Global Models for Time Series Forecasting: A Simulation Study
Online Appendix

Hansika Hewamalage[a,*], Christoph Bergmeir[a], Kasun Bandara[b]

Hansika.Hewamalage@monash.edu, Christoph.Bergmeir@monash.edu, kasun.bandara@unimelb.edu.au

[a]*Dept of Data Science and AI, Faculty of IT, Monash University, Melbourne, Australia.*
[b]*School of Computing and Information Systems, Melbourne Centre for Data Science, University of Melbourne, Melbourne, Australia.*

Appendix A. Calculating Model Complexity

A mere comparison of the number of coefficients is not a reasonable comparison of model complexity across different model families. Degrees of freedom of a model is used to measure model complexity. A similar concept has been proposed in the work by Gao and Jojic (2016) to denote the complexity of NNs in a classification context. Minimum Message Length (MML) is another technique introduced by Wallace and Boulton (1968) also encapsulating the idea of degrees of freedom, for model selection and comparison. MML for a model is the joint representation of the model itself, followed by the encoding of the data as given by the model. Therefore, apart from the complexity of the model, MML also considers how good the model fits to the data. According to Wallace and Dowe (1999), while Kolmogorov complexity and MML are both equivalent, MML has the merit that it can accept a Non-Universal Turing Machine as the underlying reference machine, making it computationally more feasible than Kolmogorov complexity.

Appendix B. Data Generating Processes

Here, we include detailed explanations of the literature related to time series simulation as well as the DGPs used for our study.

Appendix B.1. Background

In the domain of time series classification, Sun et al. (2019) generate synthetic datasets using the Lorenz system, to empirically prove the competence of their proposed algorithm that uses covariance matrices as the considered features for the classification, which makes it interpretable. On the other hand, Zhao and Itti (2018) simulate artificial time series by stretching and scaling original time series in order to demonstrate the capability of their newly proposed variant of DTW, which is shapeDTW for sequence alignment.

There are a number of research works which involve time series simulation for forecasting as well. Bergmeir and Benítez (2012) and Bergmeir et al. (2018) use simulated time series to investigate the validity of cross-validation for time series forecasting. Hyndman et al. (2002) simulate time series using ETS models and later use these series to fit ETS models and thus, evaluate the accuracy of the model selection process employed in the automated ETS models. Petropoulos et al. (2014) simulate time series in their study with the overall objective of identifying dominant factors in time series data that result in improved or degraded forecasting accuracies in several univariate forecasting methods and their combinations. Specifically, they simulate fast-moving data with the factors seasonality, trend,

*Corresponding Author Name: Hansika Hewamalage, Affiliation: Dept of Data Science and AI, Faculty of IT, Monash University, Melbourne, Australia, Postal Address: Faculty of Information Technology, P.O. Box 63 Monash University, Victoria 3800, Australia, E-mail address: Hansika.Hewamalage@monash.edu

cycle, randomness, number of observations, and forecasting horizon, as well as intermittent data by considering the factors inter-demand interval, coefficient of variation, number of observations, and forecasting horizon.

Simulation studies have also been used to evaluate NNs and other Machine Learning (ML) algorithms for forecasting. Lau and Wu (2008) use simulations of complex, chaotic time series to demonstrate the competitive performance of their novel Support Vector Regression (SVR) model for forecasting non-linear time series. Ye and Dai (2021) use synthetic datasets simulated using the Lorenz system to evaluate their novel transfer learning scheme for time series forecasting using Convolution Neural Networks (CNN) as the underlying deep learning architecture. Similarly, Vanli et al. (2019) too use simulations of delay differential equations based series, specifically Mackey-Glass and Chua's circuit series to assess the performance of their incremental decision tree method on sequential non-linear regression problems. In their study, Fischer et al. (2018) perform simulation of time series data using eight linear and non-linear DGPs such as AR models, Threshold Auto-Regressive (TAR) models and Smooth Threshold Auto-Regressive (STAR) models to evaluate the performance of the eight ML algorithms multilayer perceptron (MLP), logistic regression, naive Bayes, k-nearest neighbours, decision trees, random forests and gradient-boosted trees. To make the evaluation process more robust to outliers, from each DGP, 1000 series are generated and results averaged over the 1000 cases. Although their study develops global forecasting models, they consider forecasting as a classification problem by classifying whether the time series rises or remains constant at time t. Zhang and Qi (2005) use time series simulation to address specific questions related to NNs for one-step-ahead forecasting. In particular, they attempt to investigate the capability of NNs to model seasonality and trend, whether data preprocessing is required prior to feeding the data into NNs and the benefit of using NNs over the traditional forecasting method ARIMA on seasonal time series. However, they build NNs in a local manner where one model is built per each series. Zhang (2007) uses a simulation study to measure the performance of his proposed univariate ensemble FFNN with jittered training data to augment the original set of training data. The time series simulation is carried out in terms of several DGPs including AR processes, Auto-Regressive Moving Average (ARMA) processes, Non-linear AR (NAR) processes, and STAR processes. Zhang et al. (2001) specifically simulate non-linear time series using non-linear DGPs such as Sign AR processes, TAR models, NAR models, Non-linear Moving Average (NMA) models and STAR models to assess the effect from the input nodes, hidden nodes of NNs, and sample size on the forecasting accuracy of FFNNs in comparison with ARIMA models. Similar to this, Suhartono et al. (2018) also perform a study using non-linear seasonal time series simulated with an Exponential Smooth Transition Auto-Regressive (ESTAR) model to explore determining factors such as the inputs and the number of neurons in the hidden layers on the FFNN's forecasting accuracy.

In the `R` programming language (R Core Team, 2020), there are several packages that provide the facility to simulate time series with many different statistical models. Table B.1 summarises these packages along with their respective functions. The underlying statistical models that these DGPs use include ARIMA models, Seasonal ARIMA (SARIMA) models, ETS models, Complex Exponential Smoothing (CES) models, Generalized Exponential Smoothing (GES) models, Simple Moving Average (SMA) models, Vector Exponential Smoothing (VES) models, TAR models and SETAR models. Unlike the other models, the Regime Switching models SETAR and TAR allow for creating structural change in the time series, thereby creating a non-linearity in the resulting time series. The number of thresholds determines the number of regimes in the series where every individual regime is modelled by a different AR process. However, not all these statistical models are easy to use in simulating series. For complex models such as CES, GES and others, Svetunkov (2019) recommend to always first fit the model of choice to a real time series and then use the derived coefficients to simulate new series. This is mainly because such complex models have many different parameters which are hard to control, and therewith it is hard to estimate a reasonable value for those parameters to obtain realistic time series.

Appendix B.2. Data Generating Processes Used for the Study

Further details of the DGPs specifically used in our study are as follows.

Table B.1: Data generation processes in the R programming language

R Package	Function	Details
`forecast` (Hyndman and Khandakar, 2008)	`simulate`	Generates series given fitted models from `ets`, `auto.arima`, and others.
`stats` (R Core Team, 2020)	`arima.sim`	Generates series given the AR, MA coefficients for an ARIMA model.
`CombMSC` (Smith, 2019)	`sarima.Sim`	Generates series given the AR, MA coefficients for a SARIMA model.

`smooth` (Svetunkov, 2019)	`sim.es`
	`sim.ces`
	`sim.ssarima`
	`sim.gum`
	`sim.sma`
	`sim.ves`
		Generate series out of ETS, CES, SARIMA, GES, SMA and VES models.

`tsDyn` (Fabio Di Narzo et al., 2019)	`setar.sim`	Generates series from a SETAR model. Either bootstraps given a series or simulates given the coefficients.
`TAR` (Zhang and Nieto, 2017)	`simu.tar.norm` `simu.tar.lognorm`	Generates series either from a TAR model or log-normal TAR model with Gaussian distributed errors, given the orders and coefficients of AR processes, the number of regimes and the thresholds.
`gratis` (Kang et al., 2020)	`generate_ts` `generate_ts_with_target`	Generates series from MAR models given the frequency and the number of mixing components. Either generates diverse series or series having a set of tsfeatures pre-specified.

Appendix B.2.1. Auto-Regressive Models

For the convenience of explanation of the AR models, we first introduce the backshift operator as in Equation B.1.

$$\begin{aligned} By_t &= y_{t-1} \\ B(By_t) &= B^2 y_t = y_{t-2} \end{aligned} \tag{B.1}$$

The backshift operator on the variable y_t is used to denote its lags.

Non-Seasonal Auto-Regressive Models

Equation B.2 defines a non-seasonal AR model of order p using the backshift operator.

$$y_t = c + \phi_1 B y_t + \phi_2 B^2 y_t + \phi_3 B^3 y_t + ... + \phi_p B^p y_t + \epsilon_t \tag{B.2}$$

Here, ϵ_t indicates a white noise error term, which is the error in the AR modelling, and c is a constant term which is also known as the intercept. As Equation B.2 shows, an AR model is a simple linear regression model where the past lags are the predictors of the value of the series at the t^{th} time step. The coefficients $\phi_1, \phi_2, ..., \phi_p$ are estimated to fit an AR model for a particular time series by minimising a loss criterion such as the Maximum Likelihood Estimation (MLE). Similarly, given the values for the coefficients $\phi_1, \phi_2, ..., \phi_p$ of the different lags, the AR model can simulate series. In this study, we use an AR(3) DGP to simulate simple linear patterns in the time series data.

We use the `arima.sim` function from the `stats` package of the R core libraries (R Core Team, 2020) to simulate series using AR(3) DGP. The `arima.sim` function is provided with AR coefficients generated for a stationary process. Rather than generating coefficients randomly and checking for their stationarity, we follow the procedure of Bergmeir and Benítez (2012) to randomly produce the

roots of the characteristic polynomial first and then generate the AR coefficients based on those. To obtain a stationary AR process, the coefficients should be such that the absolute values of the characteristic polynomial roots are greater than one (Cryer and Chan, 2008). Therefore, we first sample the roots of the characteristic polynomial randomly from a unifom distribution in the range $[-root_{max}, -1.1] \wedge [1.1, root_{max}]$ where $root_{max}$ is set to 5 (Bergmeir and Benítez, 2012). Since the DGP takes some time to reach stability in its generated values, to avoid issues from initial values of the DGP, a burn-in period of 100 points is cut off from the beginning of the series. Once the series are generated in this manner, some further processing is done on them according to the work of Bergmeir et al. (2014). First, the series are normalised to zero mean and unit variance. Then, to make the scenarios more realistic we make all the data non-negative by subtracting from every series its smallest value if that is negative. Since according to Hyndman and Koehler (2006) the error measures associated with the study such as Symmetric Mean Absolute Percentage Error (SMAPE) are prone to issues with values close to zero, as the next step we add one to the whole series if the minimum value is less than one. The series from the simple AR(3) DGPs are generated likewise.

Appendix B.2.2. Seasonal Auto-Regressive Models

To make the patterns more complex, we also simulate series using an SAR model. Equation B.3 shows such an SAR model of order P, where the period is denoted by S.

$$y_t = c + \Phi_1 B^S y_t + \Phi_2 B^{2S} y_t + \Phi_3 B^{3S} y_t + ... + \Phi_P B^{PS} y_t + \epsilon_t \tag{B.3}$$

Comparing Equation B.2 with Equation B.3, in the Seasonal AR model, the predictor variables are now seasonal lags as opposed to normal lags as in the non-seasonal AR model. Therefore, the Seasonal AR model is a regression model where the predictor variables are seasonal lags of the time series. Thus, an SAR DGP can simulate time series having a seasonality of a particular periodicity which are also very commonly seen in real-world scenarios. Similar to the non-seasonal AR model, given the values of the coefficients $\Phi_1, \Phi_2, ..., \Phi_p$, the model can simulate new time series. However, unlike in simple AR models, since it is the seasonal lags that are significant in the SAR models, they account for much longer-term dependencies, thus generating series with longer memory. In our study, to generate series with SAR models, we first fit an SAR model of order 1 to the `USAccDeaths` monthly series of the `datasets` package (R Core Team, 2020) available in the R core libraries. Then, we use this model to simulate series using the `simulate` function of the `forecast` package (Hyndman et al., 2020). The `USAccDeaths` series contains the monthly totals of accidental deaths in the USA for the period from the year 1973 to 1978, with 72 data points (Brockwell and Davis, 1991). This series shows strong yearly seasonal patterns as shown in Figure B.1.

We use this real-world series in this case as the base series for the homogeneous experimental settings, since that ensures the expected seasonality with long memory in the generated series, rather than randomly picking coefficients. The formula of the SAR(1) model fitted to the `USAccDeaths` series is as shown in Equation B.4. The value of the coefficient Φ_1 for this model is 0.85, which is close to 1 and thus resulting in significant long term memory in the series. A burn-in period of 100 is used to reduce the effects from initial values.

$$y_t = 9072.24 + 0.85 B y_t \tag{B.4}$$

Similar to the AR(3) DGP scenario, here also we scale the generated series to zero mean, unit variance and make all the data points greater than one. For the scenarios which introduce heterogeneity into the datasets, we also use the `sim.ssarima` function of the `smooth` package (Svetunkov, 2019) to simulate series with different SAR coefficients for every new series. The coefficients in the latter case are randomly sampled from the Uniform Distribution U(-0.5, 0.5).

Appendix B.2.3. Chaotic Logistic Map

The Chaotic Logistic Map DGP used in this study generates data as per Equation B.5. This is a zero-bounded chaotic process where ϵ_t indicates a white noise error term. r is the coefficient of the model.

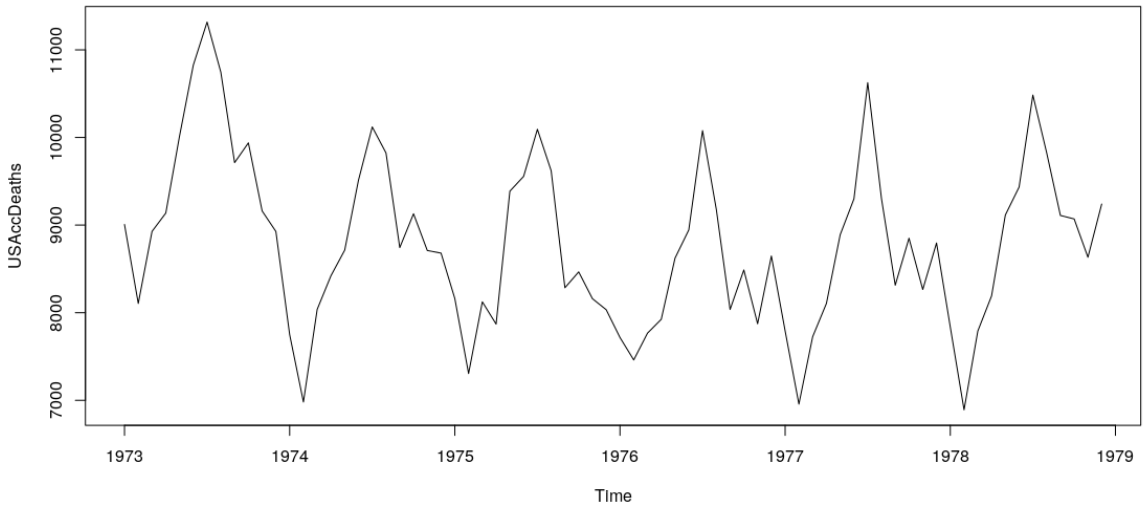

Figure B.1: USAccDeaths monthly series with yearly seasonality

$$y_t = max\left(ry_{t-1}\left(1 - y_{t-1}\right) + \frac{\epsilon_t}{10}, 0\right) \tag{B.5}$$

The datasets generated from the Chaotic Logistic Map DGPs may have zero values. We have specifically chosen the coefficient $r = 3.6$ and the initial value $y_0 = 0.5$ in all homogeneous series scenarios. This is because, for logistic maps the chaotic behaviour is heavily observed for $r \approx 3.6$. In this way, for all homogeneous cases, we make the simulated time series difficult to model by forecasting techniques. However, for heterogeneous datasets where different coefficients are involved, r and y_0 are chosen randomly from the Uniform Distributions $U(0, 4)$ and $U(0, 1)$, respectively, to align with findings from the literature (May, 1976). Similar to other DGPs, a burn-in period of 40 points is introduced to reduce the effects of initial values.

Appendix B.2.4. Self-Exciting Threshold AutoRegressive Models

A TAR model which is the basis for the SETAR model is defined as in Equation B.6.

$$y_t = \begin{cases} c_1 + \phi_1^1 B y_t + \phi_2^1 B^2 y_t + \phi_3^1 B^3 y_t + ... + \phi_p^1 B^p y_t + \epsilon_t, & \text{if } z_t \leq r_1 \\ c_2 + \phi_1^2 B y_t + \phi_2^2 B^2 y_t + \phi_3^2 B^3 y_t + ... + \phi_p^2 B^p y_t + \epsilon_t, & \text{if } r_1 \leq z_t \leq r_2 \\ ... \\ c_k + \phi_1^k B y_t + \phi_2^k B^2 y_t + \phi_3^k B^3 y_t + ... + \phi_p^k B^p y_t + \epsilon_t, & \text{if } r_{k-1} \leq z_t \end{cases} \tag{B.6}$$

According to Equation B.6, the TAR model involves $k-1$ number of threshold values $(r_1, r_2, ..., r_{k-1})$, which separate the space into k regimes, where each one is modelled by a different AR process of order p. The threshold variable is denoted by z_t which is an exogenous variable.

For this study, we use the simulation capability of SETAR models implemented in the `setar.sim` function of the `tsDyn` R package (Fabio Di Narzo et al., 2019). To avoid the practical difficulty of ensuring that the simulated series reaches the chosen threshold at a certain point, we use the AR coefficients and the threshold values presented in the example of the `tsDyn` package. This includes two AR(2) processes with coefficients $(c_1 = 2.9, \phi_1^1 = -0.4, \phi_2^1 = -0.1)$ and $(c_2 = -1.5, \phi_1^2 = 0.2, \phi_2^2 = 0.3)$ along with a single threshold value of 2 and initial values $(2.8, 2.2)$. This is the case for all homogeneous experimental scenarios. On the other hand, for the heterogeneous scenarios, if we randomly generate

AR coefficients, threshold values and initial values, we again face the issue of ascertaining whether the simulated observations go beyond the respective threshold value. Therefore, instead of generating coefficients randomly in this manner, to create the heterogeneous datasets of this DGP, for every new series we add a small amount of random noise to the initial coefficients. While doing so, it is also essential to retain the coefficients within the stationary bounds of the AR processes. Therefore, to meet all these requirements, the amount of noise added to the coefficients is sampled from a Normal Distribution of the form $N(0, 0.007)$. Similar to the AR(3) DGP, once the series are generated, they are z-score normalised and made greater than one.

Appendix B.2.5. Mackey-Glass Equation

The Mackey-Glass equation is defined as in Equation B.7.

$$\frac{dy_t}{dt} = \frac{\beta y_{t-\tau}}{1 + y_{t-\tau}^n} - \gamma y_t \tag{B.7}$$

In Equation B.7, τ is called the delay whereas $\beta, \gamma, n > 0$ are parameters that determine the periodicity and the chaos induced into the resulting series. Following from Equation B.7, the solution to the Mackey-Glass equation is as shown in Equation B.8. Therefore, Equation B.8 can be directly used to simulate complex time series.

$$y_{t+1} = y_t + \frac{\beta y_{t-\tau}}{1 + y_{t-\tau}^n} - \gamma y_t \tag{B.8}$$

For this study we use the Mackey-Glass based time series generation implemented in the `nolitsa` Python package (Mannattil, 2017). Most of the publicly available implementations of Mackey-Glass based series generation use $\beta = 0.2, \gamma = 0.1, n = 10$, and $\tau \geq 17$. These are also the default values in the `nolitsa` package, with τ being equal to 23. Usually, $n \approxeq 10$ are the values which create chaos in the resulting series. Following common practice, we use the default values of the parameters in the `nolitsa` package to generate series for all homogeneous scenarios. As for the heterogeneous cases, the values for β, γ, and n are retained at the defaults to ensure chaos while τ is sampled randomly from the Uniform Distribution $U(17, 100)$. Once generated likewise, the series are shifted to eliminate small values below 1.

Appendix B.2.6. Fourier Terms

A Fourier term is a pair of Sine and Cosine terms defined as in Equation B.9a. Where K is a defined number of Fourier terms, for example 5, where k goes from 1,2, ..., 5. m is the length of the seasonal period being modelled, such as 24 for hourly data with daily seasonality, 7 for daily data with weekly seasonality or 12 for monthly data with yearly seasonality etc. α_k and β_k are the coefficients for the Sine and Cosine components of the kth Fourier term F_{kt}^m modelling seasonal period m. The coefficients are the same throughout time, meaning that the seasonal components modelled remain fixed in shape throughout the whole series.

$$F_{kt}^m = \alpha_k sin\left(\frac{2\pi kt}{m}\right) + \beta_k cos\left(\frac{2\pi kt}{m}\right) \tag{B.9a}$$

$$y_t = \sum_{m \in M} \sum_{k=1}^{K} F_{kt}^m + \epsilon_t \tag{B.9b}$$

Fourier terms as in Equation B.9a, have to be defined for each seasonality in a multiple seasonal scenario. Once Fourier terms are defined like this, the series can be generated as in Equation B.9b, where ϵ_t denotes a random noise and M defines the set of all seasonal periods being modelled.

For our work, in the Fourier Terms, we set $M = (24, 168, 8760)$, to simulate hourly data with three seasonalities, daily, weekly and yearly. The number of Fourier terms $K = 5$ for all the periods,

although it can be different for different periods as well. The coefficients α_k and β_k are sampled from a normal distribution $N(0, 1)$. For homogeneous series a random noise with standard deviation 0.3 ($N(0, 0.3)$) is added to the coefficients where as to generate heterogeneous seasonal patterns the noise standard deviation is increased to 8 (sampled from $N(0, 8)$). ϵ_t is also sampled from a normal distribution $N(0, 0.5)$.

Appendix C. Forecasting Techniques

This section presents further information specific to the different forecasting techniques used in the study. Although the evaluation is performed on a large number of datasets (100/1000) for each scenario, to make the process more efficient we tune the hyper-parameters of the models only on one of those datasets for each scenario and apply that same optimal hyper-parameter combination in all the 100/1000 evaluations. Yet, on the one dataset where the hyper-parameters are tuned, they are tuned for all the different lengths and number of series to account for the data availability conditions.

Appendix C.1. Recurrent Neural Networks

RNNs are a type of NNs that are specialised for sequence modelling problems due to their states which are propagated up to the end of the sequence and thus help them distinguish between every individual series among a set of series. Like other NNs, RNNs too are universal approximators meaning that they are inherently non-linear models (Schäfer and Zimmermann, 2006). RNNs can be implemented with many different architectures. In the study by Hewamalage et al. (2020), those authors have identified the Stacked architecture as the generally best RNN architecture across a number of real-world datasets. Therefore, in this study we choose that same architecture along with residual connections. Residual connections were introduced by He et al. (2016) on CNNs mainly for the purpose of image recognition. Such networks were named as Residual Nets (ResNets). However, later many other domains were inspired by this architecture and implemented different modified versions of it for their own work. For the time series forecasting domain, Smyl and Kuber (2016a) used residual connections on RNN layers. Moreover, the winning solution by Smyl (2020) at the M4 forecasting competition also involved RNNs with residual connections. Inspired by this work, we use an architecture similar to the work of Smyl and Kuber (2016a) in our study. In particular, our RNN architecture is illustrated in Figure C.1.

As shown in Figure C.1, residual connections operate as a form of shortcut connections between layers. Specifically in this scenario, it allows to connect the input of one layer to the input of the next immediate layer in the stack, in addition to the usual information flow along the layers. The residual block which sits in-between layers, performs a simple addition of the output of one layer with the input of that same layer. Equation C.1 demonstrates this concept where $Z_{j,t} \in \mathbb{R}^d$ represents the input to the layer j and $h_{j,t} \in \mathbb{R}^d$ represents the output from the layer j at the time step t, d being the cell dimension. For LSTM cells, $h_{j,t}$ is also the hidden state of the layer j whereas $C_{j,t}$ indicates the cell state at the time step t. Thus, the residual connections start from the second layer, since the input to the first layer $X_t \in \mathbb{R}^m$ is of the input window size and the rest has the size of the cell dimension which makes it impossible to perform an addition operation between them.

$$Z_{j,t} = h_{j-1,t} + Z_{j-1,t} \tag{C.1}$$

The residual connections are especially helpful to avoid vanishing gradient issues of the layers of NNs. This is because, with residual connections, the output of every layer $h_{j-1,t}$ corresponds to a residual mapping in addition to the identity mapping performed by the connection from the input $Z_{j-1,t}$. Therefore, even if the upper layers perform very poorly, the final output would be at least as good as the initial input. In other words, the final error of the overall network should not go beyond the error from its shallow counterparts. In contrast, in the usual Stacked architecture as in the work of Hewamalage et al. (2020), the stack of layers attempts to model the final expected mapping. He et al. (2016) also state that it is easier for layers to learn a residual mapping than the true underlying

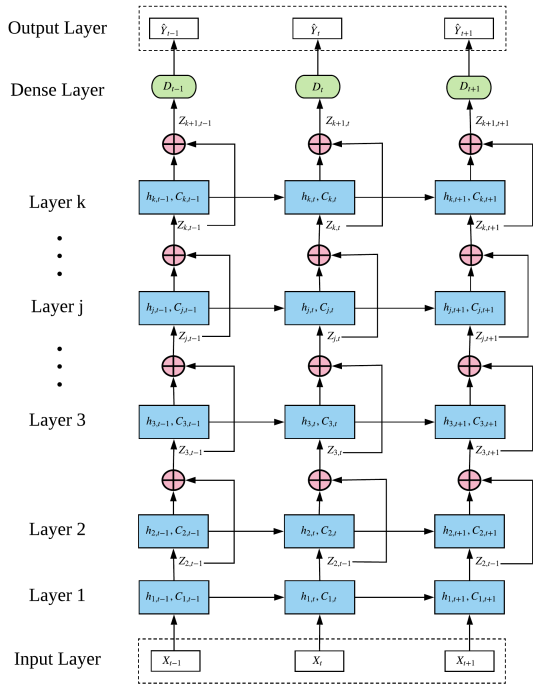

Figure C.1: Recurrent Neural Network with residual connections

mapping. The benefit of these shortcut connections is that they come with no cost of additional trainable parameters. On top of the final layer of the RNN with k layers, an affine neural layer (fully connected layer without a bias vector) indicated by D_t exists which converts the output from the RNN $Z_{k+1,t} \in \mathbb{R}^d$ into the final output $\hat{Y}_t$ having the size of the expected forecasting horizon H.

The RNNs used for this study are implemented with version 2.0 of the Tensorflow open-source deep learning framework (Abadi et al., 2015). Throughout the study we use the stacked architecture along with the COntinuous COin Betting (COCOB) optimiser introduced by Orabona and Tommasi (2017) as the underlying learning algorithm for the Backpropagation Through Time (BPTT) procedure. The COCOB optimiser relieves the user from defining an initial learning rate, which makes the whole process far more automatic. We use a Tensorflow-based implementation of the optimiser available as an open-source library (Orabona, 2017). The RNN unit that constitutes the networks is the LSTM cell with peephole connections introduced by Gers et al. (2003). The choice of this RNN architecture, optimiser and the recurrent cell type follows from the conclusions of a recent study by Hewamalage et al. (2020).

A number of hyperparameters are tuned for RNNs. However, these hyperparameters and their initial ranges are different between the scenarios which involve just a single series in the experiment and many series. For the experiments with a single series, the hyperparameters tuned include the cell dimension, the number of layers, the number of epochs, the L2 regularisation parameter, the standard deviation of the Gaussian noise added, and the standard deviation of the normal distribution from which values are randomly picked for initialisers of the network. Besides these hyperparameters, the experiments with multiple series also tune the mini-batch size and the epoch size. All these hyperparameters are used in this study in the same notion as in the work of Hewamalage et al. (2020). The Gaussian noise inclusion and the L2 regularisation are used as means of reducing the overfitting effects of the RNNs to the training data. The epoch size denotes the number of times that the whole dataset is repeated within one epoch. Therefore, epoch size and mini-batch size are both

Table C.1: Initial hyperparameter ranges for RNNs

Hyperparameter	Many Series Setting	Single Series Setting
Cell Dimension	20-50	20-50
Number of Layers	1-2	1-2
Mini-batch Size	10-100 or 1-100	-
Number of Epochs	5-30	20-300
Epoch Size	1-10	-
L2 Regularisation Param.	0.0001-0.0008	0.0001-0.0008
Std. of Gaussian Noise	0.0001-0.0008	0.0001-0.0008
Std. of Initialiser	0.0001-0.0008	0.0001-0.0008

not relevant in the context of a single series. For tuning the hyperparameters we use the Sequential Model-based Algorithm Configuration (SMAC) tool introduced by Hutter et al. (2011). In particular we use the version 0.11.1 of the `smac` Python package (Lindauer et al., 2017). For the tuning of the hyperparameters of each experiment, we use 25 or 50 `smac` iterations (depending on the complexity of the experiment) while evaluating the accuracy on a validation dataset. The common initial ranges of the hyperparameters provided to the SMAC tool for both the single series and multiple series scenarios are as mentioned in Table C.1.

Since the single series scenarios do not involve the epoch size hyperparameter, we need to ensure that these scenarios will still receive approximately the same amount of weight updates as the multiple series settings. Therefore, for the single series scenarios, we have increased the initial range for the number of epochs to 20-300. Since the LSTM cells themselves are complex RNN units, the number of hidden layers can be as low as 1-2, to control the complexity of the overall architecture (Hewamalage et al., 2020). The total number of series (100 for each multiple series scenario) is merely an upper limit on the mini-batch size. The actual mini-batch size is rather dependent on the available memory capacity of the system and the total computational costs. A smaller mini-batch size is expected to use a smaller memory amount at the cost of higher computation times. On the other hand, larger batch sizes take comparatively less training time at the cost of more memory. Furthermore, recent studies have demonstrated superior performance of RNNs when trained with gradient descent of mini-batch size 1 as opposed to other mini-batch sizes (Smyl, 2020). Depending on the scale of the different experimental scenarios employed in this study, we set the lower bound of the initial range for the mini-batch size to either 1 or 10. Due to computational constraints resulting from the long lengths of the series associated with MS-Hom-Long scenarios, for those experiments we set the lower bound of the mini-batch size to 10. During every epoch of the training process, the dataset is randomly shuffled before fed into the RNN. This makes the RNN encounter the data in different sequences in every epoch and improves its generalisation capability. For the training loss in the RNNs we use the Mean Absolute Error (MAE) along with the L2 regularisation loss as per the work of Hewamalage et al. (2020). For the purpose of the clustering experiments, one RNN is built separately for the set of series from each DGP.

Appendix C.2. Feed-Forward Neural Networks

An FFNN can be described as the most basic type of an NN. The neurons within a single layer of an FFNN have no feedback connections as in an RNN and the neurons of each layer feed their outputs directly into the next immediate layer of the stack. Thus, FFNNs, when used for forecasting problems as described here, do not differentiate between individual time series as an RNN, since they do not have per-series states that propagate towards the end of the series. FFNNs can only distinguish between windows when used in a moving window scheme. The basic architecture of an FFNN is as illustrated in Figure C.2.

The layers that we use in the FFNNs are fully connected. In Figure C.2, m and n denote the input and output window sizes, respectively, where l indicates the total number of layers including the

output layer. As shown in Figure C.2, the number of nodes in the input and output layers correspond to the input and output sizes of the windows.

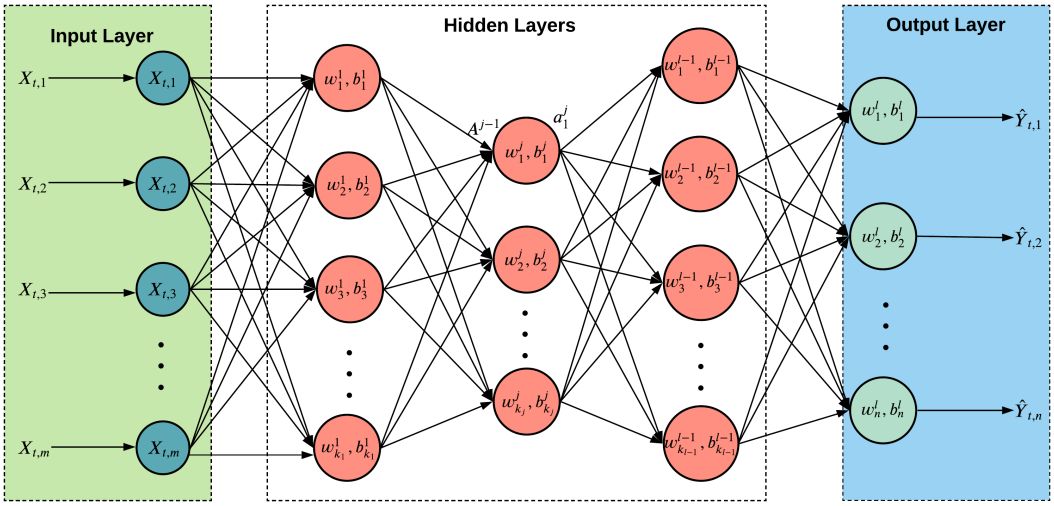

Figure C.2: Feed-Forward Neural Network

$$z_i^j = w_i^j \odot A^{j-1} + b_i^j \tag{C.2a}$$
$$a_i^j = f(z_i^j) \tag{C.2b}$$

$$Z^j = W^j \odot A^{j-1} + B^j \tag{C.3a}$$
$$A^j = f(Z^j) \tag{C.3b}$$

Equation C.2 defines the functionality of the i^{th} neuron of layer j in an FFNN where $z_i^j, b_i^j, a_i^j \in \mathbb{R}$ and $w_i^j, A^{j-1} \in \mathbb{R}^{k_{j-1}}$. Here, k_{j-1} denotes the number of hidden nodes in layer $j-1$ of the FFNN. w_i^j and b_i^j are the weights vector and the bias of the neuron, respectively. A^{j-1} are the final outputs at the $j-1$ layer, z_i^j is the output from the linear mapping and a_i^j is the final output from the neuron. Therefore, Equation C.2a is simply a linear function of the outputs of the previous layer whereas the activation function f in Equation C.2b introduces the non-linearity into the model. Although there are many options for this activation function, in our study we use the *tanh* function. Equation C.3 depicts the same functionality as in Equation C.2, but on the whole layer instead of just one neuron. In Equation C.3, $Z^j, B^j, A^j \in \mathbb{R}^{k_j}$, and k_j indicates the number of hidden nodes in layer j of the FFNN. $W^j \in \mathbb{R}^{k_j \times k_{j-1}}$ is the weight matrix of layer j of the FFNN and $A^{j-1} \in \mathbb{R}^{k_{j-1}}$ is as in Equation C.2.

The FFNNs used for this study are implemented with version 2.0 of the Tensorflow open-source deep learning framework (Abadi et al., 2015). Unlike the RNNs, for the FFNNs, we use the well-known Adam optimiser as the underlying learning algorithm (Kingma and Ba, 2015). From a preliminary study involving few experimental scenarios, the Adam optimiser was better for the FFNNs than the COCOB optimiser. Furthermore, although the literature holds evidence for the competitive forecasting performance of RNNs with the COCOB optimiser, there is no such evidence for any FFNN architectures with respect to forecasting. Therefore, for FFNNs we use the standard built-in Tensorflow implementation of the Adam optimiser.

Table C.2: Initial hyperparameter ranges for FFNNs

Hyperparameter	Many Series Setting	Single Series Setting
Number of Layers	1-5	1-5
Number of Hidden Nodes	3-input size	3-input size
Mini-batch Size	10-100 or 1-100	-
Number of Epochs	5-60	20-300
Epoch Size	1-10	-
Initial Learning Rate	0.0001-0.1	0.0001-0.1
L2 Regularisation Param.	0.0001-0.0008	0.0001-0.0008
Std. of Gaussian Noise	0.0001-0.0008	0.0001-0.0008
Std. of Initialiser	0.0001-0.0008	0.0001-0.0008

The initial hyperparameter ranges used for the FFNNs are as shown in Table C.2. The cell dimension of RNNs is not relevant in the context of FFNNs. Apart from the other hyperparameters of the RNNs, the FFNNs also have a number of hidden nodes in each layer and the initial learning rate of the Adam optimiser which need to be tuned. The number of hidden nodes at each layer can be different from each other. To set values for the number of hidden nodes, we employ the commonly used heuristic to restrict the range in between the input window size and the output window size. In our forecasting scenarios, the output window size is always less than the input window size and the smallest output window size in any scenario is 3. Therefore, the initial range for the number of hidden nodes is varied in between 3 and the input window size in each scenario. Since the neurons in an FFNN are not as complex as LSTM cells, to increase the complexity of the overall model with respect to the RNN architectures, we set the initial range for the number of layers to be 1-5. Furthermore, for the learning rate of the Adam optimiser, a value in between 0.0001 and 0.1 is set. All the other hyperparameters are tuned analogously to the RNNs.

Apart from the architectural differences and the aforementioned hyperparameters, the training process of both RNNs and FFNNs are the same where the SMAC tool is first used to tune the hyperparmeters and later, the optimal hyperparameter combination is used for model testing. The loss functions used are also similar to RNNs. For the clustering experiments, similar to the RNNs, FFNNs too are built as one per each cluster.

Appendix C.3. Light Gradient Boosting Models

We also use LGBMs in our study as another forecasting technique. Extreme Gradient Boosting (XGBoost) models (Chen and Guestrin, 2016) recently have become popular by winning many ML related challenges. Gradient Boosting Models (GBM) are a type of ML algorithms based on decision trees and can be used to solve a wide range of ML problems including regression, classification, and others. As the name implies, this technique uses boosting as an ensembling approach among different decision trees. Although a quite competitive model, XGBoost can become extremely slow when fed with lots of data. This is where LGBM comes into play. As the very name suggests, LGBM is a lighter version of GBM which trains faster with fewer memory usage while achieving higher accuracy over the original GBMs (Ke et al., 2017). The key to this performance gain are two new techniques introduced in LGBM, named as the Gradient-based One-Side Sampling and Exclusive Feature Bundling which support dealing with large amounts of data with many features. The idea behind these techniques is to systematically reduce the number of data instances and features considered for computing the information gain at each split-point of the decision tree.

Due to the non-linear nature of decision trees, which is the fundamental building block of LGBMs, they too are non-linear models. The competitive performance of LGBMs when used for forecasting has been most recently demonstrated at the M5 Forecasting Competition, where an LGBM-based solution won the first place followed by many other LGBM-based solutions securing top ranks (Makridakis et al., 2020). However, similar to FFNNs, LGBMs by default have no notion of the individual series of

a dataset, when used for forecasting. They are only aware of the windows and they do not remember anything beyond what they see immediately within this input window. Since LGBMs can produce only a single output, to output windows using LGBMs, we build one model per every step in the output horizon rather than using a recursive strategy. This is similar to the functionality of the final layer of an NN which has a separate set of weights, and thus a separate function for every point in the forecast horizon. However, in the LGBM the whole model is different for the individual steps in the horizon. Nevertheless, the efficiency encouraged by the LGBM implementation makes this procedure computationally feasible.

Many of the hyperparameters that LGBMs have are associated with the underlying random forest, and we leave most of them at their default values as specified by the `lightgbm` Python package (Ke et al., 2017). This is mainly because, in practice the LGBM models perform quite competitively even without extensive hyperparameter tuning. To avoid the overhead of ensuring that the models do not overfit, we employ an early stopping mechanism to stop the training when an improvement in the validation error metric is not achieved over 5 consecutive epochs. The maximum number of epochs and the initial learning rate are specified as $1,200$ and 0.075, respectively.

In our study we use the LGBM implementation available in the Python package `lightgbm` (Ke et al., 2017). Also, the same input and output format as the NNs, is used for the LGBMs as well. However, since the LGBMs have no notion of the individual series beyond the input and output windows, the validation periods are not required to be reserved from the end of the sequences. The validity of using standard K-fold cross validation for time series forecasting in this way, is proved by Bergmeir et al. (2018) in their work. Therefore, we use a standard 7:3 split for the training and validation datasets of the LGBMs. MAE is used as the objective function for training. For the purpose of the clustering based experiments, with LGBMs we do not really perform a clustering of the dataset as is done with the RNNs and FFNNs, rather we only feed a categorical external feature as an additional input to indicate which out of the different DGPs each series belongs to. Due to the tree-based nature of the LGBM, the tree can be split at the top based on this extra feature to build completely different sub-trees for each of the DGPs as appropriate. Therefore, LGBM in this sense can perform an implicit clustering of the data with the help of the external categorical feature.

Appendix C.4. Linear Auto-Regressive Models

We also use local AR models, both non-seasonal and seasonal as other forecasting techniques in the study. This means that they are built as local models which learn only from the past values of the target series, and not across series. The modelling process of these AR models is the same as described in the format of DGPs in Equations B.2 and B.3 of our manuscript. AR models are linear in the lags of the target series. Unlike NNs, AR models are trained by considering a sequence of only one-step-ahead predictions. To implement the AR and SAR models for forecasting, we use the `Arima` function of the `forecast` package in the R programming language by providing the required seasonal or non-seasonal orders (Hyndman et al., 2020). The loss function used for training is the default in the `forecast` package, the Maximum Likelihood Estimate (MLE), which is effectively Mean Squared Error (MSE) as a normal distribution of the errors is assumed.

Appendix C.4.1. Auto-Regressive Integrated Moving Average Models

Apart from the aforementioned AR models, MA models the future in terms of a linear combination of the past forecast errors. AR and MA models, once combined together with sufficient differencing of the time series to achieve stationarity, are called ARIMA models. ARIMA models the autocorrelation in the data (Hyndman and Athanasopoulos, 2018). ARIMA model too is a local model trained by considering one-step ahead forecasts. The number of lags used for the AR model (p/P) and the MA model (q/Q) and the amount of differencing (d/D) defines the complexity of the underlying ARIMA model. We use ARIMA as a forecasting technique specifically on the real-world datasets where we have no knowledge of the true underlying DGP. We use the `auto.arima` function from the `forecast` package in the R language (Hyndman et al., 2020), which performs automatic selection of the best orders for the AR, MA terms as well as the differencing. For a SARIMA model, the maximum values

of P, Q and D are 2, 2 and 1 respectively. The underlying loss function for model training in ARIMA is MLE which is the same as in AR.

Appendix C.4.2. Dynamic Harmonic Regression with ARIMA Errors

DHR-ARIMA is used as a benchmark specifically for the simulation scenarios involving Fourier Terms. The DHR part of the model deals with the seasonality by using Fourier Terms as defined in Equation B.9a. The ARIMA part deals with any remaining autocorrelations in the series by fitting an ARIMA model to the remainder from the DHR model. The combined DHR-ARIMA model can be defined as in Equation C.4 where N_t denotes an ARIMA process.

$$y_t = \sum_{m \in M} \sum_{k=1}^{K} F_{kt}^{m} + N_t \tag{C.4}$$

Similar to ARIMA and AR, DHR-ARIMA too is a local model trained using one-step ahead forecasts. To implement DHR-ARIMA in our work, we use `auto.arima` function from the `forecast` package. Fourier terms are defined by the `fourier` function and added as external regressors of ARIMA using the `xreg` argument (Hyndman et al., 2020). The loss function is the default in the `forecast` package, the MLE.

Appendix C.5. Self-Exciting Threshold Auto-Regressive Models

For the scenarios involving the datasets generated from the SETAR DGP, we also experiment with SETAR as a forecasting technique. The underlying modelling process of the SETAR forecast model is as described in Equation B.6 under Appendix B.2.4. Due to the inherent regime-switching nature, SETAR is a non-linear forecast model. Similar to the AR models, SETAR model too is built as a local forecasting model fitted to every individual series separately using one-step-ahead forecasts. In this study, to implement the SETAR models we use the `setar` function from the `tsDyn` package in the R programming language (Fabio Di Narzo et al., 2019). The loss function used for the SETAR model is the default in the `tsDyn` package for the `setar` function, which is Conditional Least Squares (CLS) estimate.

Appendix C.6. Pooled Regression Models

We implement PR models similar to the work of Trapero et al. (2015) as another forecasting technique for the study. The PR model is effectively a global version of an AR model. The term pooling indicates that the coefficients of such regression models are calculated by pooling across many series. However, compared with, e.g., RNNs, the pooled regression models can only capture linear relationships in the data. Also, similar to FFNNs, PR models also do not maintain a state per every time series in the dataset. A PR model of order p can be defined as in Equation C.5. This equation is the same as in Equation B.2 for the non-seasonal AR model, except now the coefficients Θ are calculated by pooling across all the series in the dataset.

$$y_t = c + \Theta_1 B y_t + \Theta_2 B^2 y_t + \Theta_3 B^3 y_t + ... + \Theta_p B^p y_t + \epsilon_t \tag{C.5}$$

PR models in this context are used to distinguish potential gains of using non-linear models such as RNNs, FFNNs, LGBMs as opposed to linear models as global models. The PR models may also assist in identifying accuracy gains from purely using cross-series information without any complex architectural additions from RNNs (Hewamalage et al., 2020). Similar to the AR models, PR models too are trained using one-step-ahead predictions. To implement the PR models, we use the `glm` function in the `stats` package of the R core libraries (R Core Team, 2020). As for the clustering based experiments, for PR models too, similar to LGBMs we do not perform an explicit clustering of the dataset. Instead, we use interaction terms supported by the `glm` function which allows to have an interaction between the categorical external feature which indicates the DGP and the other lags fed to the regression model. This way, the coefficients of the lags change for each DGP and thus create

an implicit clustering based model for the PR. The loss function used for the PR models is Iteratively Re-weighted Least Squares (IRLS) which is the default in the `glm` function from the R package `stats` used in our experiments.

Appendix D. Data Preprocessing

The details of the data preprocessing steps are as follows.

Appendix D.1. Time Series Normalisation

The different time series that constitute a dataset may be in different scales. Although not quite problematic for local models built per each series, having series with such different scales can have adverse effects on global models (Sen et al., 2019). Therefore, we strive to normalise the series for scale before feeding into the global forecasting models by performing a mean scale transformation as done in the work of Bandara et al. (2019). This step is performed as depicted in Equation D.1 where l denotes the length of the i^{th} time series, n the number of time series available and $\tilde{y}_{i,t}$, the mean scaled value of the t^{th} time step of the i^{th} time series.

$$\tilde{y}_{i,t} = \frac{y_{i,t}}{\frac{1}{l}\sum_{t=1}^{t=l} y_{i,t}} \qquad \text{for } i = 1, 2, \ldots, n \tag{D.1}$$

Appendix D.2. Logarithmic Transformation

In this study, we train the forecasting techniques to model the seasonality and trend components of the time series by itself, rather than extracting them from the series beforehand. This is mainly because it is our objective to perform minimal preprocessing on the data and get the models themselves to fit the data and do the forecasting as appropriate. Although not very common with the DGPs used in this study, time series may often contain exponential trends and multiplicative seasonal patterns. Especially, NNs have saturation effects after certain thresholds due to their activation functions such as *tanh*. Therefore, the predicted values from NNs are anyway bounded by the limits of such transfer functions which may result in large forecasting errors especially in scenarios with exponential trends (Bandara et al., 2019). The reasons behind using logarithmic transformations are stabilising the variance in the data, scaling down the time series values and removing multiplicative seasonality components prior to applying additive seasonality decomposition techniques (Hewamalage et al., 2020). Hence, logarithmic transformations are capable of stabilising the variance in the data by removing exponential trends in time series and converting them into linear trends. Thus, in this study we use log transformation to ensure that the predictions of ML models are consistent when modelling the originally exponential trend components directly, as well as to scale down the time series further after the mean scaling step. In this study we use the log transformation analogous to the work of Hewamalage et al. (2020). This is done as indicated in Equation D.2 where y_t denotes the value of the time series at the t^{th} time step.

$$w_t = \begin{cases} log(y_t) & \text{if } min(y) > \epsilon, \\ log(y_t + 1) & \text{if } min(y) \leq \epsilon \end{cases} \tag{D.2}$$

In Equation D.2, ϵ can be defined as either zero for count data or a small positive value close to zero for real-valued data (Hewamalage et al., 2020). However, for this study, we set the value of ϵ to 0. For 0 values in the data, they are shifted by adding 1 to make the log transformation possible. This study does not involve series with negative values.

Appendix D.3. Moving Window Approach

The forecasting scenarios that we simulate in this study are all multi-step-ahead forecasting settings, which means that the models are required to produce forecasts for a forecasting horizon which comprises of more than one future time step. For such forecasting problems, with NNs it has been shown in the literature that the direct multi-step-ahead strategy which directly predicts the whole horizon is the best approach (Ben Taieb et al., 2012; Wen et al., 2017; Hewamalage et al., 2020). This is mainly due to the error accumulation issues associated with the recursive schemes which use previous forecasts as the input data for the successive prediction steps. Therefore, to feed the data into the NNs (RNNs, FFNNs) in this study we employ the Multi-Input Multi-Output (MIMO) strategy enforced by the moving window scheme, as also discussed by Smyl and Kuber (2016b). For LGBMs, the same moving window input and output format is used. However, LGBMs are only capable of producing one output at a time, not multiple outputs for a window. Therefore, to cater for this requirement, rather than using a recursive strategy, the training procedure for LGBMs is slightly modified to train one model per every step in the forecast horizon. On the other hand, both AR models and their corresponding global version PR models use a recursive strategy by taking as input a window of time series values corresponding to the specific AR order. SETAR models too use a mechanism of recursive one-step-ahead forecasts to produce an output window. The moving window scheme implemented in this study is the same as the one from Hewamalage et al. (2020).

Appendix D.4. Window Normalisation

As mentioned before in Section Appendix D.2, the log transformation itself supports the modelling process of machine learning models, by scaling down the values of the time series and thus reducing potential risks from saturation effects. However, when preprocessed using the moving window strategy, the time series values can be further normalised within the individual input-output window pairs. To this end, we perform a within-window local normalisation of the data similar to the work of Bandara et al. (2019). Consequently, the mean of every input window is subtracted from the corresponding input-output window pair. This mechanism helps particularly in scenarios where the size of the input window is comparatively high, resulting in the last values within the input window to lie outside of the saturation regions due to trends present in the data. This local normalisation step helps NNs to produce conservative but yet responsive forecasts using their inherent saturation effects by shifting the time series values to lie within the saturation bounds (Bandara et al., 2020). The models that use this per-window normalisation are the same as those which use the moving window scheme.

Appendix E. Performance Measures

Equation E.1 and E.2 denote the SMAPE and MASE metrics, respectively.

$$SMAPE = \frac{100\%}{H} \sum_{k=1}^{H} \frac{|F_k - Y_k|}{(|Y_k| + |F_k|)/2} \tag{E.1}$$

$$MASE = \frac{\frac{1}{H} \sum_{k=T+1}^{T+H} |F_k - Y_k|}{\frac{1}{T-M} \sum_{k=M+1}^{T} |Y_k - Y_{k-M}|} \tag{E.2}$$

In Equations E.1 and E.2, F_k and Y_k indicate the forecasts and the actual values, respectively, where H denotes the size of the forecast horizon. For the data containing 0 values, Equation E.3 is used as the denominator of the usual SMAPE formula to compute the SMAPE variant.

$$max(|Y_k| + |F_k| + \epsilon, 0.5 + \epsilon) \tag{E.3}$$

Following the work of Hewamalage et al. (2020), we set ϵ in Equation E.3 to 0.1, a small positive quantity. However, the SMAPE error metric also has other issues such as its lack of interpretability (Hyndman and Koehler, 2006). This is why we also use the MASE metric which was introduced by Hyndman and Koehler (2006).

Both SMAPE and MASE are scale-independent error measures. Furthermore, MASE is a relative error metric meaning that it measures the performance of the model with respect to the average in-sample one-step ahead error of some other standard benchmark such as the naïve forecast for data with no seasonality or the seasonal naïve forecast for seasonal data. As seen in Equation E.2, the denominator indicates this error from the relative benchmark where M denotes the seasonal period in case of seasonal data. According to Hyndman and Koehler (2006), MASE is the best performance metric available, being less susceptible to outliers and more interpretable than other error metrics. However, Equations E.1 and E.2 only define the error calculation for a single series.

Appendix F. Computational Resources

The computing resources for the experiments are allocated from the Massive High Performance Computing Facility (eResearch Centre., 2019). Thus, the experiments are run on an allocation having an Intel Xeon E5-2680 v3 processor (2.50 GHz), 6 CPU cores with 2 threads per core and a main memory of 64GB.

Appendix G. Analysis of Results

Here, we include further analysis of the results obtained from our experiments.

Appendix G.1. Different DGPs

Results pertaining to different DGPs are presented first.

Appendix G.1.1. AR(3) DGP

In the SS setting of the AR(3) DGP, the AR(3) model performs the best, with the AR(2) being the second and the AR(10) the third, with respect to mean SMAPE. Since AR(3) DGP is a subclass of the AR(10) forecast model, the training process of the AR(10) model resolves to learning that the coefficients beyond the third lag should be close to 0. Among the non-linear models, RNN(3) is the closest to the best performing model AR(3). The observation that the AR(2) is better than AR(3) at the beginning lengths, complies with the findings of Kunst (2008) that an AR(2) can be better than an AR(3) if there are not enough data for the model to fit all parameters reliably. A similar situation is observed among the AR(10) model and the other non-linear models RNN(3), FFNN(3) and LGBM(3). Due to the high amount of coefficients to be trained in the AR(10) model, the small lengths of the series are not sufficient to capture the underlying AR(3) process clearly.

The MS-Hom-Short scenario reaffirms these findings by Kunst (2008). In the MS-Hom-Short scenario the RNN(3) model is the next best to the PR(3). Furthermore, in a very short length scenario like this, it is evident how the global models such as PR(3) and RNN(3) which learn from all the series can be better than the local models, even AR(3) which is the one closest to the true DGP. At the beginning dataset sizes the AR(2) is the best model and gradually the PR(3) starts to outperform as the number of series in the dataset increases. AR(10) remains the worst performing model across all dataset sizes.

In the MS-Hom-Long scenario, the difference of the performance between the two models PR(3) and PR(10) is quite small. Similar to the MS-Hom-Short scenario, the PR models still remain the best out of all the models, in terms of Mean SMAPE. It is also seen that adding more memory/complexity to the models, supported by the increased lengths of the series helps the complex models learn better. This is because, the lag 10 models all outperform their lag 3 variants for the complex models. In fact, with respect to mean MASE, RNN(10) demonstrates an equivalent performance to that of the best models PR(3), PR(10). In terms of Mean SMAPE, the RNN(10) model demonstrates the same result as the AR(3) model which is the true DGP. Also, in this scenario, both RNN variants are better than the AR(10) model in terms of Mean SMAPE values. This indicates that 180 is still not enough length

for the AR(10) model to identify the underlying AR(3) pattern, since on the other hand in the SS setting with much longer lengths, the AR(10) outperforms the RNN.

In the MS-Het setting, the dataset can be seen as no longer holding a single simple linear pattern, but a combination of many of them. This is why the local models that train upon each series become competitive in this case. Also, similar to the MS-Hom-Long scenario in this setting too, lag 10 variants outperform the lag 3 variants. Among the global models, the RNN(10) model performs better than the linear PR models since the inherent complex, non-linear modelling capacity of the RNN(10) model can account for the heterogeneity among the series to a certain extent. According to the *Diff* column in Table 9 of the paper, the percentage difference between the best performing local models and the RNN(10) is very small (4.77% in terms of Mean SMAPE) compared to the differences of other models. Although AR(10) outperforms all the global models at the maximum length, it performs the worst in the beginning lengths. Therefore, at such short lengths, having global models, especially complex non-linear ones such as RNNs in a heterogeneous setting like this, which learn across all the short series together can be quite competitive. It is also important to observe that the MASE values of all the non-linear models in all the scenarios are less than 1, indicating that their performance is better than the performance of the naïve forecasts in-sample.

Table G.1: Mean SMAPE results for AR(3) DGP SS scenario

Length of Series	RNN(3)	FFNN(3)	LGBM(3)	AR(2)	AR(3)	AR(10)
18	23.02	24.82	23.67	18.48	19.26	32.72
36	21.58	21.07	21.77	18.1	18.27	20.62
54	20.54	21.12	21.59	18.08	18.3	19.77
72	19.16	21.57	21.72	17.94	18.02	18.93
90	19.62	20.36	20.61	17.71	17.85	18.9
180	19.41	19.46	19.99	17.63	17.66	18
360	18.84	18.7	19.1	16.67	16.69	16.9
540	19.99	20.09	19.35	17.5	17.48	17.58
720	17.89	19.22	19.61	17.45	17.47	17.62
900	19.06	18.86	19.29	17.17	17.2	17.25
1,350	18.53	19.44	19.87	17.87	17.86	17.92
1,800	18.04	19.11	19.25	17.48	17.47	17.55

Table G.2: Mean SMAPE results for AR(3) DGP MS-Hom-Short scenario

No. of Series	RNN(3)	FFNN(3)	LGBM(3)	AR(2)	AR(3)	AR(10)	PR(10)	PR(3)
1	23.06	23.32	23.67	18.48	19.25	32.72	83.30	20.65
2	20.44	22.04	21.95	19.01	19.49	31.51	96.31	18.69
3	20.45	21.02	20.98	19.27	20.04	33.29	35.80	18.87
4	19.88	20.60	21.26	18.95	19.59	31.72	26.36	18.35
5	19.72	20.36	20.73	19.07	19.77	33.30	23.11	18.27
10	19.00	19.68	20.25	18.90	19.33	33.07	19.99	17.94
20	18.10	18.78	19.51	18.26	19.20	32.95	18.41	17.08
30	18.40	19.19	19.53	18.85	19.40	32.19	18.45	17.83
40	18.19	19.25	19.65	19.14	19.76	32.31	18.63	17.86
50	18.08	18.94	19.23	18.45	19.12	33.72	18.15	17.55
75	18.58	19.53	19.86	19.57	20.04	32.32	18.61	18.13
100	18.12	18.91	19.25	18.79	19.53	32.92	18.15	17.71

Table G.3: Mean SMAPE results for AR(3) DGP MS-Hom-Long scenario

Length of Series	RNN(10)	RNN(3)	FFNN(10)	FFNN(3)	LGBM(10)	LGBM(3)	AR(2)	AR(3)	AR(10)	PR(10)	PR(3)
18	-	-	-	-	-	-	22.71	23.44	39.29	22.02	21.43
36	21.46	21.70	21.87	22.98	22.03	23.36	21.73	22.05	24.98	21.28	21.20
54	21.32	21.43	21.82	22.96	21.89	23.26	21.44	21.63	23.38	21.15	21.11
72	21.26	21.45	21.84	22.98	21.84	23.19	21.36	21.49	22.68	21.13	21.11
90	21.25	21.35	21.77	22.91	21.74	23.08	21.22	21.33	22.25	21.04	21.03
108	21.22	21.59	21.69	23.03	21.75	23.10	21.22	21.31	22.06	21.07	21.06
126	21.22	21.36	21.84	23.00	21.74	23.07	21.22	21.30	21.92	21.10	21.09
144	21.32	21.40	21.81	22.96	21.69	23.03	21.15	21.22	21.77	21.04	21.04
162	21.14	21.26	21.62	22.95	21.67	23.01	21.14	21.19	21.64	21.03	21.03
180	21.22	21.37	21.84	23.06	21.72	23.07	21.17	21.22	21.63	21.08	21.07

Table G.4: Mean SMAPE results for AR(3) DGP MS-Het scenario

Length of Series	RNN(10)	RNN(3)	FFNN(10)	FFNN(3)	LGBM(10)	LGBM(3)	AR(2)	AR(3)	AR(10)	PR(10)	PR(3)
18	-	-	-	-	-	-	21.72	22.51	36.98	24.40	22.59
36	22.02	22.19	22.70	23.49	22.39	23.59	20.60	20.89	23.77	22.19	21.98
54	21.94	22.10	22.21	23.37	22.19	23.49	20.28	20.44	22.12	21.96	21.88
72	21.01	21.83	21.99	23.52	22.02	23.34	20.14	20.25	21.39	21.78	21.75
90	20.89	21.90	22.39	23.30	21.90	23.28	20.00	20.08	20.93	21.70	21.68
108	20.84	21.83	22.04	23.27	21.92	23.31	20.00	20.06	20.76	21.74	21.73
126	21.11	21.83	21.88	23.42	21.89	23.26	19.95	19.98	20.58	21.71	21.70
144	21.02	21.00	22.08	23.28	21.81	23.25	19.91	19.93	20.44	21.65	21.64
162	20.71	21.89	21.94	23.39	21.77	23.18	19.90	19.92	20.37	21.64	21.64
180	20.88	21.78	22.21	23.35	21.85	23.29	19.93	19.95	20.34	21.72	21.72

Appendix G.1.2. SAR(1) DGP

AR(12) is a superclass of the true DGP SAR(1), where only the 12th coefficient needs to be significant, which can be learned with sufficient length in the series. In the MS-Hom-Short scenario, the AR(3) model remains the worst performing model. However, the AR(12) has also become the second worst performing model in the MS-Hom-Short setting, due to the heavy complexity of the AR(12) model compared to the SAR(1) model and the short lengths of the series (24 points). The model closest to the true DGP which is SAR(1) shows the same performance as the RNN(12) and is the second best model. The FFNN(12) outperforms the SAR(1) model which is the true DGP, although the short lengths of the series in the MS-Hom-Short setting are adequate to learn the one coefficient in the SAR(1) model. This shows the competence of the models that learn across series in such homogeneous contexts. The PR(12) model in the MS-Hom-Short scenario closely follows the SAR(1) and RNN(12) models.

Further to the results discussed in the paper, in the MS-Hom-Long scenario of the SAR(1) DGP, the PR(12) model is almost the same as the SAR(1) model with the FFNN being the next following the PR(12). All three of them are equivalent in performance with respect to the Mean MASE values. It is evident that the unnecessary complexity of the models lead to poor performance, since the underlying DGP is an SAR(1) with just one coefficient. For example, the AR(12) model which is built per every series is a quite complex model with 12 coefficients for every individual series, with just one of them being actually necessary to model the series perfectly. Similarly, the RNN(12), although built across series, still holds inherent complexity in the model. This explains their poor performance on the MS-Hom-Long scenario. The PR(12) on the other hand, although it has 12 coefficients, has just sufficient complexity with its linear modelling to perform competitively. With the AR(3), it is apparent how

too much simplicity can also result in impaired performance of the misspecified models.

The MS-Het scenarios results indicate that, although the RNN is a competitive model due to its complexity in the heterogeneous setting, under the SAR(1) DGP with just one coefficient for every series, the RNN's high complexity may be unnecessary. Surprisingly, the misspecified AR(3) model outperforms all the non-linear global models in the MS-Het scenario. Theoretically it is unlikely for AR(3) to outperform SAR(1) even with the heterogeneous series. However, this result is due to generating series in the MS-Het scenario by randomly picking a coefficient from U(-0.5, 0.5) to induce heterogeneity. Some of these series may end up getting non-significant 12th lags depending on how big the coefficient is. On the other hand in the MS-Hom-Long setting, the underlying SAR(1) model used has a coefficient of 0.85, which is chosen intentionally to create series with longer memory, and thus the inferior performance of AR(3) in that scenario.

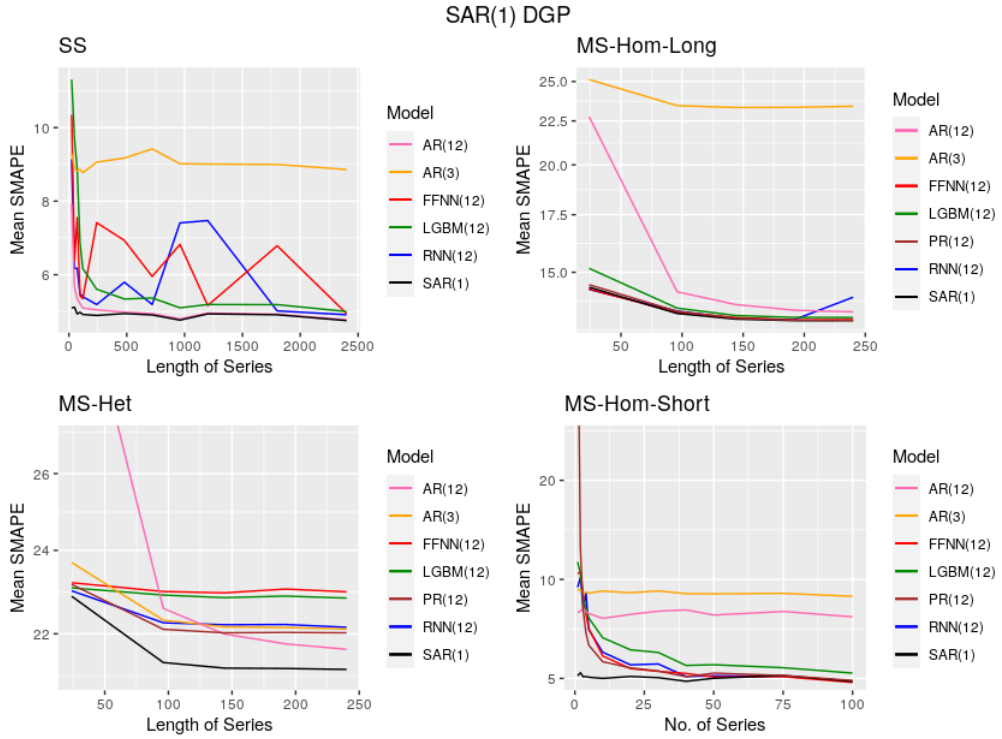

Figure G.1: Visualisation of the change of errors of the models across different amounts of data in the SAR(1) DGP scenarios.

Appendix G.1.3. Chaotic Logistic Map DGP

For the SS scenario, even though there is only one DGP with the same coefficients, the complex patterns produced by the DGP are not captured well by the linear model AR(15). The LGBM(15) is the best model on the SS scenario. On the MS-Hom-Short scenario, the RNN(15) is the best model with the FFNN(15) and the LGBM(15) being quite close. Once again, the lengths of the short series in the MS-Hom-Short scenario are not sufficient for the AR(15) model to properly learn all its 15 coefficients. In the MS-Hom-Long case, the LGBM(15) is the best. The drop of the errors in the AR model due to the length increase is notable. Despite their complexity, the lengths of the individual series in this case seem sufficient for the AR(15) to learn competitively. Although the PR(15) model outperforms the AR(15) in both the MS-Hom-Short and MS-Hom-Long scenarios, since the individual series hold complex patterns, the linear PR is still not sufficient to surpass the complex global models. In the MS-Het scenario, the SMAPE values of all the models have dropped substantially compared

Table G.1: Mean SMAPE results for SAR(1) DGP SS scenario

Length of Series	RNN(12)	FFNN(12)	LGBM(12)	AR(12)	AR(3)	SAR(1)
24	9.14	10.35	11.31	7.92	9.29	5.09
48	6.17	6.37	9.71	5.68	8.90	5.12
72	6.17	7.55	8.86	5.34	8.82	4.93
96	5.48	5.43	6.88	5.21	8.86	4.98
120	5.40	5.35	6.17	5.10	8.77	4.92
240	5.19	7.42	5.60	5.04	9.06	4.89
480	5.80	6.93	5.34	4.98	9.17	4.94
720	5.19	5.95	5.37	4.94	9.42	4.91
960	7.41	6.82	5.10	4.80	9.02	4.76
1,200	7.47	5.16	5.19	4.96	9.01	4.93
1,800	5.02	6.79	5.19	4.92	9.00	4.91
2,400	4.91	4.94	5.00	4.77	8.86	4.75

Table G.2: Mean SMAPE results for SAR(1) DGP MS-Hom-Short scenario

No. of Series	RNN(12)	FFNN(12)	LGBM(12)	AR(12)	AR(3)	SAR(1)	PR(12)
1	9.46	10.40	11.31	7.92	9.29	5.09	73.11
2	10.14	10.54	10.40	8.01	9.30	5.20	12.37
3	8.14	8.97	8.68	8.18	9.12	5.06	8.02
4	9.09	7.82	8.17	7.97	9.13	5.06	6.86
5	7.04	7.10	7.63	7.88	9.08	5.04	6.31
10	6.01	5.85	6.65	7.61	9.21	5.00	5.62
20	5.50	5.35	6.11	7.82	9.11	5.07	5.38
30	5.53	5.26	6.00	8.00	9.22	5.03	5.27
40	5.07	5.18	5.48	8.08	9.05	4.90	5.06
50	5.09	5.05	5.50	7.79	9.04	5.00	5.19
75	5.08	5.06	5.39	7.99	9.07	5.10	5.10
100	4.90	4.86	5.19	7.69	8.88	4.90	4.93

Table G.3: Mean SMAPE results for SAR(1) DGP MS-Hom-Long scenario

Length of Series	RNN(12)	FFNN(12)	LGBM(12)	AR(12)	AR(3)	SAR(1)	PR(12)
24	14.33	14.34	15.16	22.76	25.13	14.41	14.51
96	13.52	13.47	13.63	14.24	23.44	13.43	13.51
144	13.26	13.29	13.36	13.76	23.32	13.23	13.27
192	13.22	13.20	13.30	13.55	23.33	13.18	13.20
240	14.03	13.22	13.29	13.49	23.40	13.18	13.19

to all the other scenarios. This is due to the fact that in all the other homogeneous scenarios of this DGP, we specifically select the coefficients of the DGP such that the generated patterns are equally hard to be captured by all the models. However, in the heterogeneous scenario, since the coefficients are generated randomly for the different series, there is no such guarantee regarding the coefficients. Even though the patterns become relatively easy to be modelled likewise, still the non-linear global models outperform the two linear AR and PR models.

Table G.4: Mean SMAPE results for SAR(1) DGP MS-Het scenario

Length of Series	RNN(12)	FFNN(12)	LGBM(12)	AR(12)	AR(3)	SAR(1)	PR(12)
24	23.01	23.20	23.08	33.14	23.70	22.88	23.16
96	22.26	22.99	22.91	22.59	22.31	21.35	22.10
144	22.21	22.96	22.85	22.00	22.16	21.23	22.02
192	22.21	23.05	22.88	21.77	22.14	21.22	22.03
240	22.15	22.98	22.84	21.65	22.11	21.20	22.02

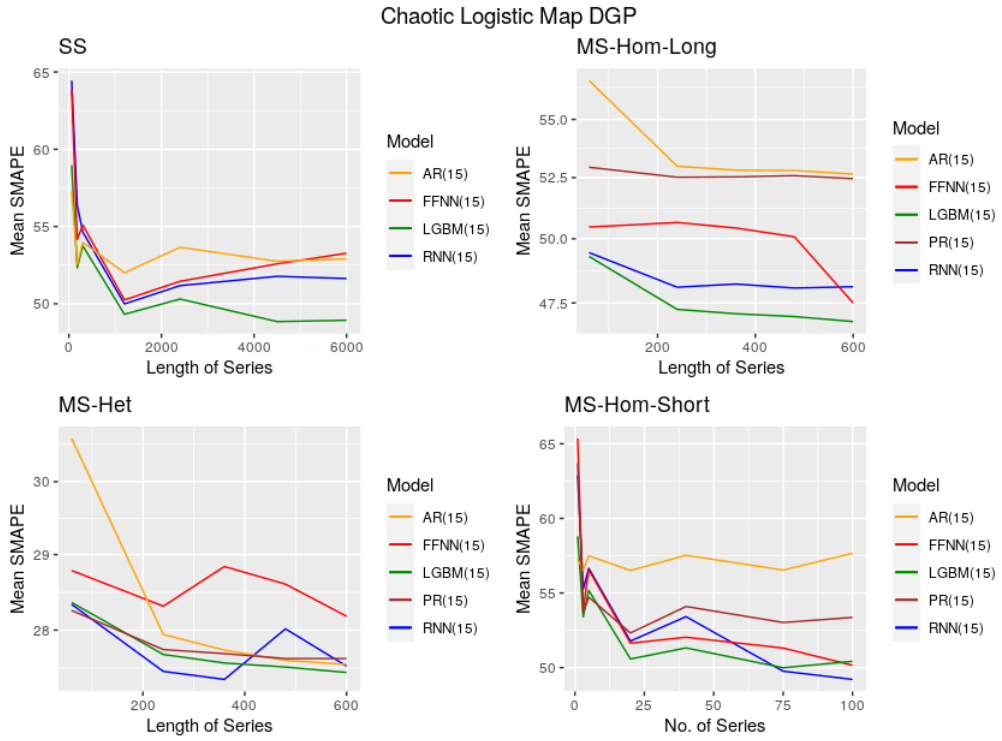

Figure G.2: Visualisation of the change of errors of the models across different amounts of data in the Chaotic Logistic Map DGP scenarios

Table G.1: Mean SMAPE results for Chaotic Logistic Map DGP SS scenario

Length of Series	RNN(15)	FFNN(15)	LGBM(15)	AR(15)
60	64.44	63.86	58.97	57.27
180	56.38	54.17	52.33	52.54
300	54.64	55.10	53.75	53.98
1,200	49.99	50.25	49.32	52.00
2,400	51.17	51.45	50.31	53.65
4,500	51.78	52.59	48.85	52.75
6,000	51.63	53.27	48.93	52.91

Appendix G.1.4. SETAR DGP

The non-linear models FFNN(15) and LGBM(15) outperform the linear AR(15) model in the SS scenario. However, RNN(15) is the worst model in the SS setting of the SETAR DGP. SETAR models

Table G.2: Mean SMAPE results for Chaotic Logistic Map DGP MS-Hom-Short scenario

No. of Series	RNN(15)	FFNN(15)	LGBM(15)	AR(15)	PR(15)
1	62.87	65.36	58.80	57.27	63.69
3	55.26	53.90	53.40	56.52	53.72
5	56.66	56.56	55.18	57.49	54.72
20	51.78	51.64	50.58	56.51	52.32
40	53.42	52.04	51.32	57.53	54.10
75	49.76	51.31	49.99	56.54	53.03
100	49.21	50.15	50.43	57.67	53.37

Table G.3: Mean SMAPE results for Chaotic Logistic Map DGP MS-Hom-Long scenario

Length of Series	RNN(15)	FFNN(15)	LGBM(15)	AR(15)	PR(15)
60	49.45	50.47	49.30	56.73	52.94
240	48.10	50.65	47.26	52.98	52.52
360	48.22	50.43	47.09	52.81	52.53
480	48.07	50.08	46.99	52.80	52.58
600	48.12	47.50	46.80	52.65	52.46

Table G.4: Mean SMAPE results for Chaotic Logistic Map DGP MS-Het scenario

Length of Series	RNN(15)	FFNN(15)	LGBM(15)	AR(15)	PR(15)
60	28.34	28.79	28.36	30.61	28.26
240	27.47	28.31	27.68	27.94	27.75
360	27.37	28.84	27.58	27.74	27.70
480	28.02	28.60	27.52	27.61	27.63
600	27.54	28.18	27.45	27.56	27.63

are regime switching models meaning that the coefficients of the underlying AR model changes from time to time even within the same series. This is the origin of the non-linearity in the SETAR DGPs. On the other hand, the nature of the RNN models is that they have a long-term memory especially with the LSTM cells on such long series with 6000 points. Nevertheless, on a SETAR generated series, such long memories are not quite useful due to regime switching. Apart from the SS scenario, the SETAR DGP results are quite similar to the Chaotic Logistic Map DGP. For the MS-Hom-Short experiment, the RNN(15) is the best model followed by the FFNN(15) and the LGBM(15). The errors of the local AR and SETAR models increase due to short lengths of the series in the MS-Hom-Short scenario.

In the MS-Hom-Long scenario of the SETAR DGP, we fix the lengths of the series to 240, and the number of series to 100. Therefore, the MS-Hom-Long scenario in this case acts as an increased length version of the MS-Hom-Short scenario. In the MS-Hom-Long scenario, the AR(15) model seems to learn some patterns better than at least the SETAR model with the increased lengths. The same length of 240 is used for the MS-Het scenario too with a set of 100 series in one dataset. Although the true model is the best under the heterogeneous settings of the previous DGPs, it is not the case with the SETAR DGP. The SETAR model in the heterogeneous case is still the worst performing model due to the insufficient lengths of the series. We can see that in the MS-Hom-Long scenario, the difference of the percentage differences among the local and global models is smaller than in the MS-Het scenario. Furthermore, the percentage difference of the SETAR model has increased substantially from 8.15%

to 12.70% in terms of Mean SMAPE. This can be explained by the involvement of random coefficients in the MS-Het scenario which makes it difficult for all forecasting techniques to model the series in the MS-Het context in general.

Appendix G.1.5. Mackey-Glass DGP

The LGBM(15) model is remarkably better than all the other models under the SS, MS-Hom-Short and MS-Hom-Long scenarios. The complexity of the datasets in the MS-Het scenario is evident by the substantial increase of the SMAPE values in all the models.

Appendix G.1.6. Fourier Terms DGP

Due to its local modelling, the performance of the DHR-ARIMA model remains the same in both the MS-Hom-Long and MS-Het scenarios, being robust to the heterogeneity. The complex global model LGBM performs competitively in the MS-Hom-Long scenario with the RNN and the FFNN being the next best. However, the linear modelling capacity of the PR model seems insufficient in this case to model the multiple seasonalities. In the MS-Het scenario, the RNN seems to be even more affected by the heterogeneity of the seasonal patterns than the FFNN. The LGBM model is still relatively more robust to the heterogeneity. Nevertheless, all global models have become worse in performance in the heterogeneous scenario compared to the homogeneous case.

Appendix G.2. Overall Summary of Results

Even though for global models it does not make much difference whether the data remains on one long series or spread across multiple different series, due to the diversity of the experiments involved and the different complexities of the model architectures picked by the automated hyperparameter tuning technique applied for every individual scenario, there are some exceptions to this conclusion, such as the FFNN in the Mackey-Glass Equations DGP and the RNN in the SETAR DGP. The mean MASE values for the SS and the MS-Hom-Short scenarios are not comparable here, since the data points considered for the in-sample naïve error used in the denominator of the MASE metric differ in the two contexts. The competitiveness of complex non-linear GFMs in general is evident from Table 9 of the paper where LGBMs and RNNs perform best under many scenarios. Complex GFMs have the ability to generalise better over all the series in a heterogeneous dataset, while sacrificing a fraction of the accuracy for a relatively simpler subset of series.

Appendix G.3. Statistical Tests for Significance of Differences

This section contains the information of the Hochberg's post-hoc procedures conducted on each experimental scenario.

Table G.1: Results of Hochberg's post-hoc procedure for the AR(3) DGP SS scenario, by using the AR(3) model as the control method. The adjusted p-values indicate that the AR(2) model is not significantly worse from the AR(3) model in this case. All the other models perform significantly worse from the best method AR(3).

Method	p_{Hoch}
AR(3)	-
AR(2)	0.53
AR(10)	0.04
RNN(3)	6.67×10^{-8}
FFNN(3)	2.17×10^{-19}
LGBM(3)	3.39×10^{-27}

Table G.2: Results of Hochberg's post-hoc procedure for the AR(3) DGP MS-Het scenario, by using AR(2) model as the control method. The adjusted p-values indicate that the AR(3) model is not significantly worse from the AR(2) model in this case. All the other models perform highly significantly worse from the best method AR(2).

Method	p_{Hoch}
AR(2)	-
AR(3)	0.51
AR(10)	3.73×10^{-22}
PR(3)	$< 10^{-30}$
PR(10)	$< 10^{-30}$
RNN(3)	$< 10^{-30}$
RNN(10)	$< 10^{-30}$
FFNN(3)	$< 10^{-30}$
FFNN(10)	$< 10^{-30}$
LGBM(3)	$< 10^{-30}$
LGBM(10)	$< 10^{-30}$

Table G.3: Results of Hochberg's post-hoc procedure for the Chaotic Logistic Map DGP MS-Hom-Short scenario, by using RNN(15) model as the control method. The adjusted p-values indicate that all the other models are significantly worse than the RNN(15) model in this case.

Method	p_{Hoch}
RNN(15)	-
LGBM(15)	6.08×10^{-3}
FFNN(15)	6.08×10^{-3}
PR(15)	$< 10^{-30}$
AR(15)	$< 10^{-30}$

Table G.4: Results of Hochberg's post-hoc procedure for the Chaotic Logistic Map DGP MS-Hom-Long scenario, by using the LGBM(15) model as the control method. The adjusted p-values indicate that all the other models are significantly worse from the LGBM(15) model in this case.

Method	p_{Hoch}
LGBM(15)	-
FFNN(15)	$< 10^{-30}$
RNN(15)	$< 10^{-30}$
PR(15)	$< 10^{-30}$
AR(15)	$< 10^{-30}$

Table G.5: Results of Hochberg's post-hoc procedure for the Mackey-Glass DGP, SS scenario, by using the best method LGBM(15) as the control method. The adjusted p-values indicate that all the other models are significantly worse than the LGBM(15) model in this case.

Method	p_{Hoch}
LGBM(15)	-
RNN(15)	1.08×10^{-12}
FFNN(15)	1.36×10^{-30}
AR(15)	$< 10^{-30}$

Table G.6: Results of Hochberg's post-hoc procedure for the Mackey-Glass DGP, MS-Het scenario, by using the best method RNN(15) as the control method. The adjusted p-values indicate that all the other models are significantly worse than the RNN(15) model in this case.

Method	p_{Hoch}
RNN(15)	-
LGBM(15)	1.10×10^{-4}
AR(15)	1.48×10^{-16}
PR(15)	$< 10^{-30}$
FFNN(15)	$< 10^{-30}$

Table G.7: Results of Hochberg's post-hoc procedure for the Chaotic Logistic Map DGP, Cluster scenario, by using the best method LGBM(15)-Cluster as the control method. The adjusted p-values indicate that all the models, are significantly worse than the LGBM(15)-Cluster model in this case.

Method	p_{Hoch}
LGBM(15)-Cluster	-
RNN(15)-Cluster	6.75×10^{-4}
LGBM(15)	5.58×10^{-5}
FFNN(15)-Cluster	9.64×10^{-14}
RNN(15)	1.81×10^{-23}
FFNN(15)	$< 10^{-30}$
PR(15)-Cluster	$< 10^{-30}$
PR(15)	$< 10^{-30}$
AR(15)	$< 10^{-30}$

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/. software available from tensorflow.org.

Bandara, K., Bergmeir, C., Smyl, S., 2020. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. Expert Systems with Applications 140, 112896.

Bandara, K., Shi, P., Bergmeir, C., Hewamalage, H., Tran, Q., Seaman, B., 2019. Sales demand forecast in e-commerce using a long short-term memory neural network methodology, in: Gedeon, T., Wong, K.W., Lee, M. (Eds.), Neural Information Processing, Springer International Publishing, Cham. pp. 462–474.

Ben Taieb, S., Bontempi, G., Atiya, A., Sorjamaa, A., 2012. A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. Expert Systems with Applications 39, 7067–7083. doi:10.1016/j.eswa.2012.01.039.

Bergmeir, C., Benítez, J.M., 2012. On the use of cross-validation for time series predictor evaluation. Information Sciences 191, 192 – 213. Data Mining for Software Trustworthiness.

Bergmeir, C., Costantini, M., Benítez, J.M., 2014. On the usefulness of cross-validation for directional forecast evaluation. Computational Statistics & Data Analysis 76, 132 – 143. CFEnetwork: The Annals of Computational and Financial Econometrics.

Bergmeir, C., Hyndman, R.J., Koo, B., 2018. A note on the validity of cross-validation for evaluating autoregressive time series prediction. Computational Statistics & Data Analysis 120, 70 – 83.

Brockwell, P.J., Davis, R.A., 1991. Time Series: Theory and Methods. Springer New York. doi:10.1007/978-1-4419-0320-4.

eResearch Centre., M., 2019. M3 user guide. URL: https://docs.massive.org.au/index.html.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA. p. 785–794. doi:10.1145/2939672.2939785.

Cryer, J., Chan, K., 2008. Time Series Analysis: With Applications in R. Springer Texts in Statistics, Springer.

Fabio Di Narzo, A., Luis Aznarte, J., Stigler, M., 2019. tsDyn: Nonlinear Time Series Models with Regime Switching. URL: https://CRAN.R-project.org/package=tsDyn. r package version 0.9-48.1.

Fischer, T., Krauss, C., Treichel, A., 2018. Machine learning for time series forecasting - a simulation study. FAU Discussion Papers in Economics 02/2018. Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics.

Gao, T., Jojic, V., 2016. Degrees of freedom in deep neural networks, in: Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, AUAI Press, Arlington, Virginia, USA. pp. 232–241.

Gers, F.A., Schraudolph, N.N., Schmidhuber, J., 2003. Learning precise timing with lstm recurrent networks. J. Mach. Learn. Res. 3, 115–143.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. doi:10.1109/CVPR.2016.90.

Hewamalage, H., Bergmeir, C., Bandara, K., 2020. Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting .

Hutter, F., Hoos, H.H., Leyton-Brown, K., 2011. Sequential model-based optimization for general

algorithm configuration, in: Coello, C.A.C. (Ed.), Learning and Intelligent Optimization, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 507–523.
Hyndman, R., Athanasopoulos, G., Bergmeir, C., Caceres, G., Chhay, L., O'Hara-Wild, M., Petropoulos, F., Razbash, S., Wang, E., Yasmeen, F., 2020. forecast: Forecasting functions for time series and linear models. URL: http://pkg.robjhyndman.com/forecast. r package version 8.11.
Hyndman, R., Khandakar, Y., 2008. Automatic time series forecasting: The forecast package for R. Journal of Statistical Software, Articles 27, 1–22.
Hyndman, R.J., Athanasopoulos, G., 2018. Forecasting: Principles and Practice. second ed., OTexts. URL: https://otexts.com/fpp2/.
Hyndman, R.J., Koehler, A.B., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679 – 688.
Hyndman, R.J., Koehler, A.B., Snyder, R.D., Grose, S., 2002. A state space framework for automatic forecasting using exponential smoothing methods. International Journal of Forecasting 18, 439 – 454.
Kang, Y., Li, F., Hyndman, R.J., O'Hara-Wild, M., Zhao, B., 2020. gratis: GeneRAting TIme Series with diverse and controllable characteristics. URL: https://CRAN.R-project.org/package=gratis. r package version 0.2-1.
Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y., 2017. Lightgbm: A highly efficient gradient boosting decision tree, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 3146–3154.
Kingma, D.P., Ba, J., 2015. Adam: A method for stochastic optimization, in: 3rd International Conference for Learning Representations.
Kunst, R.M., 2008. Cross validation of prediction models for seasonal time series by parametric bootstrapping. Austrian Journal of Statistics 37, 271–284.
Lau, K., Wu, Q., 2008. Local prediction of non-linear time series using support vector regression. Pattern Recognition 41, 1539 – 1547.
Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F., 2017. Smac v3: Algorithm configuration in python. URL: https://github.com/automl/SMAC3.
Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2020. The m5 accuracy competition: Results, findings and conclusions. URL: https://www.researchgate.net/publication/344487258_The_M5_Accuracy_competition_Results_findings_and_conclusions.
Mannattil, M., 2017. nolitsa. https://github.com/manu-mannattil/nolitsa.
May, R.M., 1976. Simple mathematical models with very complicated dynamics. Nature 261, 459–467.
Orabona, F., 2017. cocob. URL: https://github.com/bremen79/cocob.
Orabona, F., Tommasi, T., 2017. Training deep networks without learning rates through coin betting, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., USA. pp. 2157–2167.
Petropoulos, F., Makridakis, S., Assimakopoulos, V., Nikolopoulos, K., 2014. 'horses for courses' in demand forecasting. European Journal of Operational Research 237, 152 – 163.
R Core Team, 2020. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.
Schäfer, A.M., Zimmermann, H.G., 2006. Recurrent neural networks are universal approximators, in: Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I, Springer-Verlag, Berlin, Heidelberg. pp. 632–640. doi:10.1007/11840817_66.
Sen, R., Yu, H.F., Dhillon, I.S., 2019. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting, in: Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 4837–4846.
Smith, A.K., 2019. CombMSC: Combined Model Selection Criteria. URL: https://CRAN.R-project.org/package=CombMSC. r package version 1.4.2.1.
Smyl, S., 2020. A hybrid method of exponential smoothing and recurrent neural networks for time

series forecasting. International Journal of Forecasting 36, 75 – 85. M4 Competition.
Smyl, S., Kuber, K., 2016a. Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks, in: 36th International Symposium on Forecasting.
Smyl, S., Kuber, K., 2016b. Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks, in: 36th International Symposium on Forecasting.
Suhartono, Amalia, F.F., Saputri, P.D., Rahayu, S.P., Ulama, B.S.S., 2018. Simulation study for determining the best architecture of multilayer perceptron for forecasting nonlinear seasonal time series. Journal of Physics: Conference Series 1028, 012214.
Sun, J., Yang, Y., Liu, Y., Chen, C., Rao, W., Bai, Y., 2019. Univariate time series classification using information geometry. Pattern Recognition 95, 24 – 35.
Svetunkov, I., 2019. smooth: Forecasting Using State Space Models. URL: https://CRAN.R-project.org/package=smooth. r package version 2.5.3.
Trapero, J.R., Kourentzes, N., Fildes, R., 2015. On the identification of sales forecasting models in the presence of promotions. Journal of the Operational Research Society 66, 299–307. doi:10.1057/jors.2013.174.
Vanli, N.D., Sayin, M.O., Mohaghegh N., M., Ozkan, H., Kozat, S.S., 2019. Nonlinear regression via incremental decision trees. Pattern Recognition 86, 1 – 13.
Wallace, C.S., Boulton, D.M., 1968. An Information Measure for Classification. The Computer Journal 11, 185–194. doi:10.1093/comjnl/11.2.185.
Wallace, C.S., Dowe, D.L., 1999. Minimum Message Length and Kolmogorov Complexity. The Computer Journal 42, 270–283. doi:10.1093/comjnl/42.4.270.
Wen, R., Torkkola, K., Narayanaswamy, B., Madeka, D., 2017. A multi-horizon quantile recurrent forecaster. URL: https://arxiv.org/abs/1711.11053.
Ye, R., Dai, Q., 2021. Implementing transfer learning across different datasets for time series forecasting. Pattern Recognition 109, 107617.
Zhang, G., Patuwo, B., Hu, M.Y., 2001. A simulation study of artificial neural networks for nonlinear time-series forecasting. Computers & Operations Research 28, 381 – 396.
Zhang, G.P., 2007. A neural network ensemble method with jittered training data for time series forecasting. Inf. Sci. 177, 5329–5346. doi:10.1016/j.ins.2007.06.015.
Zhang, G.P., Qi, M., 2005. Neural network forecasting for seasonal and trend time series. Eur. J. Oper. Res. 160, 501–514.
Zhang, H., Nieto, F.H., 2017. TAR: Bayesian Modeling of Autoregressive Threshold Time Series Models. URL: https://CRAN.R-project.org/package=TAR. r package version 1.0.
Zhao, J., Itti, L., 2018. shapedtw: Shape dynamic time warping. Pattern Recognition 74, 171 – 184.