder Training



(c) Step 3: Decoder Training

Figure 3: Training Procedure for R-MSNs

Step 1: Propensity Network Training From Figure 3(a), each propensity network is first trained to estimate the probability of the treatment assigned at each time step, which is combined to compute $\mathbf{SW}(t,0)$ and $SW^*(t,0)$ at each time step. Stabilized weights for longer horizons can then be obtained from their cumulative product, i.e. $\mathbf{SW}(t,\tau) = \prod_{j=0}^{\tau} \mathbf{SW}(t+j,0)$. For tests in Section 5, propensity networks were trained using standard binary cross entropy loss, with treatment assignments and censoring treated as binary observations.

Step 2: Encoder Training Next, decoder and encoder training was divided into separate steps - accelerating learning by first training the encoder to learn representations of the patient's clinical state and then using the decoder to extrapolate them according to the intended treatment plan. As such, the encoder was trained to forecast standard one-step-ahead treatment response according to the structure in Figure 3(b), using all available information on treatments and covariates until the current time step. Upon completion, the encoder was used to perform a feed-forward pass over the training and validation data, extracting the internal states $\mathbf{h}_t$ for the final training step. As tests in Section 5 were performed for continuous outcomes, we express the loss function for the encoder as a weighted mean-squared error loss ($\mathcal{L}_{encoder}$ in Equation 5), although we note that this approach is compatible with other loss functions, e.g. cross entropy for discrete outcomes.

Step 3: Decoder Training Finally, the decoder and memory adapter were trained together based on the format in Figure 3(c). For a given patient, observations were batched into shorter sequences of up to τ_{max} steps, such that each sequence commencing at time t is made up of $[\mathbf{h}_t, \{\mathbf{A}_{t+1}, \ldots, \mathbf{A}_{t+\tau_{max}-1}\}, \{\mathbf{Y}_{t+2}, \ldots, \mathbf{Y}_{t+\tau_{max}}\}]$. These were compiled for all patient-times and randomly grouped into minibatches to be used for backpropagation through time. For continuous predictions, the loss function for the decoder is ($\mathcal{L}_{decoder}$) can also be found in Equation 5.

$$\mathcal{L}_{encoder} = \sum_{i=1}^{I} \sum_{t=1}^{T_i} e(i,t,1) \qquad \mathcal{L}_{decoder} = \sum_{i=1}^{I} \sum_{t=1}^{T_i} \sum_{\tau=2}^{min(T_i-t,\tau_{max})} e(i,t,\tau) \tag{5}$$

5 Experiments With Cancer Growth Simulation Model

5.1 Simulation Details

As confounding effects in real-world datasets are unknown a priori, methods for treatment response estimation are often evaluated using data simulations, where treatment application policies are explicitly modeled [34, 33, 35]. To ensure that our tests are fully reproducible and realistic from a medical perspective, we adopt the pharmacokinetic-pharmacodynamic (PK-PD) model of [12] - the state-of-the-art in treatment response modeling for non-small cell lung patients. The model features key characteristics present in actual lung cancer treatments, such as combined effects of chemo- and radiotherapy, cell repopulation after treatment, death/recovery of patients, and different staring distributions of tumor sizes based on the stage of cancer at diagnosis. On the whole, PK-PD models allow clinicians to explore hypotheses around dose-response relationships and propose optimal treatment schedules [5, 29, 11, 9, 1]. While we refer readers to [12] for the finer details of the model, such as specific priors used, we examine the overall structure of the model below to illustrate treatment-response relationships and how time-dependent confounding is introduced.

PK-PD Model for Tumor Dynamics We use a discrete-time model for tumor volume $V(t)$, where t is the number of days since diagnosis:

$$V(t) = \Bigg(1 + \underbrace{\rho \log(\frac{K}{V(t-1)})}_{\text{Tumor Growth}} - \underbrace{\beta_c C(t)}_{\text{Chemotherapy}} - \underbrace{(\alpha d(t) + \beta d(t)^2)}_{\text{Radiation}} + \underbrace{e_t}_{\text{Noise}}\Bigg) V(t-1) \tag{6}$$

where ρ, K, β_c, α, β are model parameters sampled for each patient according to prior distributions in [12]. A Gaussian noise term $e_t \sim N(0, 0.01^2)$ was added to account for randomness in the growth of the tumor. $d(t)$ is the dose of radiation applied at t, while drug concentration C(t) is modeled according to an exponential decay with a half life of 1 day, i.e.:

$$C(t) = \tilde{C}(t) + C(t-1)/2 \tag{7}$$

where $\tilde{C}(t)$ is an new continuous dose of chemotherapy drugs applied at time t. To account for heterogeneous effects, we added static features to the simulation model by randomly subclassing

patients into 3 different groups, with each patient having a group label $S_i \in \{1, 2, 3\}$. This represents specific characteristics which affect with patient's response to chemotherapy and radiotherapy (e.g. by genetic factors [4]), which augment the prior means of β_c and α according to:

$$\mu'_{\beta_c}(i) = \begin{cases} 1.1\mu_{\beta_c} \text{ , if } S_i = 3 \\ \mu_{\beta_c} \text{ , otherwise} \end{cases} \qquad \mu'_{\alpha}(i) = \begin{cases} 1.1\mu_{\alpha} \text{ , if } S_i = 1 \\ \mu_{\alpha} \text{ , otherwise} \end{cases} \tag{8}$$

where μ_* are the mean parameters of [12], and $\mu'_*(i)$ those used to simulate patient i. We note that the value of β is set in relation to α, i.e. $\alpha/\beta = 10$, and would also be adjusted accordingly by S_i.

Censoring Mechanisms Patient censoring is incorporated by modeling 1) death when tumor diameters reach $D_{max} = 13\ cm$ (or a volume of $V_{max} = 1150\ cm^3$ assuming perfectly spherical tumors), 2) recovery determined by a Bernoulli process with recovery probability $p_t = \exp(-V_t)$, and 3) termination of observations after 60 days (administrative censoring).

Treatment Assignment Policy To introduce time-dependent confounders, we assume that chemotherapy prescriptions $A_c(t) \in \{0, 1\}$ and radiotherapy prescriptions $A_d(t) \in \{0, 1\}$ are Bernoulli random variables, with probabilities $p_c(t)$ and $p_d(t)$ respectively that are a functions of the tumor diameter:

$$p_c(t) = \sigma\left(\frac{\gamma_c}{D_{max}}(\bar{D}(t) - \theta_c)\right) \qquad p_d(t) = \sigma\left(\frac{\gamma_d}{D_{max}}(\bar{D}(t) - \theta_d)\right) \tag{9}$$

where $\bar{D}(t)$ is the average tumor diameter over the last 15 days, $\sigma(.)$ is the sigmoid activation function, and θ_* and γ_* are constant parameters. θ_* is fixed such that $\theta_c = \theta_d = D_{max}/2$, giving the model a 0.5 probability of treatment application exists when the tumor is half its maximum size. When treatments are applied, i.e. $A_c(t)$ or $A_d(t)$ is 1, chemotherapy is assumed to be administered in $5.0\ mg/m^3$ doses of Vinblastine, and radiotherapy in $2.0\ Gy$ fractions. γ also controls the degree of time-dependent confounding - starting with no confounding at $\gamma = 0$, as treatment assignments are independent of the response variable, and an increase as γ becomes larger.

5.2 Benchmarks

We evaluate the performance of R-MSNs against MSMs and Bayesian nonparametric models, focusing on its effectiveness in estimating unbiased treatment responses and its multi-step prediction performance. An overview of the models tested is summarized below:

Standard Marginal Structural Models (MSM) For the MSMs used in our investigations, we adopt similar approximations to [19, 14], encoding historical actions via cumulative sum of applied treatments, e.g. $cum(\bar{a}_c(t-1)) = \sum_{k=1}^{t-1} a_c(k)$, and covariate history using the previous observed value $V(t-1)$. The exact forms of the propensity and prediction models are in Appendix D.

Bayesian Treatment Response Curves (BTRC) We also benchmark our performance against the model of [35] - the state-of-the-art in forecasting multistep treatment responses for joint therapies with multiple outcomes. Given that the simulation model only has one target outcome, we also consider a simpler variant of the model without "shared" components, denoting this as the reduced BTRC (R-BTRC) model. This reduced parametrization was found to improve convergence during training, and additional details on calibration can be found in Appendix G.

Recurrent Marginal Structural Networks (R-MSN) R-MSNs were designed according to the description in Section 4, with full details on training and hyperparameter in Appendix F. To evaluate the effectiveness of the propensity networks, we also trained predictions networks using the IPTWs from the MSM, including this as an additional benchmark in Section 5.3 (Seq2Seq + Logistic).

5.3 Performance Evaluations

Time-Dependent Confounding Adjustments To investigate how well models learn unbiased treatment responses from observational data, we trained all models on simulations with $\gamma_c = \gamma_d = 10$ (biased policy) and examine the root-mean-squared errors (RMSEs) of one-step-ahead predictions as γ_* is reduced. Both γ_* parameters were set to be equal in this section for simplicity, i.e. $\gamma_c = \gamma_d = \gamma$. Using the simulation model in Section 5.1, we simulated 10,000 paths to be used for model training, 1,000 for validation data used in hyperparameter optimization, and another 1,000 for out-of-sample

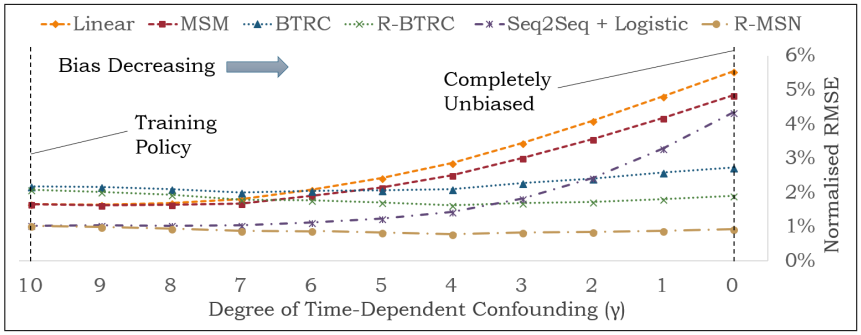


Figure 4: Normalized RMSEs for One-Step-Ahead Predictions

testing. For linear and MSM models, which do not have hyperparameters to optimized, we combined both training and validation datasets for model calibration.

Figure 4 shows the RMSE values of various models at different values of γ, with RMSEs normalized with V_{max} and reported in percentage terms. Here, we focus on the main comparisons of interest – 1) linear models to provide a baseline on performance, 2) linear vs MSMs to evaluate traditional methods for IPTWs, 3) Seq2Seq + logistic IPTWs vs MSMs for the benefits of the Seq2Seq model, 4) R-MSN vs Seq2Seq + logistic to determine the improvements of our model and RNN-estimated IPTWs, and 5) BTRC/R-BTRC to benchmark against state-of-the-art methods. Additional results are also documented in Appendix C for reference.

From the graph, R-MSNs displayed the lowest RMSEs across all values of γ, decreasing slightly from a normalized RMSE of 1.02% at $\gamma = 10$ to 0.92% at $\gamma = 0$. Focusing on RMSEs at $\gamma = 0$, R-MSNs improve MSMs by 80.9% and R-BTCs by 66.1%, demonstrating its effectiveness in learning unbiased treatment responses from confounded data. The propensity networks also improve unbiased treatment estimates by 78.7% (R-MSN vs. Seq2Seq + Logistic), indicating the benefits of more flexible models for IPTW estimation. While the IPTWs of MSMs do provide small gains for linear models, linear models still exhibit the largest unbiased RMSE across all benchmarks - highlighting the limitations of linear models in estimating complex treatment responses. Bayesian models also perform consistently across γ, with normalized RMSEs for R-BTRC decreasing from 2.09% to 1.91% across $\gamma = 0$ to 10, but were also observed to slightly underperform linear models on the training data itself. Part of this can potentially be attributed to model misspecification in the BTRC, which assumes that treatment responses are linear time-invariant and independent of the baseline progression. The differences in modeling assumptions can be seen from Equation 6, where chemotherapy and radiotherapy contributions are modeled as multiplicative with $V(t)$. This highlights the benefits of the data-driven nature of the R-MSN, which can flexibly learn treatment response models of different types.

Multi-step Prediction Performance To evaluate the benefits of the sequence-to-sequence architecture, we report the normalized RMSEs for multi-step prediction in Table 1, using the best model of each category (R-MSN, MSM and R-BTRC). Once again, the R-MSN outperforms benchmarks for all timesteps, beating MSMs by 61% on the training policy and 95% for the unbiased one. While the R-BTRC does show improvements over MSMs for the unbiased treatment response, we also observe a slight underperformance versus MSMs on the training policy itself, highlighting the advantages of R-MSNs.

6 Conclusions

This paper introduces Recurrent Marginal Structural Networks - a novel learning approach for predicting unbiased treatment responses over time, grounded in the framework of marginal structural models. Networks are subdivided into two parts, a set of propensity networks to accurately compute the IPTWs, and a sequence-to-sequence architecture to predict responses using only a planned sequence of future actions. Using tests on a medically realistic simulation model, the R-MSN demonstrated performance improvements over traditional methods in epidemiology and the state-of-the-art models for joint treatment response prediction over multiple timesteps.

Table 1: Normalized RMSE for Various Prediction Horizons τ

	τ	**1**	**2**	**3**	**4**	**5**	**Ave. % Decrease in RMSE vs MSMs**
Training Policy ($\gamma_c = 10, \gamma_d = 10$)	MSM	1.67%	2.51%	3.12%	3.64%	4.09%	-
	R-BTRC	2.09%	2.85%	3.50%	4.07%	4.58%	-32% (↑ RMSE)
	R-MSN	**1.02%**	**1.80%**	**1.90%**	**2.11%**	**2.46%**	**+61%**
Unbiased Assignment ($\gamma_c = 0, \gamma_d = 0$)	MSM	4.84%	5.29%	5.51%	5.65%	5.84%	-
	R-BTRC	1.91%	2.74%	3.34%	3.75%	4.08%	+66%
	R-MSN	**0.92%**	**1.38%**	**1.30%**	**1.22%**	**1.14%**	**+95%**
Unbiased Radiotherapy ($\gamma_c = 10, \gamma_d = 0$)	MSM	3.85%	4.03%	4.32%	4.60%	4.91%	-
	R-BTRC	1.74%	1.68%	2.14%	2.54%	2.91%	+74%
	R-MSN	**1.08%**	**1.66%**	**1.83%**	**1.98%**	**2.14%**	**+84%**
Unbiased Chemotherapy ($\gamma_c = 0, \gamma_d = 10$)	MSM	1.84%	2.65%	3.09%	3.44%	3.83%	-
	R-BTRC	1.16%	2.45%	2.97%	3.34%	3.64%	+20%
	R-MSN	**0.65%**	**1.13%**	**1.05%**	**1.17%**	**1.31%**	**+87%**

Acknowledgments

This research was supported by the Oxford-Man Institute of Quantitative Finance, the US Office of Naval Research (ONR), and the Alan Turing Institute.

References

[1] Optimizing drug regimens in cancer chemotherapy: a simulation study using a pk–pd model. *Computers in Biology and Medicine*, 31(3):157 – 172, 2001. Goal-Oriented Model-Based drug Regimens.

[2] Ahmed M. Alaa and Mihaela van der Schaar. Bayesian inference of individualized treatment effects using multi-task gaussian processes. In *Proceedings of the thirty-first Conference on Neural Information Processing Systems, (NIPS)*, 2017.

[3] Ahmed M. Alaa, Michael Weisz, and Mihaela van der Schaar. Deep counterfactual networks with propensity dropout. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[4] H. Bartsch, H. Dally, O. Popanda, A. Risch, and P. Schmezer. Genetic risk profiles for cancer susceptibility and therapy response. *Recent Results Cancer Res.*, 174:19–36, 2007.

[5] Letizia Carrara, Silvia Maria Lavezzi, Elisa Borella, Giuseppe De Nicolao, Paolo Magni, and Italo Poggesi. Current mathematical models for cancer drug discovery. *Expert Opinion on Drug Discovery*, 12(8):785–799, 2017.

[6] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2980–2988, Cambridge, MA, USA, 2015.

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *CoRR*, abs/1511.07289, 2015.

[8] R.M. Daniel, S.N. Cousens, B.L. De Stavola, M. G. Kenward, and J. A. C. Sterne. Methods for dealing with time-dependent confounding. *Statistics in Medicine*, 32(9):1584–1618, 2012.

[9] Mould DR, Walz A-C, Lave T, Gibbs JP, and Frame B. Developing exposure/response models for anticancer drug treatment: Special considerations. *CPT: Pharmacometrics & Systems Pharmacology*, 4(1):12–27.

[10] Peter H. Egger and Maximilian von Ehrlich. Generalized propensity scores for multiple continuous treatment variables. *Economics Letters*, 119(1):32 – 34, 2013.

[11] M. J. Eigenmann, N. Frances, T. Lavé, and A.-C. Walz. Pkpd modeling of acquired resistance to anti-cancer drug treatment. *Journal of Pharmacokinetics and Pharmacodynamics*, 44(6):617–630, 2017.

[12] Changran Geng, Harald Paganetti, and Clemens Grassberger. Prediction of treatment response for combined chemo- and radiation therapy for non-small cell lung cancer patients using a bio-mathematical model. *Scientific Reports*, 7, 2017.

[13] Jason Hartford, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy. Deep IV: A flexible approach for counterfactual prediction. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[14] Miguel A. Hernan, Babette Brumback, and James M. Robins. Marginal structural models to estimate the joint causal effect of nonrandomized treatments. *Journal of the American Statistical Association*, 96(454):440–448, 2001.

[15] Miguel A. Hernan and James M. Robins. Marginal structural models to estimate the causal effect of zidovudine on the survival of hiv-positive men. *Epidemiology*, 359:561–570, 2000.

[16] MA Hernán and JM Robins. *Causal Inference*. Chapman & Hall/CRC, 2018.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[18] William Hoiles and Mihaela van der Schaar. A non-parametric learning method for confidently estimating patient's clinical state and dynamics. In *Proceedings of the twenty-ninth Conference on Neural Information Processing Systems, (NIPS)*, 2016.

[19] Chanelle J. Howe, Stephen R. Cole, Shruti H. Mehta, and Gregory D. Kirk. Estimating the effects of multiple time-varying exposures using joint marginal structural models: alcohol consumption, injection drug use, and hiv acquisition. *Epidemiology*, 23(4):574–582, 2012.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[21] Clovis Lusivika-Nzinga, Hana Selinger-Leneman, Sophie Grabar, Dominique Costagliola, and Fabrice Carrat. Performance of the marginal structural cox model for estimating individual and joined effects of treatments given in combination. *BMC Medical Research Methodology*, 17(1):160, Dec 2017.

[22] Mohammad Ali Mansournia, Mahyar Etminan, Goodarz Danaei, Jay S Kaufman, and Gary Collins. A primer on inverse probability of treatment weighting and marginal structural models. *Emerging Adulthood*, 4(1):40–59, 2016.

[23] Mohammad Ali Mansournia, Mahyar Etminan, Goodarz Danaei, Jay S Kaufman, and Gary Collins. Handling time varying confounding in observational research. *BMJ*, 359, 2017.

[24] Mohammad Alia Mansournia, Goodarzc Danaei, Mohammad Hosseind Forouzanfar, Mahmoodb Mahmoodi, Mohsene Jamali, Nasrinf Mansournia, and Kazemb Mohammad. Effect of physical activity on functional performance and knee pain in patients with osteoarthritis : Analysis with marginal structural models. *Epidemiology*, 23(4):631–640, 2012.

[25] Charles E McCulloch and Shayle R. Searle. *Generalized, Linear and Mixed Models*. Wiley, New York, 2001.

[26] Kathleen M. Mortimer, Romain Neugebauer, Mark van der Laan, and Ira B. Tager. An application of model-fitting procedures for marginal structural models. *American Journal of Epidemiology*, 162(4):382–388, 2005.

[27] Mihaela van der Schaar Onur Atan, James Jordon. Deep-treat: Learning optimal personalized treatments from observational data using neural networks. In *AAAI*, 2018.

[28] Mihaela van der Schaar Onur Atan, William Zame. Constructing effective personalized policies using counterfactual inference from biased data sets with many features. In *Machine Learning*, 2018.

[29] Kyungsoo Park. A review of modeling approaches to predict drug response in clinical oncology. *Yonsei Medical Journal*, 58(1):1–8, 2017.

[30] Thomas S. Richardson and Andrea Rotnitzky. Causal etiology of the research of james m. robins. *Statistical Science*, 29(4):459–484, 2014.

[31] James M. Robins, Miguel Ángel Hernán, and Babette Brumback. Marginal structural models and causal inference in epidemiology. *Epidemiology*, 11(5):550–560, 2000.

[32] Jason Roy, Kirsten J. Lum, and Michael J. Daniels. A bayesian nonparametric approach to marginal structural models for point treatments and a continuous or survival outcome. *Biostatistics*, 18(1):32–47, 2017.

[33] Peter Schulam and Suchi Saria. Reliable decision support using counterfactual models. In *Proceedings of the thirty-first Conference on Neural Information Processing Systems, (NIPS)*, 2017.

[34] Ricardo Silva. Observational-interventional priors for dose-response learning. In *Proceedings of the Thirtieth Conference on Neural Information Processing Systems, (NIPS)*, 2016.

[35] Hossein Soleimani, Adarsh Subbaswamy, and Suchi Saria. Treatment-response models for counterfactual reasoning with continuous-time, continuous-valued interventions. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the twenty-seventh Conference on Neural Information Processing Systems, (NIPS)*, 2014.

[37] Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research*, 16:1731–1755, 2015.

[38] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. *CoRR*, abs/1502.02362, 2015.

[39] Michalis K. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Proceedings of the twelth International Conference on Artificial Intelligence and Statistic, (AISTATS)*, 2009.

[40] Panagiotis J. Vlachostergios and Bishoy M. Faltas. Treatment resistance in urothelial carcinoma: an evolutionary perspective. *Nature Review Clincal Oncology*, 2018.

[41] Stefan Wager and Susan Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2017.

[42] Yongling Xiao, Michal Abrahamowicz, and Erica Moodie. Accuracy of conventional and marginal structural cox model estimators: A simulation study. *The International Journal of Biostatistics*, 6(2), 2010.

[43] Yanbo Xu, Yanxun Xu, and Suchi Saria. A non-parametric bayesian approach for estimating treatment-response curves from sparse time series. In *Proceedings of the 1st Machine Learning for Healthcare Conference (MLHC)*, 2016.

[44] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. GANITE: Estimation of individualized treatment effects using generative adversarial nets. In *International Conference on Learning Representations (ICLR)*, 2018.

Appendix

A Extended Related Works

Potential Outcomes with Cross-sectional Data. A simpler instantiation of the problem is to estimate the effect of a treatment applied to subjects in a (static) cross-sectional dataset. This problem has recently attracted a lot of attention in the machine learning community, and various interesting ideas were proposed to account for selection bias [3, 41, 44]. Unfortunately, most of these works cast the treatment effect estimation problem as one of learning under "covariate shift", where the goal is to learn a model for the outcomes that generalizes well to a population were treatments are randomly assigned to the subjects. Because of the sequential nature of the treatment assignment process in our setup, estimating treatment effects under time-dependent confounding cannot be similarly framed as a covariate shift problem, and hence the ideas developed in those works cannot be straightforwardly applied to our setup.

Off-policy Evaluation. A closely related problem in the area of reinforcement learning is the problem of off-policy evaluation using retrospective observational data, also known as "logged bandit feedback" [38, 37, 18, 27, 28]. In this problem, the goal is to use sequences of states, actions and rewards generated by a decision-maker that operates under an unknown policy in order to estimate the expected reward of a given policy. In our setting, we focus on estimating a trajectory of outcomes given an application of a treatment (or a sequence of treatments) rather than estimating the average reward of a policy, and hence the "counterfactual risk minimization" framework in [37] would not result in optimal estimates in our setup. However, our learning model – with a different objective function– can be applied for the problem of off-policy evaluation.

B Background on Marginal Structural Models

In this section, we summarize the key relevant points from the seminal paper of Robins [31]. Without loss of generality, we consider the case of univariate treatments, response variables and baseline covariates here for simplicity.

Marginal structural models are typically considered in the context of follow-up studies, for example in patients with HIV [31]. Time in the study is typically measured in relation to a fixed starting point, such as the first follow-up date or time of diagnosis (i.e. $t = 1$). In such settings, marginal structural models are used to measure the average treatment effect conditioned on a series of potential actions and baseline covariate V taken at the start of the study, expressed in the form:

$$\mathbb{E}\left[Y_\tau | a_1, \ldots, a_\tau, V\right] = r(a_1, \ldots, a_\tau, X; \Theta) \tag{10}$$

where $r(.)$ is a generic, typically linear, function with parameters Θ.

Time-Dependent Confounding. A full description of time-varying confounding can be found in [23], with formal definitions in [14]. Time-dependent confounding in observational studies arises as confounders have values which change over time - for example in cases where treatments are moderated based on the patient's response. A causal graph for 2-step study can be found in Figure 5, where U denotes unmeasured factors. Note that U_0, U_1 do not have arrows to actions assignments, reflecting the assumption of no unmeasured confounding.

Inverse Probability of Treatment Weighting From [31], under assumptions of no unmeasured confounding, positivity, and correct model specification, the stabilized IPTWs can be expressed as:

$$SW(\tau) = \prod_{n=0}^{\tau} \frac{f(A_n | \bar{A}_{n-1})}{f(A_n | \bar{A}_{n-1}, \bar{L}_n, X)} \tag{11}$$

Noting that V is defined to be a subset of L_0 in [31]. Informally, they note the denominator to be conditional probability of a treatment assignment given past observations of treatment assignments and covariates and the numerator being that of treatment assignments alone, with the stabilized weights representing the incremental adjustment between the two.

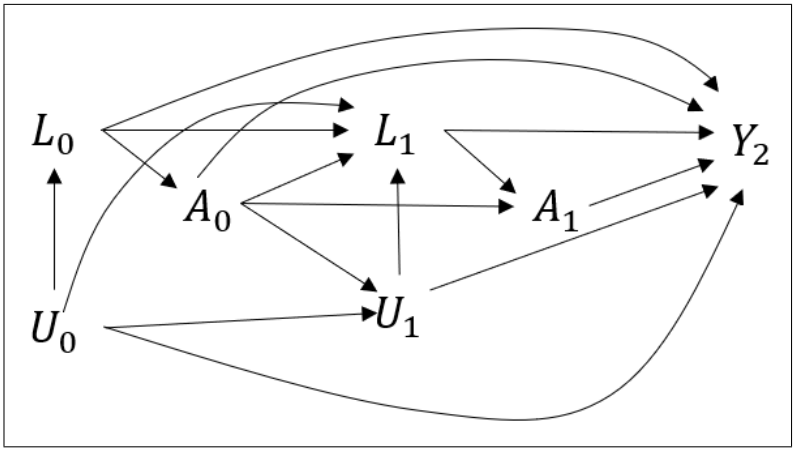


Figure 5: Causal Graph of Time-dependent Confounding for 2-step Study

In real clinical settings, it is often desirable to determine the treatment response in relation to the current follow up time, given past information. As such, we consider trajectories in relation to the last follow-up time t, retaining the form of the stabilized weights of the MSM and using all past observations, i.e.

$$SW(t,\tau) = \prod_{n=t}^{t+\tau} \frac{f(A_n|\bar{A}_{n-1})}{f(A_n|\bar{A}_{n-1},\bar{L}_n,X)} \tag{12}$$

C Additional Results for Experiments with Cancer Growth Simulation

Table 2 documents the full list of comparison for one-step-ahead predictions when tested for various γ, using different combinations of prediction and IPTW models.

$\gamma =$	**0**	**1**	**2**	**3**	**4**	**5**
Linear (No IPTWs)	5.55%	4.81%	4.09%	3.44%	2.86%	2.42%
MSM	4.84%	4.19%	3.56%	3.00%	2.51%	2.15%
MSM (LSTM IPTWs)	4.26%	3.68%	3.13%	2.64%	2.22%	1.95%
Seq2Seq (No IPTWs)	1.52%	1.39%	1.28%	1.23%	1.17%	1.17%
Seq2Seq (Logistic IPTWs)	4.34%	3.28%	2.42%	1.83%	1.43%	1.23%
R-MSN	0.92%	0.89%	0.85%	0.84%	0.79%	0.84%
BTRC	2.73%	2.59%	2.42%	2.28%	2.11%	2.07%
R-BTRC	1.91%	1.81%	1.72%	1.69%	1.63%	1.71%
$\gamma =$	**6**	**7**	**8**	**9**	**10**	
Linear (No IPTWs)	2.09%	1.80%	1.70%	1.65%	1.66%	
MSM	1.90%	1.68%	1.64%	1.64%	1.67%	
MSM (LSTM IPTWs)	1.77%	1.61%	1.62%	1.64%	1.69%	
Seq2Seq (No IPTWs)	1.13%	1.07%	1.07%	1.09%	1.08%	
Seq2Seq (Logistic IPTW)	1.12%	1.04%	1.04%	1.05%	1.04%	
R-MSN	0.88%	0.88%	0.94%	1.00%	1.02%	
BTRC	2.05%	2.01%	2.10%	2.16%	2.19%	
R-BTRC	1.77%	1.79%	1.93%	2.02%	2.09%	

Table 2: One-step-ahead Prediction Performance for Models Calibrated on $\gamma = 10$

D Marginal Structural Models for Cancer Simulation

The probabilities required for the IPTWs of the standard MSM in Section 5.2 can be described using logistic regression models with equations below:

$$f(A_t(k)|\bar{\mathbf{A}}_t) = \sigma\big(\omega_1^{(k)}\ (\sum_{n=0}^{t} \bar{A}_c(n-1)) + \omega_2^{(k)}\ (\sum_{n=0}^{t} \bar{A}_d(n-1))\big) \tag{13}$$

$$\begin{aligned} f(A_t(k)|\bar{\mathbf{H}}_t) = \sigma\big(\omega_5^{(k)}\ (\sum_{n=0}^{t} \bar{A}_c(n-1)) + \omega_6^{(k)}\ \sum_{n=0}^{t} (\bar{A}_d(n-1)) \\ +\omega_7^{(k)}\ V(t) + \omega_8^{(k)}\ V(t-1) + \omega_9^{(k)}\ S\big) \end{aligned} \tag{14}$$

$$f(C_t = 0|\mathcal{T} > n, \bar{\mathbf{A}}_{n-1}) = \sigma\big(\omega_{10}\ (\sum_{n=0}^{t} \bar{A}_c(n-1)) + \omega_{11}\ \sum_{n=0}^{t} (\bar{A}_d(n-1))\big) \tag{15}$$

$$\begin{aligned} f(C_t = 0|\mathcal{T} > n, \bar{\mathbf{L}}_{n-1}, \bar{\mathbf{A}}_{n-1}, \mathbf{X}) = \sigma\big(\omega_{12}\ (\sum_{n=0}^{t} \bar{A}_c(n-1)) + \omega_{13}\ (\sum_{n=0}^{t} \bar{A}_d(n-1)) \\ +\omega_{14}\ V(t-1) + \omega_{15}\ S\big) \end{aligned} \tag{16}$$

where $\sigma(.)$ the sigmoid function and ω_* are regression coefficients.

The regression model for prediction is given by:

$$\begin{aligned} g(\tau, a(t, \tau-1), \bar{\mathbf{H}}_t) = \beta_1\ (\sum_{n=0}^{t} \bar{A}_c(n-1)) + \beta_2\ (\sum_{n=0}^{t} \bar{A}_d(n-1)) \\ +\beta_3\ V(t) + \beta_4\ V(t-1) + \beta_5\ S \end{aligned} \tag{17}$$

E Algorithm Description for R-MSNs

To provide additional clarity on the relationship between the propensity networks and the Seq2Seq model, the pseudocode in Algorithm 1 describes the training process mentioned in Section 4.3.

We first define function $r_{(.)}\left(.; \boldsymbol{\theta}_{(.)}\right)$ to be RNN outputs given a vector of weights and hyperparameters $\boldsymbol{\theta}_{(.)}$. We refer the reader to Section 3 for more information on the functions in the MSM framework approximated by RNNs.

Propensity Networks Components of the propensity networks are used to compute the IPTWs $\mathbf{SW}(t, \tau)$ and $SW^*(t, \tau)$ as defined in Equations 2 and 3 respectively. The probabilities in the numerators and denominators are taken to be outputs of the propensity networks as below:

$$f(\mathbf{A}_n|\bar{\mathbf{A}}_{n-1}) = r_{A1}(\mathbf{A}_n|\bar{\mathbf{A}}_{n-1}; \boldsymbol{\theta}_{A1}) \tag{18}$$

$$f(\mathbf{A}_n|\bar{\mathbf{H}}_n) = r_{A2}(\mathbf{A}_n|\bar{\mathbf{H}}_n; \boldsymbol{\theta}_{A2}) \tag{19}$$

$$f(C_n = 0|\mathcal{T} > n, \bar{\mathbf{A}}_{\mathbf{n-1}}) = r_{C1}(\bar{\mathbf{A}}_{\mathbf{n-1}}; \boldsymbol{\theta}_{C1}) \tag{20}$$

$$f(C_n = 0|\mathcal{T} > n, \bar{\mathbf{L}}_{n-1}, \bar{\mathbf{A}}_{n-1}, \mathbf{X}) = r_{C1}(\bar{\mathbf{L}}_{n-1}, \bar{\mathbf{A}}_{n-1}, \mathbf{X}; \boldsymbol{\theta}_{C2}) \tag{21}$$

Encoder The encoder is also defined in a similar fashion below, with an additional function to output the internal states of the LSTM $\tilde{g}_{E1}(\bar{\mathbf{L}}_t, \bar{\mathbf{A}}_t, \mathbf{X}; \boldsymbol{\theta}_{E1})$. The encoder also computes the one-step-ahead predictions, i.e. $g(1, a(t, 0), \bar{\mathbf{H}}_t)$ as per Equation 1, which is to define the prediction error $e(i, t, 1)$ and encoder loss $\mathcal{L}_{encoder}$ – i.e.Equations 4 and 5 respectively.

$$g(1, a(t, 0), \bar{\mathbf{H}}_t) = r_E(\bar{\mathbf{L}}_t, \bar{\mathbf{A}}_t, \mathbf{X}; \boldsymbol{\theta}_E) \tag{22}$$

$$\mathbf{h}_t = \tilde{r}_E(\bar{\mathbf{L}}_t, \bar{\mathbf{A}}_t, \mathbf{X}; \boldsymbol{\theta}_E) \tag{23}$$

Decoder The decoder then uses the seq2seq architecture to project encoder states $\mathbf{h}_t$ forwards in time, incorporating planned future actions $\mathbf{a}_{t+\tau}$. This is also combined with the IPTWs to define the decoder loss $\mathcal{L}_{decoder}$ in Equation 5.

$$g(\tau, a(t, \tau-1), \bar{\mathbf{H}}_t) = r_D(\mathbf{h}_t, a_{t+1}, \ldots, a_{t+\tau}; \boldsymbol{\theta}_D), \forall \tau > 1 \tag{24}$$

Algorithm 1 Training Process for R-MSN

Input: Training/Validation Data $\bar{\mathbf{L}}_{1:T},\ \bar{\mathbf{A}}_{1:T},\ X$
Output: Neural network weights and hyperparameters for:
1) $\mathbf{SW}(t,\tau)$ networks: $\boldsymbol{\theta}_{A1},\ \boldsymbol{\theta}_{A2}$
2) $SW^*(t,\tau)$ networks: $\boldsymbol{\theta}_{C1},\ \boldsymbol{\theta}_{C2}$
3) Encoder network: $\boldsymbol{\theta}_{E1},\ \boldsymbol{\theta}_{E2}$
4) Decoder network: $\boldsymbol{\theta}_{D1},\ \boldsymbol{\theta}_{D2}$

Step 1: Fit Propensity Networks

$\boldsymbol{\theta}_{A1} \leftarrow \texttt{optimize}\Big(\sum_{n,i} \texttt{binary\_x\_entropy}(r_{A1}(\mathbf{A}_n(i)|\bar{\mathbf{A}}_{n-1}(i); \boldsymbol{\theta}_{A1}), \mathbf{A}_n(i)) \Big)$

$\boldsymbol{\theta}_{A2} \leftarrow \texttt{optimize}\Big(\sum_{n,i} \texttt{binary\_x\_entropy}(r_{A2}(\mathbf{A}_n(i)|\bar{\mathbf{H}}_n(i); \boldsymbol{\theta}_{A2}), \mathbf{A}_n(i)) \Big)$

$\boldsymbol{\theta}_{C1} \leftarrow \texttt{optimize}\Big(\sum_{n,i} \texttt{binary\_x\_entropy}(r_{C1}(\bar{\mathbf{A}}_{\mathbf{n-1}}(\mathbf{i}); \boldsymbol{\theta}_{C1}(i)), \mathbf{C}_n(i)) \Big)$

$\boldsymbol{\theta}_{C2} \leftarrow \texttt{optimize}\Big(\sum_{n,i} \texttt{binary\_x\_entropy}(r_{C2}(\bar{\mathbf{L}}_{n-1}(i), \bar{\mathbf{A}}_{n-1}(i), \mathbf{X}(i); \boldsymbol{\theta}_{C2}), \mathbf{C}_n(i)) \Big)$

Step 2: Generate IPTWs
for patient $i = 1$ to I **do**
  for $t = 1$ to T **do**
    for $\tau = 1$ to τ_{max} **do**
      $\mathbf{SW}_i(t,\tau) \leftarrow \prod_{n=t}^{t+\tau} r_{A1}(\mathbf{A}_n(i)|\bar{\mathbf{A}}_{n-1}(i); \boldsymbol{\theta}_{A1}) \ / \ r_{A2}(\mathbf{A}_n(i)|\bar{\mathbf{H}}_n(i); \boldsymbol{\theta}_{A2})$
      $SW_i^*(t,\tau) \leftarrow \prod_{n=t}^{t+\tau} r_{C1}(\bar{\mathbf{A}}_{\mathbf{n-1}}(\mathbf{i}); \boldsymbol{\theta}_{C1}(i)) \ / \ r_{C2}(\bar{\mathbf{L}}_{n-1}(i), \bar{\mathbf{A}}_{n-1}(i), \mathbf{X}(i); \boldsymbol{\theta}_{C2})$
    end for
  end for
end for

Step 3: Fit Encoder
$\boldsymbol{\theta}_E \leftarrow \texttt{optimize}\big(\mathcal{L}_{encoder}\big)$, as per Equation 5a

Step 4: Compute Encoder States {Used to Initialize Decoder}
for patient $i = 1$ to I **do**
  for $t = 1$ to T **do**
    $\mathbf{h}_t(i) \leftarrow \tilde{g}_E(\bar{\mathbf{L}}_t(i), \bar{\mathbf{A}}_t(i), \mathbf{X}(i); \boldsymbol{\theta}_E)$
  end for
end for

Step 5: Fit Decoder
$\boldsymbol{\theta}_D \leftarrow \texttt{optimize}\big(\mathcal{L}_{decoder}\big)$, as per Equation 5b

F Hyperparameter Optimization for R-MSN

For the R-MSN, 10,000 simulated paths were used for backpropagation of the network (training data), and 1,000 simulated paths for hyperparameter optimization (validation data) - with another 1,000 for out-of-sample testing. Given the differences in state initialization requirements and data batching of the decoder, we report the hyperparameter optimization settings separately for the decoder. The optimal parameters of all networks can be found in Table 5.

Settings for Propensity Networks and Encoder Hyperparameter optimization was performed using 50 iterations of random search, using the hyperparameter ranges in Table 3, and networks were trained using the ADAM optimizer [20]. For each set of sampled, simulation trajectories were grouped into B minibatches and networks were trained for a maximum of 100 epochs. LSTM state sizes were also defined in relation to the number of inputs for the network C.

Table 3: Hyperparameter Search Range for Propensity Networks and Encoder

	Hyperparameter Search Range
Hyperparameter Search Iterations	50
Dropout Rate	0.1 , 0.2 , 0.3, 0.4, 0.5
State Size	0.5C, 1C, 2C, 3C, 4C
Minibatch Size	64, 128, 256
Learning Rate	0.01, 0.005, 0.001
Max Gradient Norm	0.5, 1.0, 2.0

Settings for Decoder To train the decoder, the data was reformatted into sequences of $(\mathbf{h}_t, \{\mathbf{L}_{t+1}, \ldots, \mathbf{L}_{t+\tau_{max}}\}, \{\mathbf{A}_t, \ldots, \mathbf{A}_{t+\tau_{max}}, \mathbf{X}\})$, such that each patient i max T_i contributions to the training dataset. Given the T-fold increase in the number of rows in the overall dataset, we made a few modifications to the range of hyperparameter search, including increasing the size of minibatches and reducing the learning rate and number of iterations of hyperparameter search. The full range of hyperparameter search can be found in Table 4 and networks are trained for maximum of 100 epochs as well.

Table 4: Hyperparameter Search Range for Decoder

	Hyperparameter Search Range
Iterations of Hyperparameter Search	20
Dropout Rate	0.1 , 0.2 , 0.3, 0.4, 0.5
State Size	1C, 2C, 4C, 8C, 16C
Minibatch Size	256, 512, 1024
Learning Rate	0.01, 0.005, 0.001, 0.0001
Max Gradient Norm	0.5, 1.0, 2.0, 4.0

Table 5: Optimal Hyperparameters for R-MSN

	Dropout Rate	**State Size**	**Minibatch Size**	**Learning Rate**	**Max Norm**
Propensity Networks					
$f(\mathbf{A}_n \vert \bar{\mathbf{A}}_{n-1})$	0.1	6 (3C)	128	0.01	2.0
$f(\mathbf{A}_n \vert \tilde{H}_n)$	0.1	16 (4C)	64	0.01	1.0
$f(C_n = 0 \vert \mathcal{T} > n, \bar{\mathbf{A}}_{\mathbf{n-1}})$	0.2	4 (2C)	128	0.01	0.5
$f(C_t = 0 \vert \mathcal{T} > n, \bar{\mathbf{L}}_{n-1}, \bar{\mathbf{A}}_{n-1}, \mathbf{X})$	0.1	16 (4C)	64	0.01	2.0
Prediction Networks					
Encoder	0.1	16 (4C)	64	0.01	0.5
Decoder + Memory Adapter	0.1	16 (8C)	512	0.001	4.0

G Hyperparameter Optimization for BTRC

The parameters of the BTRC were optimized using the maximum-a-posteriori (MAP) estimation, using the same prior for global parameters and approach defined in [35]. While the model was replicated as faithfully to the specifications as possible, two slight modifications were made to adapt it to our problem. Firstly, the sparse GP approximations were avoided to ensure that we had as much accuracy as possible - using Gaussian Process with full covariance matrices for the random effects components. Secondly, as our dataset was partitioned to ensure that patient observed in the training set were not present in the test set, this means that any patient-specific parameters learned would not be used in the testing set itself. As such, to avoid optimizing on the test set, we adopt the standard approach for prediction in generalized linear mixed models [25], using the average population parameters, i.e. the global MAP estimate, for prediction.

Hyperparameter optimization was performed using grid search on the optimizer settings defined in 6, and was performed for a maximum of 5000 epochs per configuration. As convergence was observed to be slow for a number of settings, we also trained a reduced form of the full BTRC model without the "shared" parameters (indicated by '-' in Table7) to reduce the number of parameters of the model. The optimal global hyperparameters and optimizer settings can be found in Table 7.

Table 6: Hyperparameter Grid for BTRC

	Hyperparameter Search Range
Minibatch Size	2, 5, 10, 100, 500
Learning Rate	10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}

Table 7: MAP Estimates for BTRC

	BTRC	**R-BTRC**
$\bar{\chi}_{chemo}, \bar{\chi}_{radio}$	(-1.3729433, 0.065)	(-1.162, 0.007)
$\bar{\alpha}_{chemo}, \bar{\alpha}_{radio}$, $\bar{\alpha}^{(0)}_{chemo}, \bar{\alpha}^{(0)}_{radio}$	(0.760, 0.367), (0.207, 0.490)	(0.547, 0.367), (-, -)
$\bar{\beta}_{chemo}$,$\bar{\beta}_{radio}$, $\bar{\beta}^{(0)}_{chemo}, \bar{\beta}^{(0)}_{radio}$	(0.595, 0.367), (0.204, 0.368)	(0.429, 0.368), (-, -)
$\bar{\gamma}$	-0.27	-0.262
$\bar{\omega}$	-0.928	-
$\bar{l}^g$	1.223	-
$\bar{\kappa}$	0.786	0.867
$\bar{l}^v$	1.092	1.151
$\bar{\sigma}^2$	0.036	0.042
Learning Rate	0.001	0.001
Minibatch Size	100	100

Chapter 6

Extracting Useful Features From High Frequency Data

Publications Included

- **B. Lim**, S. Zohren, S. Roberts. *Detecting Changes in Asset Co-Movement Using the Autoencoder Reconstruction Ratio*. Risk, 2020. Weblink: `https://arxiv.org/abs/2002.02008`.

Detecting Changes in Asset Co-Movement Using the Autoencoder Reconstruction Ratio

Bryan Lim, Stefan Zohren, and Stephen Roberts

***Abstract*—Detecting changes in asset co-movements is of much importance to financial practitioners, with numerous risk management benefits arising from the timely detection of breakdowns in historical correlations. In this article, we propose a real-time indicator to detect temporary increases in asset co-movements, the Autoencoder Reconstruction Ratio, which measures how well a basket of asset returns can be modelled using a lower-dimensional set of latent variables. The ARR uses a deep sparse denoising autoencoder to perform the dimensionality reduction on the returns vector, which replaces the PCA approach of the standard Absorption Ratio (Kritzman *et al.*, 2011), and provides a better model for non-Gaussian returns. Through a systemic risk application on forecasting on the CRSP US Total Market Index, we show that lower ARR values coincide with higher volatility and larger drawdowns, indicating that increased asset co-movement does correspond with periods of market weakness. We also demonstrate that short-term (i.e. 5-min and 1-hour) predictors for realised volatility and market crashes can be improved by including additional ARR inputs.**

INTRODUCTION

The time-varying nature of asset correlations has long been of much interest to financial practitioners, with short-term increases in the co-movement of financial returns widely documented around periods of market stress (Preis *et al.*, 2012; Packham & Woebbeking, 2019; Campbell *et al.*, 2002; Cappiello *et al.*, 2006; Billio *et al.*, 2010). From a portfolio construction perspective, short-term increases in asset correlations have frequently been linked to spikes in market volatility (Campbell *et al.*, 2002) – with explanatory factors ranging from impactful news announcements (Aït-Sahalia & Xiu, 2016) to "risk-on risk-off" effects (Dungey *et al.*, 2018) – indicating that diversification can breakdown precisely when it is needed the most. In terms of systemic risk, increased co-movement between market returns is viewed as a sign of market fragility (Kritzman *et al.*, 2011), as the tight coupling between markets can deepen drawdowns when they occur, resulting in financial contagion. The development of methods to detect real-time changes in returns co-movement can thus lead to improvements in multiple risk management applications.

Common approaches for co-movement detection can be broadly divided into two categories. Firstly, multivariate Gaussian models have been proposed to directly model time-varying correlation dynamics – with temporal effects either modelled explicitly (Aït-Sahalia & Xiu, 2016; Dungey *et al.*, 2018), or incorporated simply by recalibrating covariance matrices daily using a sliding window of data (Preis *et al.*, 2012). More adaptive correlation estimates can also be provided by calibrating models using high-frequency data – as seen in the multivariate HEAVY models of Noureldin *et al.* (2012). However, several challenges remain with the modelling approach. For instance, naïve model specification can lead to spurious statistical effects (Loretan & English, 2000) reflecting correlation increases even when underlying model parameters remain unchanged. Furthermore, given that correlations are modelled in a pair-wise fashion between assets, condensing a high-dimensional correlation matrix into a single number measuring co-movement changes can be challenging in practice.

An alternative approach adopts the use of statistical factors models to capture common movements across the entire portfolio, typically projecting returns on low-dimensional set of latent factors using principal component analysis (PCA) (Kritzman *et al.*, 2011; Billio *et al.*, 2010; Zheng *et al.*, 2012). The popular Absorption Ratio (AR), for example, is defined by the fraction of the total variance of assets explained or absorbed by a finite set of eigenvectors (Kritzman *et al.*, 2011) – which bears similarity to metrics to evaluate eigenvalue significance in other domains (e.g. the Fractional Spectral Radius of Rezek & Roberts (1998)). Despite their ability to quantify co-movement changes with a single metric, with a larger AR corresponding to increased co-movement, PCA-based indicators require a covariance matrix to be estimated over a historical lookback window. This approach can be data-intensive for applications with many assets, as a long estimation window is required to ensure non-singularity

B. Lim, S. Zohren and S. Roberts are with the Department of Engineering Science and the Oxford-Man Institute of Quantitative Finance, University of Oxford, Oxford, United Kingdom (email:{blim, zohren, sjrob}@robots.ox.ac.uk).

of the covariance matrix (Billio *et al.*, 2010). Moreover, non-Gaussian financial returns (Jondeau *et al.*, 2007) can violate the normality assumptions required by PCA, potentially making classical PCA unsuitable for high-frequency data. While alternatives such as Independent Component Analysis (ICA) have been considered for modelling non-Gaussian financial time series data (Shah & Roberts, 2013), they typically maintain the assumption that assets are linear combinations of driving factors – and potential improvements can be made using non-linear approaches. As such, more adaptive non-parametric methods are hence needed to develop real-time indicators for changes in co-movement.

Advances in deep learning have demonstrated the benefits of autoencoders architectures for dimensionality reduction in complex datasets (Goodfellow *et al.*, 2016). In particular, autoencoders have had notable successes in feature extraction in images (Cho, 2013; Liu & Zhang, 2018; Freiman *et al.*, 2019), vastly out-performing traditional methods for dimensionality reduction in non-linear datasets. More recently, autoencoders have been explored as a replacement for PCA in various financial applications, encoding latent factors which account for non-linearities in return dynamics and allow for conditioning on exogenous covariates. Gu *et al.* (2019), for instance, use a series of encoders to estimate both latent factors and factor loadings (i.e. betas) used in conditional asset pricing models, demonstrating better out-of-sample pricing performance compared to traditional linear methods. Kondratyev (2018) explores the use of autoencoders to capture a low-dimensional representation of the term structure of commodity futures, and also shows superior reconstruction performance versus PCA. Given the strong performance enhancements of autoencoder architectures on non-linear datasets, replacing traditional factor models with autoencoders could be promising for co-movement measurement applications as well.

In this article, we introduce the Autoencoder Reconstruction Ratio (ARR) – a novel indicator to detect co-movement changes in real-time – based on the average reconstruction error obtained from applying autoencoders to high-frequency returns data. Adopting the factor modelling approach, the ARR uses deep sparse denoising autoencoders to project second-level intraday returns onto a lower-dimension set of latent variables, and aggregates reconstruction errors up to the desired frequency (e.g. daily). Increased co-movement hence corresponds to periods where reconstruction error is low, i.e. when returns are largely accounted for by the autoencoder's latent factors. In line with the canonical Absorption Ratio, we evaluate the performance of the ARR by considering an application in systemic risk – using increased co-movement of sector index returns to improve volatility and drawdown predictions for the total market. The information content of the ARR is evaluated by measuring the performance improvements of machine learning predictors when the ARR is included as a covariate. Based on experiments performed for 4 prediction horizons (i.e. 5-min, 1-hour, 1-day, 1-week), the ARR was observed to significantly improve performance for volatility and market crashes forecasts over short-term (5-min and 1-hour) horizons.

Problem Definition

For a portfolio of N assets, let $r(t,n)$ be the returns of the n-th asset at time t defined as below:

$$r(t,n) = \log p(t,n) - \log p(t-\Delta t, n), \tag{1}$$

where $p(t,n)$ is the value or price of asset n at time t, and Δt is a discrete interval corresponding to the desired sampling frequency.

In their most general form, statistical factor models map a common set of K latent variables to the returns of each asset as below:

$$\begin{aligned} r(t,n) &= \bar{r}(t,n) + \epsilon(t,n) &(2)\\ &= f_n\big(\boldsymbol{Z}(t)\big) + \epsilon(t,n), &(3)\end{aligned}$$

where the residual $\epsilon(t,n) \sim N\left(0, \xi_n^2\right)$ is the idiosyncratic risk of a given asset, and $f_n(.)$ is a generic function, and $\boldsymbol{Z}(t) = [z(t,1), \dots z(t,K)]^T$ is a vector of common latent factors.

Traditional factor models typically adopt a simple linear form, using asset-specific coefficients as below:

$$f_n\big(\boldsymbol{Z}(t)\big) = \boldsymbol{\beta}(n)^T \boldsymbol{Z}(t), \tag{4}$$

where $\boldsymbol{\beta}(n) = [\beta(n,1), \dots \beta(n,K)]^T$ is a vector of factor loadings. While various approaches are available for latent variable estimation – such as independent component analysis (ICA) (Fabozzi *et al.*, 2015) or PCA (Billio *et al.*, 2010) – the number of latent variables are typically kept low to reduce the dimensionality of the dataset (i.e. $K \ll N$).

Absorption Ratio

A popular approach to measuring asset co-movement is the Absorption Ratio (Kritzman *et al.*, 2011), which performs dimensionality reduction on returns using PCA. Co-movement changes are then measured based on the total variance absorbed by a finite set of eigenvectors. With K set to be a fifth of the number N of available

assets per Kritzman *et al.* (2011), the Absorption ratio is defined as:

$$\text{AR}(t) = \frac{\sum_{k=1}^{K} \sigma_{E_k}^2}{\sum_{n=1}^{N} \sigma_{A_n}^2}, \tag{5}$$

where $\text{AR}(t)$ is the Absorption Ratio at time t. $\sigma_{E_k}^2$ is the variance of the k-th largest eigenvector of the PCA decomposition, and $\sigma_{A_n}^2$ is the variance of the n-th asset.

Autoencoder Reconstruction Ratio

To detect changes in asset co-movements, we propose the Autoencoder Reconstruction Ratio below, which can be interpreted as the normalised reconstruction mean squared error (MSE) of high-frequency returns within a given time interval:

$$\text{ARR}(t, \Delta t) = \frac{\sum_{\tau=t-\Delta t}^{t} \sum_{n=1}^{N} \left(r(\tau, n) - \bar{r}(\tau, n)\right)^2}{\sum_{\tau=t-\Delta t}^{t} \sum_{n=1}^{N} r(\tau, n)^2}, \tag{6}$$

where Δt is the time interval matching the desired sampling frequency, and $\bar{r}(\tau, n)$ is the return of asset n reconstructed by a deep sparse denoising autoencoder (see sections below). For our experiments, we train the autoencoder using one-second returns and compute ARRs across four different sampling frequencies (i.e. 5-min, 1-hour, 1-day, 1-week).

Relationship to the Absorption Ratio

The ARR can also be interpreted as a slight reformulation of the standard Absorption Ratio, and we examine the relationship between the two in this section.

Reintroducing the linear Gaussian assumptions used by the Absorption Ratio, we note that the denominator of Equation (6) contains a simple estimator for the realised variance (RV) of each asset (Barndorff-Nielsen & Shephard, 2002), i.e.:

$$\text{RV}(t, \Delta t, n) = \sum_{\tau=t-\Delta t}^{t} r(t, n)^2 \tag{7}$$

$$\approx \sigma_{A_n}^2. \tag{8}$$

Moreover, by comparing to Equation (2), we can see that the numerator can be interpreted as an estimate of the sum of residual variances, leading to the form of the ARR below:

$$\text{ARR}(t, \Delta t) = \frac{\sum_{n=1}^{N} \xi_n^2}{\sum_{n=1}^{N} \sigma_{A_n}^2}. \tag{9}$$

Assuming that residuals are uncorrelated between assets, we note that the sum of residual variances essentially corresponds to the variance unexplained by the selected factors. The ARR can in this case be expressed as:

$$\text{ARR}(t, \Delta t) = \frac{\sum_{n=1}^{N} \sigma_{A_n}^2 - \sum_{k=1}^{K} \sigma_{E_k}^2}{\sum_{n=1}^{N} \sigma_{A_n}^2} \tag{10}$$

$$= 1 - \text{AR}(t) \tag{11}$$

Autoencoder Architecture

We adopt a deep sparse denoising autoencoder architecture (Goodfellow *et al.*, 2016) to compute reconstruction errors for the ARR. The network is predominantly divided into two parts – 1) a decoder which reconstructs the returns vector from a reduced set of latent factors, and 2) an encoder which performs the low-dimensional projection – both of which are described below.

Decoder:

$$\bar{\boldsymbol{r}}(t) = \boldsymbol{W}_1 \ \boldsymbol{h}_1(t) + \boldsymbol{b}_1 \tag{12}$$

$$\boldsymbol{h}_1(t) = \text{ELU}(\boldsymbol{W}_2 \ \boldsymbol{Z}(t) + \boldsymbol{b}_2) \tag{13}$$

where $\bar{\boldsymbol{r}}(t) = [\bar{r}(t, 1), \ldots, \bar{r}(t, N)]^T$ is the vector of reconstructed returns, ELU(.) is the exponential linear unit activation function (Clevert *et al.*, 2016), $\boldsymbol{h}_1(t) \in \mathbb{R}^H$ is the hidden state of the decoder network, and $\boldsymbol{W}_1 \in \mathbb{R}^{N \times H}, \boldsymbol{W}_2 \in \mathbb{R}^{H \times K}$, $\boldsymbol{b}_1 \in \mathbb{R}^N, \boldsymbol{b}_2 \in \mathbb{R}^H$ are its weights and biases.

Encoder:

$$\boldsymbol{Z}(t) = \text{ELU}\left(\boldsymbol{W}_3 \ \boldsymbol{h}_2(t) + \boldsymbol{b}_3\right) \tag{14}$$

$$\boldsymbol{h}_2(t) = \text{ELU}(\boldsymbol{W}_4 \ \boldsymbol{x}(t) + \boldsymbol{b}_4) \tag{15}$$

where $\boldsymbol{Z}(t) \in \mathbb{R}^K$ is a low-dimensional projection of inputs $\boldsymbol{x}(t) = [r(t, 1), \ldots, r(t, N), \mathcal{T}]^T$, $\boldsymbol{h}_2(t) \in \mathbb{R}^H$ is the hidden state of the encoder network, and $\boldsymbol{W}_3 \in \mathbb{R}^{K \times H}, \boldsymbol{W}_4 \in \mathbb{R}^{H \times N}$, $\boldsymbol{b}_3 \in \mathbb{R}^K, \boldsymbol{b}_4 \in \mathbb{R}^H$ are its weights and biases. We note that the time-of-day $\mathcal{T}$ – recorded as the number of seconds from midnight – is also included in the input along with sector returns, allowing the autoencoder to account for any intraday seasonality present in the dataset.

To ensure that dimensionality is gradually reduced, we set $K = \lfloor N/5 \rfloor$ as per the original Absorption Ratio paper, and fix $H = \lfloor (N + K)/2 \rfloor$.

Network Training

To improve generalisation on test data, sparse denoising autoencoders introduce varying degrees of regularisation.

Firstly, a sparsity penalty in the form of a L1 regularisation term is added to the reconstruction loss as below:

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{\sum_{t=1}^{T}\sum_{n=1}^{N}(r(t,n) - \bar{r(t,n)})^2}{TN} + \alpha\|\boldsymbol{Z}(t)\|_1, \quad (16)$$

where $\boldsymbol{\Theta}$ represents all network weights, α is a penalty weight which we treat as a hyperparameter, and $\|.\|_1$ is the L1 norm. Secondly, inputs are corrupted with noise during training – forcing the autoencoder to learn more general relationships from the data. Specifically, we adopt masking noise for our network, which simply corresponds to the application of dropout (Srivastava *et al.*, 2014) to encoder inputs.

Hyperparameter optimisation is performed using 20 iterations of random search, with networks trained up to a maximum of 100 epochs per search iteration. Full hyperparameter details are listed in the appendix for reference.

Forecasting Systemic Risk with the ARR

To demonstrate the utility of the ARR, we consider applications in systemic risk forecasting – using ARRs computed from subsector indices to predict risk metrics associated with the overall market. Specifically, we consider the two key use-cases for the ARR:

1) **Volatility Forecasting** – i.e. predicting market turbulence as measured by spikes in realised volatility;
2) **Predicting Market Crashes** – i.e. providing an early warning signal for sudden market declines, allowing for timely risk management.

Description of Dataset

We focus on the total US equity market and 11 constituent sector indices for our investigation, using high frequency total returns sampled every second to compute the ARR. Intraday index data from the Center of Research in Security Prices (CRSP) was downloaded via Wharton Research Data Services (WRDS (2019)) from 2012-12-07 to 2019-03-29, with the full list of indices provided in Exhibit 1.

Ticker	Index Description
CRSPTMT	CRSP US Total Market Total-Return Index
CRSPRET	CRSP US REIT Total-Return Index
CRSPENT	CRSP US Oil and Gas Total-Return Index
CRSPMTT	CRSP US Materials Total-Return Index
CRSPIDT	CRSP US Industrials Total-Return Index
CRSPCGT	CRSP US Consumer Goods Total-Return Index
CRSPHCT	CRSP US Health Care Total-Return Index
CRSPCST	CRSP US Consumer Services Total-Return Index
CRSPTET	CRSP US Telecom Total-Return Index
CRSPUTT	CRSP US Utilities Total-Return Index
CRSPFNT	CRSP US Financials Total-Return Index
CRSPITT	CRSP US Technology Total-Return Index

Exhibit 1: US Maket & Sector Indices Used in Tests

Reconstruction Performance

We compare the reconstruction accuracy of the deep sparse denoising autoencoder and standard PCA to evaluate the benefits of a non-linear approach to dimensionality reduction. To train the autoencoder, we divide the data into a training set used for network backpropagation (2012-2014), a validation set used for hyperparameter optimisation (2015 only), and a out-of-sample test set (2016-2019) used to quantify the information content of the ARR. PCA factors were calibrated using covariance matrix estimated with data from 2012-2016, keeping the same out-of-sample data as the autoencoder. The returns vector is projected onto 2 latent variables, with the dimensionality of the latent space taken to be $1/5$ the number of indices used – as per the original Absorption Ratio paper (Kritzman *et al.*, 2011).

We evaluate the reconstruction accuracy using the test dataset, based on the R-squared (R^2) of reconstructed returns for each prediction model. Given that the ARR is computed based on sector data alone, we only focus on the reconstruction accuracy of the 11 sector indices. We also test for the significance of the results with a bootstrap hypothesis test – using 500 bootstrap samples and adopting the null hypothesis that there is no difference in R^2 between the autoencoder and PCA reconstruction.

	PCA	Autoencoder	P-Value
$\boldsymbol{R^2}$	0.340	**0.461***	< 0.01

Exhibit 2: Out-of-Sample Reconstruction R^2 (2016-2019)

From the results in Exhibit 2, we can see that the autoencoder greatly enhances reconstruction accuracy – increasing the out-of-sample R^2 by more than 35% and statistically significant with 99% confidence. These improvements highlight the benefits of adopting a non-linear approach to dimensionality reduction, and the suitability of the autoencoder for modelling intraday returns. Such an approach can have wider applications in various areas of quantitative finance.

Empirical Analysis

Next, we perform an exploratory investigation into the relationships between the total market index, and the

ARR of its constituent sectors. Specifically, we examine the following metrics aggregated over different sampling frequencies ($\Delta t \in \{5\text{-min}, 1\text{-hour}, 1\text{-day}, 1\text{-week}\}$):

- **Returns** – Corresponding to the difference in log prices based on Equation (1).
- **Log Realised Volatility (Log RV)** – Computed based on the logarithm of the simple realised volatility estimator of Equation (7).
- **Drawdowns (DD)** – Determined by the factional decrease from the maximum index value from the start of our evaluation period.

ARRs were similarly computed for the 4 sampling frequencies, and aggregated based on Equation (6). Using the KDE plots of Exhibits 3 to 5, we perform an in-sample (2012-2015) empirical analysis of the coincident relationships between the ARR and the metrics above. To avoid spurious effect, we remove the eves of public holidays from our analysis – where markets are open only in the morning and trading volumes are abnormally low. In addition, we winsorise the data at the 1^{st} and 99^{th} percentiles, reducing the impact of outliers to improve data visualisation.

Looking at the KDE plots for returns against the ARR in Exhibit 3, we observe a noticeable increase in the dispersion of returns appears at low ARR values. This is echoed by the KDE plots for Log RV in 4 – which appears to increase linearly with decreasing ARR. Drawdowns behave in a similar fashion as well, with larger drawdowns appearing to occur at low values of the ARR. Effects also are consistent across different horizons, with similar patterns observed for all sampling frequencies. On the whole, the results validate the findings observed in previous works – indicating that increased asset co-movements do indeed coincide with periods of market weakness – with lower ARR values observed around periods of high volatility or high drawdowns.

To visualise how the ARR changes over time, we plot 5-min ARR (i.e. ARR$(t, 5\text{-min})$) against the prices and drawdowns of the CRSP Total Market Index in Exhibit 6. Given the noisiness of the raw ARR values, we also included a smoothed version of 5-min ARR in the bottom – computed based on an exponentially weighted moving average with a 1-day half-life. We can see from the results that dips in the ARR occur slightly before large drawdown periods, particularly around the sell-offs of August 2015 and early 2018. As such, the ARR could potentially be used to improve predictions of volatility spikes or sudden market crashes in the near future – which we further investigate in the next section.

QUANTIFYING THE INFORMATION CONTENT OF THE ARR

We attempt to quantify the information content of the ARR by observing how much it improves forecasting models for the risk metrics above. To do so, we adopt several machine learning benchmarks for risk prediction, and evaluate the models' forecasting performance both with and without the ARR included in its inputs. This allows us to evaluate how much additional information is provided by the ARR, above that provided by temporal evolution of realised volatility or drawdowns alone.

Benchmark Models

Given the non-linear relationships observed between returns and drawdowns versus the ARR, we adopt a series of machine learning benchmarks on top of linear models. Specifically, we consider 1) linear/logistic regression, 2) gradient boosted decision trees (GBDTs), and 3) simple multi-layered perceptrons (MLPs) for our forecasting applications. Hyperparameter optimisation is performed using up to 200 iterations of random search, with optimal hyperparameters selected using 3-fold cross validation. Additional training details can also be found in the appendix.

Volatility Forecasting Methodology

We treat volatility forecasting as a regression problem, focusing on predicting Log RV over different horizons – using a variant of the HAR-RV model of Corsi (2009). For 5-min Log RV, a simple HAR-RV model can take the form below, incorporating log RV terms for longer horizons into the forecast:

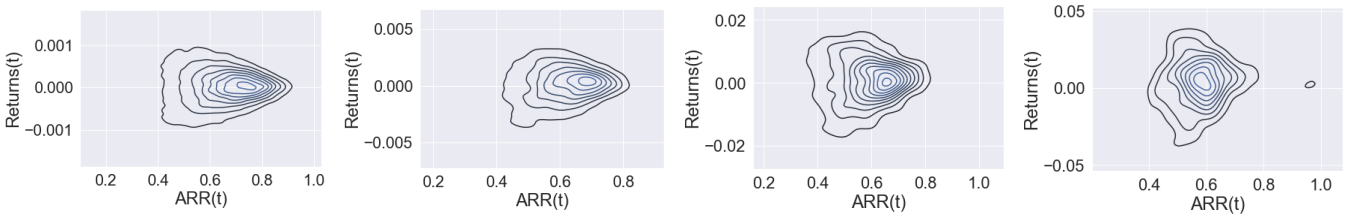
$$\begin{aligned}&\bar{\nu}(t + 5\text{-min}, 5\text{-min})\\ &= \beta_1\ \nu(t, 5\text{-min}) + \beta_2\ \nu(t, 1\text{-hour})\\ &+ \beta_3\ \nu(t, 1\text{-day}) + \beta_4\ \nu(t, 1\text{-week}),\end{aligned} \tag{17}$$

where $\bar{\nu}(t + 5\text{-min}, 5\text{-min})$ is 5-min log RV predicted over the next time-step, $\nu(t, \Delta t)$ is the log RV from $t - \Delta t$ to t, and $\beta_{(.)}$ are linear coefficients.

We adopt a similar non-linear variant for our forecast for each prediction horizon δ. For tests with the ARR, this takes the form:

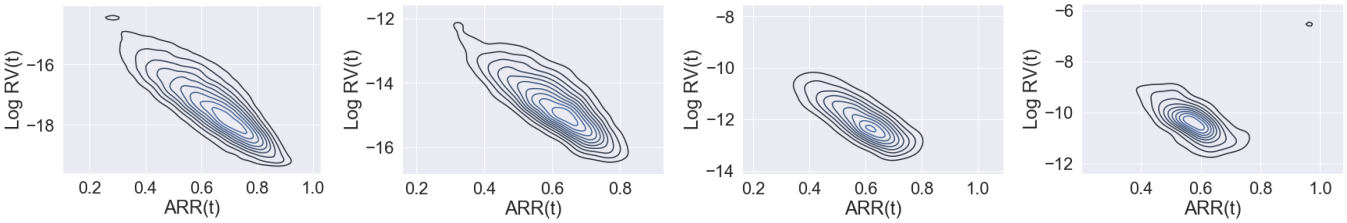
$$\bar{\nu}(t + \delta, \delta) = g_1(\psi(t), \omega(t)), \tag{18}$$

where $\delta \in \Psi$, $\Psi = \{5\text{-min}, 1\text{-hour}, 1\text{-day}, 1\text{-week}\}$, $\psi(t) = \{\nu(t, d) : d \in \Psi \text{ and } d >= \delta\}$, $\omega(t) = \{\text{ARR}(t, d) : d \in \Psi \text{ and } d >= \delta\}$, and $g_2(.)$ is a prediction model mapping inputs to returns. The combination of both realised volatility and ARR values hence allows



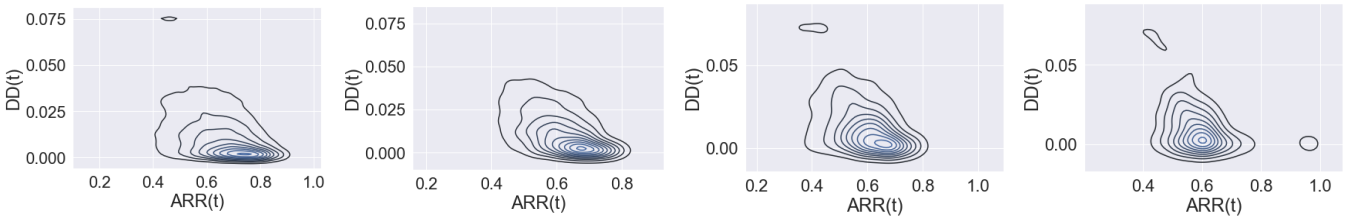
(a) $\Delta t = 5$-min (b) $\Delta t = 1$ hour (c) $\Delta t = 1$-day (d) $\Delta t = 1$-week

Exhibit 3: In-sample KDE Plots for Returns vs. ARR.



(a) $\Delta t = 5$-min (b) $\Delta t = 1$ hour (c) $\Delta t = 1$-day (d) $\Delta t = 1$-week

Exhibit 4: In-Sample KDE Plots for Log RV vs. ARR.



(a) $\Delta t = 5$-min (b) $\Delta t = 1$ hour (c) $\Delta t = 1$-day (d) $\Delta t = 1$-week

Exhibit 5: In-Sample KDE Plots for Drawdowns vs. ARR.

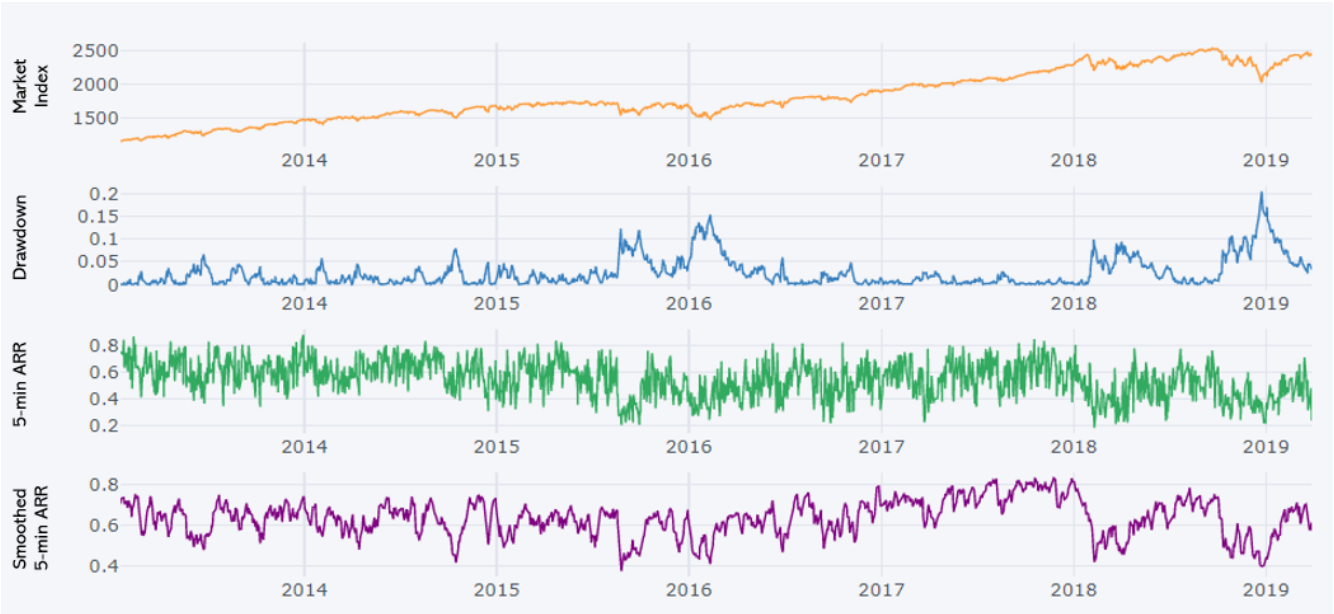


Exhibit 6: 5-min ARR from 2012 to 2019.

us to determine if the ARR supplies any information above that provide the volatility time series alone.

For tests without the ARR, we continue to use Equation (18) but omit ARR values, i.e. $\omega(t) = \emptyset$. We evaluate regression performance using the R^2 of forecasted log RV.

Crash Prediction Methodology

We take a binary classification approach to forecasting market crashes, using our benchmark models to predict the onset of a sharp drawdown. First, we define a z-score metric for returns $\gamma(t, \lambda)$ as below:

$$\gamma(t, \lambda) = \frac{r(t) - m(t, \lambda)}{s(t, \lambda)}, \tag{19}$$

where $r(t)$ is the return for the total market index, $m(t, \lambda)$ is the exponentially weighted moving average for returns with a half-life of λ, and $s(t, \lambda)$ its exponentially weighted moving standard deviation.

Next, based on our z-score metric, we define a market crash to be a sharp decline in market returns, i.e.:

$$c(t) = \mathbb{I}(\gamma(t, \lambda) < C), \tag{20}$$

where $c(t) \in 1, 0$ is a crash indicator, $\mathbb{I}(.)$ is an indicator function, and C is the z-score threshold for returns. For our experiments, we set λ to be 10 discrete time steps and $C = -1.5$.

We then model crash probabilities using a similar form to Equation (18):

$$p(c(t) = 1) = g_2(\psi(t), \omega(t)), \tag{21}$$

where $g_2(.)$ is a function mapping inputs to crash probabilities $p(c(t) = 1)$.

Given that crashes are rare by definition – with $c(t) = 1$ for less than 10% of time steps for daily frequencies – we also oversample the minority class to address the class imbalance problem. Classification performance is evaluated using the area under the receiver operating characteristic (AUROC).

Results and Discussion

The results for both volatility forecasting and market crash prediction can be found in Exhibit 7, both including and excluding ARR values. To determine the statistical significance of improvements, we conduct a bootstrap hypothesis test under the null hypothesis that performance results are better when the ARR is included, using a non-parametric bootstrap with 500 samples.

From the volatility forecasting R^2 values in Exhibit 7a, we can see that the ARR consistently improves 5-min and 1-hour forecasts for all non-linear models, and significant improvements are observed for linear models for 5-min sampling frequencies. This indicates that the ARR is informative for short-term forecasts, and can help enhance risk predictions in the near-term. For longer horizons however (i.e. 1-day and 1-week) we observe that the inclusion of the ARR reduces prediction accuracy – potentially indicating the presence of overfitting on the training set when ARRs are introduced.

For crash predictions AUROC results in Exhibit 7b, we note that the ARRs are observed to improve forecasts for all models and sampling frequencies – with statistical significance at the 99% level observed for both linear and GBDT forecasts over shorter horizons. This echoes the volatility forecasting results – indicating that ARRs can be useful to inform short-term risk predictions.

CONCLUSIONS

We introduce the Autoencoder Reconstruction Ratio (ARR) in this paper, using it as a real-time measure of asset co-movement. The ARR is based on the normalised reconstruction error of a deep sparse denoising autoencoder applied to a basket of asset returns – which condenses the returns vector onto a lower dimensional set of latent variables. This replaces the PCA modelling approach used by the Absorption Ratio of Kritzman *et al.* (2011), which allows the ARR to better model returns that violate basic PCA assumptions (e.g. non-Gaussian returns). Through experiments on a basket of 11 CRSP US sector indices, we demonstrate that the autoencoder significantly improves the out-of-sample reconstruction performance when compared to PCA, increasing the combined R^2 by more than 35%.

Given the links identified between increased asset co-movements and the fragility of the overall market in previous works (Kritzman *et al.*, 2011; Campbell *et al.*, 2002), we also evaluate the use of the ARR in a systemic risk application. First, we conduct an empirical analysis of the relationship between risk metrics of the combined market (using the CRSP US Total Market Index as a proxy) and the ARR computed from its sub sectors of the market. Based on an analysis of the KDE plots of risk metrics vs. ARRs, we show that low values of the ARR coincide with high volatility and high drawdown periods in line with previous findings. Next, we evaluate the information content of the ARR by testing how much the ARR improves risk predictions for a various benchmark models. We find that the ARR is informative for both volatility and market crash predictions over short horizons, and significantly increases 5-min and 1-hour forecasting performance across most model benchmarks.

		5-min	1-hour	1-day	1-week
Linear	With ARR	**0.635***	0.496	0.389	0.579
	No ARR	0.626	**0.536***	**0.431***	**0.611***
	P-Values	<0.01	>0.99	>0.99	>0.99
GBDT	With ARR	**0.635***	**0.554***	0.425	0.399
	No ARR	0.627	0.536	**0.434**	**0.593***
	P-Values	<0.01	<0.01	0.821	>0.99
MLP	With ARR	**0.641***	**0.571***	0.387	0.527
	No ARR	0.631	0.549	**0.426***	**0.636***
	P-Values	<0.01	<0.01	>0.99	>0.99

(a) R^2 for Log RV Predictions

		5-min	1-hour	1-day	1-week
Linear	With ARR	**0.598***	**0.629***	**0.585**	**0.372**
	No ARR	0.587	0.570	0.582	0.333
	P-Values	<0.01	<0.01	0.464	0.284
GBDT	With ARR	**0.590***	**0.562***	**0.529**	**0.605**
	No ARR	0.567	0.516	0.465	0.575
	P-Values	<0.01	0.01	0.138	0.359
MLP	With ARR	**0.589**	**0.564**	**0.586**	**0.423**
	No ARR	0.588	0.558	0.540	0.210
	P-Values	0.374	0.392	0.234	<0.01

(b) AUROC for Crash Predictions

Exhibit 7: Out-of-Sample Forecasting Results With and Without ARR Inputs.

REFERENCES

Abadi, Martín, *et al.* 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Aït-Sahalia, Yacine, & Xiu, Dacheng. 2016. "Increased correlation among asset classes: Are volatility or jumps to blame, or both?,". *Journal of Econometrics*, **194**(2), 205–219.

Barndorff-Nielsen, Ole E., & Shephard, Neil. 2002. "Estimating quadratic variation using realized variance,". *Journal of Applied Econometrics*, **17**(5), 457–477.

Billio, Monica, Getmansky, Mila, Lo, Andrew W., & Pelizzon, Loriana. 2010. "Measuring systemic risk in the finance and insurance sectors,". *MIT Sloan Research Paper No. 4774-10*. http://ssrn.com/abstract=1571277.

Campbell, Rachel, Koedijk, Kees, & Kofman, Paul. 2002. "Increased correlation in bear markets,". *Financial Analysts Journal*, **58**(1), 87–94.

Cappiello, Lorenzo, Engle, Robert F., & Sheppard, Kevin. 2006. "Asymmetric Dynamics in the Correlations of Global Equity and Bond Returns,". *Journal of Financial Econometrics*, **4**(4), 537–572.

Cho, Kyung Hyun. 2013. "Simple sparsification improves sparse denoising autoencoders in denoising highy noisy images,". *In: Proceedings of the 30th International Conference on Machine Learning*. ICML 2013.

Clevert, Djork-Arne, Unterthiner, Thomas, & Hochreiter, Sepp. 2016. "Fast and accurate deep network learning by exponential linear units (ELUs),". *In: International Conference on Learning Representations*. ICLR 2016.

Corsi, Fulvio. 2009. "A Simple Approximate Long-Memory Model of Realized Volatility,". *Journal of Financial Econometrics*, **7**(2), 174–196.

Dungey, Mardi, Erdemlioglu, Deniz, Matei, Marius, & Yang, Xiye. 2018. "Testing for mutually exciting jumps and financial flights in high frequency data,". *Journal of Econometrics*, **202**(1), 18 – 44.

Fabozzi, Frank, Giacometti, Rosella, & Tsuchida, Naoshi. 2015. *The ICA-based Factor Decomposition of the Eurozone Sovereign CDS Spreads*. IMES Discussion Paper Series 15-E-04. Institute for Monetary and Economic Studies, Bank of Japan.

Freiman, Moti, Manjeshwar, Ravindra, & Goshen, Liran. 2019. "Unsupervised abnormality detection through mixed structure regularization (MSR) in deep sparse autoencoders,". *Medical Physics*, **46**(5), 2223–2231.

Goodfellow, Ian, Bengio, Yoshua, & Courville, Aaron. 2016. "Autoencoders,". *Chap. 14 of: Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gu, Shihao, Kelly, Bryan T., & Xiu, Dacheng. 2019. "Autoencoder asset pricing models,". *Yale ICF Working Paper No. 2019-04; Chicago Booth Research Paper No. 19-24*. https://ssrn.com/abstract=3335536.

Jondeau, Eric, Poon, Ser-Huang, & Rockinger, Michael. 2007. *Financial modeling under non-gaussian distributions*. Springer Finance. Springer.

Ke, Guolin, *et al.* 2017. "Lightgbm: A highly efficient gradient boosting decision tree,". *Page 3149–3157 of: Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc.

Kondratyev, Alexei. 2018. "Learning curve dynamics with artificial neural networks,". *SSRN*. https://ssrn.com/abstract=3041232.

Kritzman, Mark, Li, Yuanzhen, Page, Sébastien, & Rigobon, Roberto. 2011. "Principal components as a measure of systemic risk,". *The Journal of Portfolio Management*, **37**(4), 112–126.

Liu, Yan, & Zhang, Yi. 2018. "Low-dose CT restoration via stacked sparse denoising autoencoders,". *Neurocomputing*, **284**, 80 – 89.

Loretan, Mico, & English, William B. 2000. "Evaluating correlation breakdowns during periods of market volatility,". *Board of Governors of the Federal Reserve System International Finance Working Paper*.

Noureldin, Diaa, Shephard, Neil, & Sheppard, Kevin. 2012. "Multivariate high-frequency-based volatility (heavy) models,". *Journal of Applied Econometrics*, **27**(6), 907–933.

Packham, N., & Woebbeking, C.F. 2019. "A factor-model approach for correlation scenarios and correlation stress testing,". *Journal of Banking and Finance*, **101**, 92 – 103.

Pedregosa, F., *et al.* 2011. "Scikit-learn: Machine Learning in Python ,". *Journal of Machine Learning Research*, **12**, 2825–2830.

Preis, T, Kenett, DY, Stanley, HE, Helbing, D, & Ben-Jacob, E. 2012. "Quantifying the behavior of stock correlations under market stress,". *Scientific Reports*, **752**(2).

Rezek, I. A., & Roberts, S. J. 1998. "Stochastic complexity measures for physiological signal analysis,". *IEEE Transactions on Biomedical Engineering*, **45**(9), 1186–1191.

Shah, Nauman, & Roberts, Stephen. 2013. "Dynamically measuring statistical dependencies in multivariate financial time series using independent component analysis,". *International Scholarly Research Notices*.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, & Salakhutdinov, Ruslan. 2014. "Dropout: A simple way to prevent neural networks from overfitting,". *Journal of Machine Learning Research*, **15**, 1929–1958.

WRDS. 2019. *The Center for Research in Security Prices (CRSP)*

Index History - Intraday. https://wrds-www.wharton.upenn.edu/.
Zheng, Zeyu, Podobnik, Boris, Feng, Ling, & Li, Baowen. 2012. "Changes in cross-correlations as an indicator for systemic risk,". *Scientific Reports*, **888**(2).

APPENDIX

A. Additional Training Details

Python Libraries: Deep sparse denoising autoencoders are defined and trained using the `TensorFlow` (Abadi *et al.*, 2015). For Gradient Boosted Decision Trees, we use the `LightGBM` library (Ke *et al.*, 2017) – using the standard `LightGBMRegressor` and `LightGBMClassifier` depending on the forecasting problem. The remainder of the models are implemented using standard `scikit-learn` classes (Pedregosa *et al.*, 2011) – with classes described in the hyperparameter optimisation section.

Hyperparameter Optimisation Details: Random search is conducted by sampling over a discrete set of values for each hyperparameter, which are listed below for each hyperparameter. For ease of reference, hyperparameters for all `scikit-learn` and `LightGBM` classes are referred to by the default argument names used in their respective libraries.

Deep Sparse Denoising Autoencoder

- Dropout Rate – [0.0, 0.2, 0.4, 0.6, 0.8]
- Regularisation Weight α – [0.0, 0.01, 0.1, 1.0, 10]
- Minibatch Size – [256, 512, 1024, 2048]
- Learning Rate – [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0]
- Max. Gradient Norm – [10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0, 10.0]

Linear Regression

- Package Name – `sklearn.linear_model`
- Class Name – `LogisticRegression`
- 'alpha' – [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2],
- 'fit_intercept' – [False, True],

Logistic Regression

- Package Name – `sklearn.linear_model`
- Class Name – `LogisticRegression`
- 'penalty' – ['l1']
- 'C' – [0.01, 0.1, 1.0, 10, 100]
- 'fit_intercept' – [False]
- 'solver' –['liblinear']

Gradient Boosted Decision Tree

- Package Name – `lightgbm`
- Class Name – `LGBMRegressor` or `LGBMClassifier`
- 'learning_rate' – [10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
- 'n_estimators' – [5, 10, 20 ,40, 80, 160, 320]
- 'num_leaves' – [5, 10, 20, 40, 80]
- 'n_jobs' – [5]
- 'reg_alpha' – [0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
- 'reg_beta' – [0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
- 'boosting_type' – ['gbdt']

Multi-layer Perceptron

- Package Name – `sklearn.neural_network`
- Class Name – `MLPRegressor` or `MLPClassifier`
- 'hidden_layer_sizes' – [5, 10, 20, 40, 80, 160]
- 'activation' – ['relu']
- 'alpha' – [0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2]
- 'learning_rate_init' – [10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
- 'early_stopping' – [True],
- 'max_iter' – [500]

Chapter 7

Discussion & Conclusions

In this thesis, we introduce a collection of novel methods to improve the performance of deep learning models for time series forecasting, and propose extensions to facilitate decision making over time. While model design can vary from application to application – due to the heterogeneity of forecasting problems in different domains – our central theme is in the closer alignment of models with traditional time series models. This allows us to customise existing building blocks in deep learning to better capture the nuances of complex temporal processes, leading to demonstrable improvements in tests on a wide variety of datasets.

Contributions in Time Series Forecasting: With the growing size and complexity of time series datasets, data-driven deep learning approaches have increasingly outperformed parametric modelling techniques. In many forecasting applications, however, neural network architectures are often taken directly from other domains (e.g. LSTMs and CNNs) and applied with modification, neglecting to account for the unique characteristics of time series datasets. We hence aim to improve deep learning models by drawing inspiration from traditional time series modelling – designing networks with the appropriate inductive biases and outputs for time series data.

For general time series problems, we propose two new architecture for one-step-ahead and multi-horizon forecasting in Chapter 3. With the Recurrent Neural Filter (RNF), we propose a new architecture for one-step-ahead prediction, comprising a series of encoders and decoders that are aligned with the Bayesian filtering steps. We also show that each stage of the RNF can be separated at run-time, due to the skip training approach and multi-task loss function used to encourage decoupled representations. This allows the RNF to be applied using data only when available – helping the RNF to handle datasets with missing data, and to improve performance when applied in an autoregressive fashion for multi-horizon prediction. To better

accommodate the full range of inputs present in complex multi-horizon forecasting scenarios, such as static metadata, we also developed a new attention-based model in Temporal Fusion Transformers (TFT). From a forecasting perspective, the TFT uses both recurrent and attention layers for local and long-term temporal processing, and contains a series of gated residual networks to allow the network to suppress unnecessary components for a given dataset. This allows the TFT to perform well on a wide range of data regimes, outperforming state-of-the-art methods for multi-horizon prediction.

To further incorporate domain expertise for specific problems, we examine a range of hybrid deep learning models in Chapter 4 – focusing on an application in finance and another in medicine. Hybrid models enhance traditional methods using deep learning by parameterising well-studied quantitative models for a given domain with neural network outputs. With Deep Momentum Networks (DMNs), we enhance standard time series momentum signals using deep learning-based trading rules, demonstrating improved performance over traditional and machine learning-based systematic signals. In addition, we augment joint models for longitudinal and time-to-event data in biostatistics with the the Disease-Atlas, demonstrating improvements in predicting multiple clinical outcomes in Cystic Fibrosis patients.

Finally, we analyse the use of deep neural networks as feature extraction mechanisms in Chapter 6, feeding features produced by autoencoders to improve the accuracy of generic forecasting models. Considering a financial use-case in systemic risk prediction, we develop the Autoencoder Reconstruction Ratio (ARR) – which is based on a measure of the reconstruction error for high-frequency returns. The ARR computes the amount of information captured by a low dimensional set of non-linear latent variables, allowing us to quantify any changes in asset co-movement over time. Through tests on index data, we demonstrate that the autoencoder approach provides a better model for high-frequency returns, and that the ARR can be used to enhance short-term risk predictions.

Contributions in Decision Support Over Time: While model users do rely on forecasts as one source of information, they may desire a deeper understanding of their dataset to guide their actions, e.g. through counterfactual simulations or the identification of key input features driving predictions. However, deep learning models can suffer several limitations in this respect. Firstly, deep neural networks are designed to capture complex correlations, without distinguishing between cause and effect or accounting for confounding factors. As a result, slight shifts in the

input distribution can severely affect model performance even if causal mechanisms remain unchanged – limiting the use of standard models for counterfactual predictions [13]. Furthermore, the black-box nature of standard deep neural networks can make it difficult to interpret the relationships learnt by the model, or to identify which features are important for forecasts.

As such, on top of innovations in time series forecasting, we also explore two extensions to deep neural networks to better facilitate decision making over time. We start by presenting Recurrent Marginal Structural Networks in Chapter 5, which provides a framework to training deep neural networks to learn causal effects from observational data. Based on the inverse probability of treatment weighting approach of marginal structural models in epidemiology, RMSNs use deep neural networks to learn probabilities of treatment assignment and censoring. The probabilities are then used to generate weights to loss functions for a prediction network, adjusting neural network training to account for time-dependent confounding effects. Through tests with a clinically realistic simulation model, we show that RMSNs outperform state-of-the-art benchmarks in learning unbiased treatment effects. Moreover, with the TFT, we demonstrate how attention weights can be analysed to provide general insights into the temporal relationships present in the dataset. This is showcased through three interpretability use-cases – 1) analysing feature importance, 2) visualising persistent temporal patterns, and 3) identifying significant regimes and events.

7.1 Extensions and Future Work

While large strides have been made with the field of deep learning for time series prediction, the greatest impact has been on univariate time series sampled at discrete intervals – which are most suited for current neural network designs. In this section, we propose two directions to extend our research to handle more complex time series forecasting problems.

Continuous-Time Models: As universal function approximators [9], neural networks have traditionally been used to model fixed input-output relationships – making them inherently suited to handle discrete data. However, forecasting in continuous-time can be a more suitable approach in many applications, particularly in the case of datasets with irregularly sampled observations and high-frequency streaming data. A recent promising direction in deep learning research has been in Neural Ordinary Differential Equation (ODE) and Stochastic Differential Equation (SDE) models

[3, 20, 12, 10, 5] – which model the deterministic or stochastic evolution of hidden states, and naturally handle continuous time forecasts. However, improvements have mainly been demonstrated using simulated data, which make it difficult to assess the performance of Neural ODEs/SDEs on real-world time series datasets. In addition to benchmarking and extensions to state-of-the-art time series architectures, we also recommend investigations into hybrid continuous-time models – which would allow for direct comparisons to traditional methods in applications where they are most beneficial.

Multivariate Time Series: The majority of previous work (see Section 2) has focused on the development of univariate forecasting models – i.e. assuming that targets are driven only by inputs for a given entity, and are independent of each other. In this manner, deep neural networks are trained to capture temporal relationships that are generally applicable across all entities, without taking into account cross-sectional relationships between them. While this allows networks to be trained with a larger pool of data – as mini-batches are sampled across entities and time – there are instances where multivariate forecast can be beneficial, particularly in non-stationary datasets. For instance, in portfolio management, market breakdowns can cause a general decline of all stocks and result in short-time increases in asset correlations. In retail forecasting, unforeseen events, such as natural disasters, can also lead to spikes in the joint demand for specific categories of goods. As such, multivariate models can help improve performance in datasets where common driving factors exist across entities – motivating the need for extensions to existing univariate architectures.

Bibliography

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *arXiv e-prints*, page arXiv:1206.5538, Jun 2012.

[2] James V. Candy. *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Wiley-Interscience, New York, NY, USA, 2009.

[3] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[4] F.X. Diebold. *Time-Series Econometrics*. Department of Economics, University of Pennsylvania, 2018.

[5] Lea Duncker, Gergo Bohner, Julien Boussard, and Maneesh Sahani. Learning interpretable continuous-time models of latent stochastic dynamical systems. In *International Conference on Machine Learning (ICML)*, 2019.

[6] Chenyou Fan, Yuze Zhang, Yi Pan, Xiaoyue Li, Chi Zhang, Rong Yuan, Di Wu, Wensheng Wang, Jian Pei, and Heng Huang. Multi-horizon time series forecasting with temporal attention learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, 2019.

[7] Gonzalo Farias, Sebastián Dormido-Canto, Jesús Vega, Giuseppe Rattá, Héctor Vargas, Gabriel Hermosilla, Luis Alfaro, and Agustín Valencia. Automatic feature extraction in large fusion databases by using deep learning approach. *Fusion Engineering and Design*, 112:979 – 983, 2016.

[8] Shihao Gu, Bryan T. Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *Chicago Booth Research Paper No. 18-04; 31st Australasian Finance and Banking Conference 2018*, 2017.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[10] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[11] J. F. Kolen and S. C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long Term Dependencies*. IEEE, 2001.

[12] Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David K. Duvenaud. Scalable gradients and variational inference for stochastic differential equations. In *Proceedings of The 2nd Symposium on Advances in Approximate Bayesian Inference*, 2020.

[13] Bryan Lim, Ahmed Alaa, and Mihaela van der Schaar. Forecasting treatment responses over time using Recurrent Marginal Structural Networks. In *NeurIPS*, 2018.

[14] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *arXiv e-prints*, page arXiv:1912.09363, 2019.

[15] Bryan Lim and Mihaela van der Schaar. Disease-atlas: Navigating disease trajectories using deep learning. In *Proceedings of the Machine Learning for Healthcare Conference (MLHC)*, 2018.

[16] Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*, 2019.

[17] Bryan Lim, Stefan Zohren, and Stephen Roberts. Detecting Changes in Asset Co-Movement Using the Autoencoder Reconstruction Ratio. *arXiv e-prints*, page arXiv:2002.02008, January 2020.

[18] Bryan Lim, Stefan Zohren, and Stephen Roberts. Recurrent Neural Filters: Learning independent bayesian filtering steps for time series prediction. In *International Joint Conference on Neural Networks (IJCNN)*. 2020.

[19] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3):1–26, 03 2018.

[20] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. *arXiv e-prints*, page arXiv:2002.08071, 2020.

[21] Diaa Noureldin, Neil Shephard, and Kevin Sheppard. Multivariate high-frequency-based volatility (heavy) models. *Journal of Applied Econometrics*, 27(6):907–933.

[22] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[23] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, 2014.

[24] David Salinas, Valentin Flunkert, and Jan Gasthaus. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *arXiv e-prints*, page arXiv:1704.04110, 2017.

[25] Neil Shephard and Kevin Sheppard. Realising the future: forecasting with high-frequency-based volatility (heavy) models. *Journal of Applied Econometrics*, 25(2):197–231, 2010.

[26] Kevn Sheppard and Wen Xu. Factor high-frequency based volatility (heavy) models. *SSRN*, 2014.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, 2017.

[28] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*. 2017.

[30] Lai Z. and Deng H. Medical image classification based on deep features extracted by deep model and statistic feature fusion with multilayer perceptron. *Comput Intell Neurosci*, Sep 2018.