Explainable Learning of Long-term Dependencies through Machine Learning

Explainable Learning of Long-term Dependencies through Machine Learning

BY

Fernando Martinez-Garcia, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

PH.D. IN COMPUTER SCIENCE

(Department of Computing and Software) Hamilton, Ontario, Canada

TITLE: Explainable Learning of Long-term Dependencies through Machine Learning

AUTHOR: Fernando Martinez-Garcia

M.Sc. (Computer Science)

Instituto Tecnologico de Estudios Superiores de Monterrey, Monterrey, Mexico

SUPERVISOR: Dr. Douglas G. Down

NUMBER OF PAGES: xviii, 146

Lay Abstract

Machine learning has made big advances and transformed industries, but challenges such as growing model sizes and diminishing interpretability have hindered their usage and reliability. This research aims to enhance machine learning models for time-series forecasting. It starts by showcasing an interpretable-by-design linear model and its effectiveness in solving a real-world industry-related problem by means of incorporating new data while dynamically forgetting old information. Then, to consider nonlinear time-series components, the study delves into improving the Long Short-Term Memory (LSTM) Neural Network by creating an extended version, named E-LSTM, able to better exploit nonlinear long-term dependencies, resulting in a model of similar size and improved performance. Finally, the Generalized Interpretable LSTM (GI-LSTM), a more general LSTM architecture with higher temporal connectivity and embedded interpretability, is introduced. This architecture is shown to offer a more holistic interpretation of learned long-term dependencies while outperforming the previous architectures, all while keeping a compact model size.

Abstract

Machine learning-based models have yielded remarkable results in a wide range of applications, revolutionizing industries over the last few decades. However, a variety of challenges from the technical point of view, such as the drastic increase in model size and complexity, have become a barrier for their portability and human interpretation. This work focuses on enhancing specific machine learning models used in the time-series forecasting domain.

The study begins by demonstrating the effectiveness of a simple and interpretable-by-design machine learning model in handling a real-world time-series industry-related problem. This model incorporates new data while dynamically forgetting previous information, thus promoting continuous learning and adaptability laying the groundwork for practical applications within industries where real-time interpretable adaptation is crucial.

Then, the well-established LSTM Neural Network, an advanced but less interpretable model able to learn long and more complex time dependencies, is modified to generate a model, named E-LSTM, with extended temporal connectivity to better capture long-term dependencies. Experimental results demonstrate improved performance with no significant increase in model size across various datasets, showcasing the potential to have balance between performance and model size.

Finally, a new LSTM architecture built upon the E-LSTM's increased temporal connectivity while embedded with interpretability is proposed, called Generalized Interpretable LSTM (GI-LSTM). This architecture is designed to offer a more holistic interpretation of its learned long-term dependencies, providing semi-local interpretability by offering insights into the detected relevance across time-series data. Furthermore, the GI-LSTM outperforms alternative models, generally produces smaller models, and shows that performance does not necessarily come at the cost of interpretability.

In loving memory of

my grandmother María de los Ángeles Paredes Jerez,

whose kindness, uncommon personality, and sense of humor

have positively influenced my life decisions contributing to my existence.

Acknowledgements

I would like to thank my supervisor Dr. Douglas Down for being patient and understanding with me across the most challenging time the Canadian society has faced in recent years, while remaining consistent in his guidance, providing detail-oriented editorial comments, supporting my research decisions, and going beyond his duty on several occasions.

I also want to thank Dr. Ghada Badawy and Dr. Souvik Pal for providing their support in the early stages of my research and for the opportunity to keep developing my skills through FYELABS.

My appreciation extends to my mother, who exemplified that a person can have a balanced life while doing what is right even when is far from easy. I draw inspiration from my grandfather's work ethic, and to this day, he maintains his diligent commitment to his work, a legacy I have embraced.

I am deeply grateful to my long-term friends for their acceptance and invaluable insights. Jesús Tamez-Duque for his lessons in stoicism and the pursuit of meaningful things, Ulises Tamez-Duque for his kindness, Juan Carlos San-Martín-Alcazar for motivating me to move forward, Miguel Romero-Medellín for exemplifying the essence of friendship, Eduardo Reyes-Álvarez for

being supportive and sharing his knowledge, Fernanda Zapata-Murrieta for sharing her calmness and positive attitude and Sofía Romero-Mandujano for listening to my life stories.

I also thank my new friends and colleagues for sharing with me their companionship and unique ways of living, Max Moore, Jake Swarts, Akshobhya Katte, Roxana Roshankhah, Hannah Krivic, Zahra Esmaeilnezhad, Saman Sami, Brendan Fallon, Zhaleh Rahimi, Alireza Ganjali, Maryam Motamedi, Mohammad Moshtagh, Jason Balaci, Kamel Zarei, Rohit Gupta, Shizu Shi, Ricardo López, Sofía Caraza, Karen Alexandra Acosta-Cavazos, Jason Jiang.

Finally, to my previous mentors and educators for their exceptional dedication and beyond-duty teachings: Rogelio Soto, Lorena Arana-Tirado, María Leifa Wong-Balboa, and Felix Arellano-Flores.

I am immensely grateful for the support, guidance, and inspiration I have received from these individuals throughout my academic journey.

I also express my gratitude to the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Mexican National Council for Science and Technology (CONACYT) for partly financing the research here presented.

Contents

List of Figures

List of Tables

Table I. High-level notation

Symbol	Description	Implication
$\boldsymbol{a}$	Bolded lower-case variable	Vector
$\boldsymbol{A}$	Bolded upper-case variable	Matrix
$\boldsymbol{a}(\boldsymbol{k})$	Variable depending on k	Discrete time dependence
$\boldsymbol{A}_{\boldsymbol{k_1}:\boldsymbol{k_2}}$	Variable with subscripts separated by ':'	Matrix of contiguous time dependent variables
$\boldsymbol{a}^{(\boldsymbol{i})}$	Superscript with parenthesis	Naming a variable while differentiating from exponents
$\boldsymbol{\mathcal{L}}$	Scripted L	Loss function
$[\boldsymbol{a}]_{\boldsymbol{i}}$	Variable within subscripted square brackets	ith element of the variable (assumed to be a vector)
$[\boldsymbol{A}]_{\boldsymbol{i_1},\boldsymbol{i_2}}$	Variable within double subscripted square brackets	Element in row i and column j of the variable (assumed to be a matrix).
$[\boldsymbol{A}]_{\boldsymbol{row_i}}$	Variable within row subscripted square brackets	Row i of the variable (assumed to be a matrix).
$\boldsymbol{D_B}(\boldsymbol{A})$	Derivative of $\boldsymbol{B}$ with respect to $\boldsymbol{A}$	Derivative of $\boldsymbol{B} \in \mathbb{R}^{p\times q}$ with respect to variable $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ [1, p. 194], where $\boldsymbol{B} = F(\boldsymbol{A})$, $\boldsymbol{F}$: $\mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^{p\times q}$, $\boldsymbol{D_B}(\boldsymbol{A}) \in \mathbb{R}^{mp\times nq}$ and $[\boldsymbol{D_B}(\boldsymbol{A})]_{m(j_1-1)+i_1,n(j_2-1)+i_2} = \frac{d[\boldsymbol{B}]_{j_1,j_2}}{d[\boldsymbol{A}]_{i_1,i_2}}$.
$\boldsymbol{Vec}(\boldsymbol{A})$	Vectorization of a matrix	A matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is transformed into a vector $\boldsymbol{a}_{vec} \in \mathbb{R}^{mn}$ with $[\boldsymbol{A}]_{i_1,i_2} = [\boldsymbol{a}_{vec}]_{i_1+(i_2-1)i_1}$

Chapter 1

Introduction

Machine learning (ML) models have produced remarkable results in a wide range of applications, aiding and revolutionizing fields and industries in the last couple of decades [2]-[5]. Despite these extraordinary results a variety of challenges have become a barrier for the application of these models, specifically their dramatic increase in size and reduced interpretability, understood as the ability to provide explanations in understandable terms to a human. Given the ever-increasing size of ML models [6]-[7], mostly driven by the resulting increase in performance, the ability to train these models in portable devices has been negatively affected, limiting their accessibility to the public. In addition, this increased size has also resulted in a reduction in the capability of humans to understand the patterns the models learn, since these patterns are often encoded in nonlinear relations, which represent the basis of many current advanced models.

Time-series forecasting, an area characterized by data containing temporal dependencies of different complexity levels, is among the areas that have been influenced by machine learning models, encompassing fields from finance to environmental science [3], [5]. However, this influence is still affected by the challenges previously mentioned, which limits their application in real-world scenarios in which decisions need to be made about health, well-being, and long-term planning, due to a lack of robustness and safety [8]-[10]. The low interpretability in ML models, typical in this area, can be partly attributed to the increasing complexity they face, promoted by

the search for better performance and the non-obvious dependence types a time-series is driven by.

The use of linear models (LM) to handle time-dependent systems is a well-studied area [11]-[17], that predates the current ML field, and it can be considered as part of the field's origins due to the models being data-driven, proving to be adapted for a variety of applications. Furthermore, these simple models have consistently shown the power to capture statistical correlations while allowing for an easy interpretation of the input-output dependencies they learn through their parameters/coefficients, since these explicitly indicate a level of relevance the linear model gives to the input data across time. Despite the difficulty for LMs to cope with nonlinearities when present in time-dependent systems, their low complexity allows for the online recomputation of their parameters, enabling adaptive strategies to be designed and producing so-called adaptive linear models (ALMs).

ALM functionalities have been widely employed as an option to address the challenge of modeling nonlinearities for time-varying systems across several applications with acceptable results [18]-[22]. This performance can be mostly attributed to the capability of the ALM to quickly overwrite/erase its stored/encoded information, linked to previous data, and generate linear relations based on the most up-to-date data. For time series in which linearization can occur due to the relatively slow dynamics of the system, ALMs become an option in terms of performance and interpretability. This is a reasonable approach for a number of industrial applications [19]-[20].

Neural Networks (NN), among the most popular and successful ML models [23]-[26], carry out a more direct approach when used for data containing nonlinear components, trying to model their effects through nonlinear functions. For the case of time series, Recurrent Neural

Networks (RNNs), a type of NN characterized by using recurrent connections to capture time dependencies has been the basis over which more advanced ML models have been developed to exploit time dependencies.

Among RNN models, the Long Short-Term Memory (LSTM) network, introduced at the end of the 1990s [27]-[29], has shown the ability to exploit long-term dependencies by producing competitive results in a diverse set of applications [30]-[34]. In comparison to LMs, the LSTM network mitigated the need to retrain a model due to nonlinear complex behaviour in the data and, when this is the case, tends to yield better performance by learning more intricate patterns. However, this gain in performance comes at the expense of losing interpretability in the model and greatly increasing the number of parameters.

Considering the previous ML challenges in the time-series domain, this research aims to promote the progress in the area by designing and implementing ways to enhance the performance, size, and interpretability of the models used, potentially leading to valuable insights on the extracted information of the data to improve decision-making and extending the limits of ML implementation. Furthermore, as the proposed models are progressively developed in this work, the ability to incorporate long-term information is increased while pushing towards improving or maintaining their interpretability.

The contributions provided in this work can be summarised as follows. First, an adaptive linear model able to forget previous information at a dynamic rate is proposed and implemented to regulate server temperatures in an industrial setting, producing competitive results with respect to standard control algorithms and producing significant energy-consumption savings. Also, an extension to the LSTM model is proposed, named Extended LSTM, which increases the LSTM internal temporal connectivity to better capture long-term dependencies. The proposed model

shows improved performance across a variety of datasets without significant size increase. Furthermore, a generalized LSTM architecture with higher temporal connectivity than the E-LSTM and featuring embedded interpretability, called GI-LSTM, is designed to exploit long-term dependencies more efficiently in terms of the number of parameters; resulting in a better-performing model which is easier to interpret, and is of similar/smaller size to the E-LSTM and the LSTM. Next, the organization of thesis and its contributions are described in detail.

In Chapter 2 an adaptive linear model, based on the Weighted Recursive Least Squares (WRLS) algorithm, is proposed to incorporate information about prior model errors more actively. This is accomplished by the introduction of a time-varying forgetting factor $\lambda(k)$, an approach explored in [35]-[38] that has been shown to allow for a more precise regulation of how much past information should be forgotten to sufficiently adapt to the system. The resulting ALM, named Variable WRLS (VWRLS), can adapt with a dynamic rate, depending on user-defined physically-interpretable thresholds, and is constrained by design to keep a user-defined fraction of previous information to mitigate online overfitting. The proposed VWRLS is used to design an Adaptive Predictive Controller (APC), based on the General Predictive Controller (GPC) approach, which is implemented on a real rack-mounted cooling unit to control server temperatures in data centres. The designed APC outperforms both standard control algorithms in simulated experiments and when implemented in a real system.

Chapter 3 functions as a review and a bridge between Chapter 2 and Chapter 4, in which NN models intended for time series are presented, emphasizing the LSTM architecture. The backpropagation algorithm, the core of parameter tuning in NN models, is concisely introduced. Also, capabilities and limitations of NNs for time series are expressed in order to motivate the need for newer architectures.

In Chapter 4 the E-LSTM architecture is presented, serving as an initial step to overcome LSTM-specific limitations when identifying long-term dependencies. Also, the need for increased internal temporal connectivity, between distant and current cell states, is mathematically justified and experimentally corroborated by a performance comparison with alternative models. In addition, a selection process for the location of the increased connectivity is presented, based on the Distance Correlation measure. Experimental results show that in most cases, the E-LSTM model reduces the number of parameters needed to achieve similar or better performance to the LSTM, by an order of magnitude in some experiments.

Chapter 5 introduces a generalized LSTM architecture with embedded interpretability, Generalized Interpretable LSTM (GI-LSTM), which is built upon the higher temporal connectivity approach of the E-LSTM. This advanced and more complex LSTM network enables a semi-local interpretation [39], providing direct information about how much relevance it gives to parts of the time series, up to a user-defined maximum dependence, and removes the need for precisely locating the temporal connectivity. Despite the increased complexity, experimental comparative results show that the GI-LSTM results in a model with even better performance than the E-LSTM, the LSTM, and alternative models without significantly increasing the model size and producing comparatively better results for small model sizes; resulting in a model that performs better size-wise and is more accessible in interpretation.

Finally, Chapter 6 discusses the limitations of the current research and how these could be addressed from a practical point of view. It also explores realistic options for future work aligned with the aims established for this research and indicates alternative goals in the direction of dynamic connectivity to improve performance.

Chapter 2

An Adaptive Linear Model for Time Series with Control Applications

This chapter focuses on proposing a linear model intended for time-series modeling, directly interpretable by observing its parameters. The linear model is constructed based on the well-known Weighted Recursive Least Squares method, and a variable forgetting factor is proposed to regulate the speed at which information is forgotten, enabling adaptation to current trends in the data. The linear model is then instantiated in the context of control applications, specifically by following the General Predictive Controller approach, which is further modified by integrating a variable prediction horizon. This instantiation results in an Adaptive Predictive Controller capable of quickly adapting to changes in the system and able to accommodate potential nonlinearities. Comparative simulations are performed on the controller to validate its performance, and real experiments are carried out on a cooling system used in a real-world single-rack server system for industrial applications. A relevant part of the results and contributions presented in this chapter have been published in [40]. Here, we expand on that work, adding a number of useful details and insights.

2.1 Weighted Recursive Least Squares with Time-varying Forgetting Factor

2.1.1 Linear Models

The modeling of multi-input single-output (MISO) systems through linear models is a well-studied approach across several fields [11]-[17] due to its power to capture correlations and its ability to explicitly express input-output dependencies through its learnable parameters, an inherent and desirable feature of this type of models, as observed in (2.1.1)

$$\hat{y}(k) = \boldsymbol{\theta}\boldsymbol{x}(k). \tag{2.1.1}$$

Here, $\hat{y}(k) \in \mathbb{R}$ represents the output of the model; $\boldsymbol{x}(k) \in \mathbb{R}^n$ is the input data; and $\boldsymbol{\theta} \in \mathbb{R}^{1\times n}$ represents the learnable parameters of the model.

When sufficient data points are available, $k \geq n$, a matrix $\boldsymbol{\theta}$ can be computed so that it minimizes the Mean Square Error loss function $\mathcal{L}_{LM} = \sum_{j=0}^{k} e(j)^2$, with $e(j) = y(j) - \hat{y}(j)$. The result of this minimization is the well-known Ordinary Least Squares (OLS) regression in (2.1.2).

$$\boldsymbol{\theta}_{OLS} = (\boldsymbol{X}_{0:k}(\boldsymbol{X}_{0:k})^T)^{-1}\boldsymbol{X}_{0:k}(\boldsymbol{y}_{0:k})^T \tag{2.1.2}$$

with $\boldsymbol{X}_{0:k} = [\boldsymbol{x}(0), \dots, \boldsymbol{x}(k)]$, $\boldsymbol{y}_{0:k} = [y(0), \dots, y(k)]$.

Nevertheless, the OLS approach might produce overfitting to the data points in $\boldsymbol{X}_{0:k}$, making it susceptible to causing larger than acceptable errors when new data points are presented. One way to mitigate the previous effect is by splitting the data into training and validation sets, $\boldsymbol{X}_{0:k_{train}}$ and $\boldsymbol{X}_{k_{train}:k}$ respectively, and using any of the family of Gradient Descent (GD) algorithms [41]-[43] to iteratively compute matrices $\boldsymbol{\theta}^{(i)}$ to progressively minimize the MSE of the training set, $\sum_{j=1}^{k_{train}} e(j)^2$. During the latter minimization, the MSE of the validation set, $\sum_{j=k_{train}+1}^{k} e(j)^2$, is tracked and used as a stopping criterion for the minimization. The previous

process is carried out using the standard GD algorithm as given in (2.1.3), with $\boldsymbol{e}_{0:k_{train}} = [e(0), \dots, e(k_{train})]$ and $\alpha \in \mathbb{R}^+$ being a positive scalar hyperparameter usually referred to as the learning rate.

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \boldsymbol{X}_{0:k_{train}} \left(\boldsymbol{e}_{0:k_{train}}\right)^T \tag{2.1.3}$$

When properly trained to avoid overfitting, LMs can generate reasonable performance for a wide variety of applications [44]-[45]; however, an important limitation might arise when they are implemented for more complex systems: the difficulty for LMs to cope with nonlinearities when present in systems. For instance, if during a time interval $[k_{j_1}, k_{j_2}]$ the system remained withing a subspace characterized by a high degree of nonlinear behavior, an LM, as expressed in (2.1.3), could experience low performance, i.e., larger than acceptable errors. In other words, since the loss function $\mathcal{L}_{LM}$ gives the same relevance to all quadratic errors the significance of such subspaces is not highlighted; furthermore, assigning more weight to errors of data points from this type of subspace becomes a non-trivial memory.

2.1.2 Weighted Recursive Least Squares

Weighted Recursive Least Squares (WRLS), an Adaptive Linear Model (ALM), has been employed as an option to address the challenge of subspaces with nonlinearities across several applications, showing acceptable results [18]-[22]. In general, the approach followed by an ALM consists of making an online update, $\Delta\boldsymbol{\theta}(k) \in \mathbb{R}^{1\times n}$, to the learnable parameters at each (discrete) time instant in order to give more relevance to the newer values whenever necessary, promoting a linearization with more focus in the current subspaces. The latter process generates a time-varying model described in (2.1.4)-(2.1.5)

$$\hat{y}(k) = \boldsymbol{\theta}(k)\boldsymbol{x}(k) \tag{2.1.4}$$

$$\boldsymbol{\theta}(k) = \boldsymbol{\theta}(k-1) + \alpha(k)\Delta\boldsymbol{\theta}(k) \tag{2.1.5}$$

where $\alpha(k) \in \mathbb{R}^+$ is the learning rate (possibly time-varying) which regulates the influence of the update $\Delta\boldsymbol{\theta}(k)$ into the time-varying learnable parameters $\boldsymbol{\theta}(k)$.

In the specific case of the standard WRLS algorithm the update $\Delta\boldsymbol{\theta}(k)$ incorporates real-time information through the minimization of a time-varying loss function $\mathcal{L}_{WRLS}(k)$ defined by the recurrence relation in (2.1.6) and explicitly defined by (2.1.7)

$$\mathcal{L}_{WRLS}(k) = e(k|k-1)^2 + \lambda\mathcal{L}_{WRLS}(k-1) \tag{2.1.6}$$

$$\mathcal{L}_{WRLS}(k) = \sum_{j=0}^{k-1} \lambda^j e(k-j|k-1-j)^2 \tag{2.1.7}$$

where $\lambda \in (0,1]$ is a hyperparameter known as the forgetting factor that defines how much the relevance of previous datapoints will be reduced (often fixed and/or computed based on preprocessed data), and $e(k-j|k-1-j) = y(k-j) - \boldsymbol{\theta}(k-j-1)\boldsymbol{x}(k-j)\ \forall j \geq 0$ is a prior error linked to the minimum value of the loss function, $\mathcal{L}_{WRLS}(k-1-j)$, at a prior instant.

Similar to OLS the standard WRLS approach has an explicit recursive solution described by (2.1.8)-(2.1.10) which promotes an online linearization of the system within the current subspace

$$\boldsymbol{\theta}_{WRLS}(k) = \boldsymbol{\theta}_{WRLS}(k-1) + e(k|k-1)\boldsymbol{b}(k) \tag{2.1.8}$$

$$\boldsymbol{b}(k) = \frac{\boldsymbol{P}(k-1)\boldsymbol{x}(k)}{\lambda + \boldsymbol{x}^T(k)\boldsymbol{P}(k-1)\boldsymbol{x}(k)} \tag{2.1.9}$$

$$\boldsymbol{P}(k) = \frac{\boldsymbol{P}(k-1) - \boldsymbol{b}(k)\boldsymbol{x}^T(k)\boldsymbol{P}(k-1)}{\lambda} \tag{2.1.10}$$

where $\boldsymbol{P}(k) \in \mathbb{R}^{n\times n}$ is the inverse of a weighted sample-covariance matrix centered around $\mathbf{0} \in \mathbb{R}^n$ and $\boldsymbol{b}(k) \in \mathbb{R}^n$ is the gradient direction, pointing away from the global minimum when

$e(k|k-1) > 0$ or towards it when $e(k|k-1) < 0$, with respect to the current parameter values, $\boldsymbol{\theta}_{WRLS}(k-1)$.

When information is available before starting the iterative process the values of $\boldsymbol{\theta}_{WRLS}(0)$ and $\boldsymbol{P}(0)$ can be computed using an OLS approach. Otherwise, they can be initialized as $\boldsymbol{\theta}_{WRLS}(0) = \mathbf{0}$ and $\boldsymbol{P}(0) = \rho \boldsymbol{I}_{n\times n}$, *where* $\rho \geq 1$ is a scalar value and $\boldsymbol{I}_{n\times n}$ is the identity matrix.

2.1.3 Variable forgetting factor Weighted Recursive Least Squares

One of the relevant aspects associated with the capability of the standard WRLS approach to adapt to newer values resides in the constant forgetting factor, λ, which promotes the relevance of newer data points by exponentially decreasing the relevance of previous values at a constant rate. Such an approach, although being a function of prior errors, $\{e(k|k-1), e(k-1|k-2), \dots, e(1|0)\}$, is not able to adjust the need to forget since a constant λ is used, only producing a reactive influence on the magnitude of the gradient $\boldsymbol{b}(k)$ through the current prior error $e(k|k-1)$.

In order to more actively incorporate information about prior errors in the WRLS approach beyond the most recent value, a time-varying forgetting factor $\lambda(k)$, with its respective loss function $\mathcal{L}_{VWRLS}(k)$, is proposed to generate a Variable WRLS (VWRLS) approach. This approach has been explored in [36]-[38] where it has been shown to allow for a more precise regulation of how much past information, in terms of prior errors $e(k-j|k-1-j)$, is appropriate to forget/introduce in online implementations; this can be interpreted as how much the loss function should be changed to adapt to the current subspace. Based on this and motivated by the how-much-information-to-forget approach, the proposed time-varying forgetting factor $\lambda(k)$ is designed as a

function of user-defined physically-interpretable thresholds, $\{e_{min}, e_{max}, \Delta\tau_{min}, A_{\Delta\tau}, p_{\text{old}}\}$ to facilitate implementation.

The previously defined thresholds are: a minimum time-window length, $\Delta\tau_{min} \in \mathbb{R}^+$, of most-recent previous information; a fixed interval, $[e_{min}, e_{max}] \in \mathbb{R}^+$, defining the minimum and maximum absolute values for the most-recent prior error, $e(k|k-1)$, that will influence $\lambda(k)$; a minimum old-information fraction, $p_{\text{old}} \in (0,1)$, expressing the minimum influence the oldest information will have in the adaptation of the forgetting factor when $k \to \infty$; and a multiplier, $A_{\Delta\tau} \geq 1$, defining how much the minimum time-window length can be extended.

The derivation of these thresholds starts by analyzing the loss function of the WRLS approach, $\mathcal{L}_{WRLS}(k)$. First, notice that in (2.1.7) the factor λ^{j-1} can be interpreted as the weight assigned to $e(k-j-1|k-j-2)^2$; therefore, when $k \to \infty$ the most recent $\Delta\tau$ seconds of information, equivalent to the first j terms using a sampling period $T_{sampling}$, have a weight of $\left(1-\lambda^j\right)/(1-\lambda)$ and the remaining terms (oldest) have a weight of $\lambda^j/(1-\lambda)$; in relative proportions (fractions), these weights would be $1-\lambda^j$ and λ^j, respectively.

From the previous realization, it will be our aim when defining $\lambda(k)$ to create the relation: $\lambda_{min}^{j_{min}} = p_{\text{old}}$, with $\lambda_{min} > 0$ as the minimum value of $\lambda(k)$ and $j_{min} = \left\lceil \frac{\Delta\tau_{min}}{T_{sampling}} \right\rceil$. To create the previous relation a variable time-window length, $\Delta\tau(k)$, with values in the interval $[\Delta\tau_{min}, A_{\Delta\tau}\Delta\tau_{min}]$ is defined as in (2.1.11)-(2.1.12)

$$\Delta\tau(k) = A_{\Delta\tau}{}^{\eta\left(e(k|k-1)\right)}\Delta\tau_{min} \tag{2.1.11}$$

$$\eta\left(e(k|k-1)\right) = min\left(1, max\left(0, \frac{|e(k|k-1)|-e_{min}}{e_{max}-e_{min}}\right)\right)^2 \tag{2.1.12}$$

where it is important to notice that $\left\lceil \frac{\Delta\tau(k)}{T_{sampling}} \right\rceil$ would be the index of the jth term in the context of the relative proportions $1-\lambda^j$ and λ^j.

From (2.1.11) a variable forgetting factor and its respective time-varying loss function, $\mathcal{L}_{VWRLS}(k)$, can be defined as

$$\lambda(k) = p_{\text{old}}^{\left\lfloor \frac{T_{sampling}}{\Delta\tau(k)} \right\rfloor} \tag{2.1.13}$$

$$\mathcal{L}_{VWRLS}(k) = e(k|k-1)^2 + \lambda(k)\mathcal{L}_{VWRLS}(k-1) \tag{2.1.14}$$

where the minimum value of $\lambda(k)$ can be calculated as $\lambda_{min} = p_{\text{old}}^{\left\lfloor \frac{T_{sampling}}{\Delta\tau_{min}} \right\rfloor}$, equivalent to $p_{\text{old}}^{\frac{1}{j_{min}}}$ and consequently producing the desired relation. Also, except for the use of $\lambda(k)$, the equations describing the VWRLS model's parameters, $\boldsymbol{M}_{VWRLS}(k)$, remain the same as in (2.1.8)-(2.1.10).

From (2.1.11)-(2.1.13) it can be observed that if $e(k|k-1) \geq e_{max}$ then $\lambda(k)$ will be equal to λ_{min}. Similarly, if $e(k|k-1) \leq e_{min}$ then $\lambda(k) \approx 1$, as long as $A_{\Delta\tau}\Delta\tau_{min}$ is large enough; for instance, if $A_{\Delta\tau}\Delta\tau_{min} > \frac{\ln(p_{old})}{\ln(0.99)} T_{sampling}$ then $\lambda(k) = 0.99$. Also, an increasing exponential-adaptation speed (derivative) of $\lambda(k)$ (2.1.15), with respect to $|e(k|k-1)|$, is obtained within the interval $[e_{min}, e_{max}]$.

$$D_{|e(k|k-1)|}\big(\lambda(k)\big) = \frac{\ln(p_{\text{old}}^{-2}) p_{\text{old}}^{\left\lfloor \frac{T_{sampling}}{\Delta\tau_{min}} \right\rfloor} \ln(A_{\Delta\tau})(|e(k|k-1)| - e_{min})}{\Delta\tau(k)(e_{max} - e_{min})^2} \tag{2.1.15}$$

As observed in (2.1.15), the adaptation speed increases exponentially near e_{max} which exponentially reduces both the value of $\lambda(k)$ and the relevance given to the least-recent prior errors, creating a desirable outcome since beyond e_{max} the model would produce a beyond-acceptable error. Furthermore, if at instant k we denote the accumulated weight caused by $\lambda(k)$ in

(2.1.14), up to the j most-recent prior errors by $S_{mr}(k)$ and the accumulated weight for the j least-recent by $S_{lr}(k)$, the property in (2.1.16) can be established.

$$\frac{S_{lr}(k)}{S_{mr}(k)+S_{lr}(k)} \geq p_{old}\frac{\left(1-\lambda_{min}^{k-j-1}\right)}{1-\lambda_{min}^{k}} \tag{2.1.16}$$

In more detail, since the first j terms in (2.1.14) depend on the time-varying time-window length $\Delta\tau(k)$ and $j \geq \left\lceil \frac{\Delta\tau_{min}}{T_{sampling}} \right\rceil$, then (2.1.16) implies $\frac{S_{lr}(k)}{S_{mr}(k)+S_{lr}(k)} \to p_{old}$ when $k \to \infty$. Consequently, over time the proposed time-varying forgetting factor will assign a normalized relevance of at least p_{old} to terms occurring after the $\left\lceil \frac{\Delta\tau(k)}{T_{sampling}} \right\rceil$ index in (2.1.14). Also, the derivation of (2.1.16) can be obtained using the relations shown in (2.1.17)-(2.1.19).

$$S_{mr}(k) = \left(1 + \lambda(k) + \lambda(k)\lambda(k-1) + \cdots + \lambda(k)\ldots\lambda(k-j+1)\right) \tag{2.1.17}$$

$$S_{lr}(k) = \prod_{i_1=0}^{j-1}\lambda(k-i_1)\left(\lambda(k-j) + \lambda(k-j)\lambda(k-j-1) + \cdots + \lambda(k-j)\lambda(k-j-1)\ldots\lambda(1)\right)$$

$$\geq \prod_{i_1=0}^{j-1}\lambda(k-i_1)\,\lambda_{min}\frac{1-\lambda_{min}^{k-j-1}}{1-\lambda_{min}} \tag{2.1.18}$$

$$\frac{S_{lr}(k)}{S_{mr}(k)+S_{lr}(k)} = \frac{1}{\frac{S_{mr}(k)}{S_{lr}(k)}+1} \geq \frac{1}{\left(\frac{1+\frac{1}{\lambda(k)}+\cdots+\frac{1}{\lambda(k)\lambda(k-1)\ldots\lambda(k-j+1)}}{\lambda_{min}\frac{1-\lambda_{min}^{k-j-1}}{1-\lambda_{min}}}\right)+1} \geq \frac{\lambda_{min}\frac{1-\lambda_{min}^{k-j-1}}{1-\lambda_{min}}}{1+\frac{1}{\lambda_{min}}+\cdots+\frac{1}{\lambda_{min}^{j-1}}+\lambda_{min}\frac{1-\lambda_{min}^{k-j-1}}{1-\lambda_{min}}}$$

$$= \frac{\lambda_{min}^{j}\frac{1-\lambda_{min}^{k-j-1}}{1-\lambda_{min}}}{\frac{1-\lambda_{min}^{k}}{1-\lambda_{min}}} \geq \lambda_{min}^{j_{min}}\frac{\left(1-\lambda_{min}^{k-j-1}\right)}{1-\lambda_{min}^{k}} = p_{old}\frac{\left(1-\lambda_{min}^{k-j-1}\right)}{1-\lambda_{min}^{k}} \tag{2.1.19}$$

One limitation of the proposed approach is $\lambda(k) < 1$ since $\Delta\tau(k)$ is upper bounded, in other words, the time-varying time-window cannot extend indefinitely. Consequently, to address this edge case whenever $e(k|k-1) < e_{min}$ the forgetting factor is set to 1, $\lambda(k) = 1$.

While using any ALM algorithm redundant information might be present when performing the adaptive process (2.1.8)-(2.1.10). Specifically, in the case of the previously described VWRLS, when $e(k|k-1) < e_{min}$, a negligible value for the prior error could occur but the iterative process would be performed regardless, potentially adding redundant information. Furthermore, the time-series signals might contain a level of noise due to the finite resolution in the acquiring devices used, possibly creating numerical instability in the form of overflow in the matrix $\boldsymbol{P}(k)$, which compresses the information of current and previous datapoints, $\boldsymbol{x}(k)$. Consequently, to promote computational stability a user-defined parameter representing the level of negligible error, e_{nl}, will be added so that the VWRLS algorithm is executed only when $e(k|k-1) > e_{nl}$; otherwise $\boldsymbol{\theta}_{VWRLS}(k) = \boldsymbol{\theta}_{VWRLS}(k-1)$ and $\boldsymbol{P}(k) = \boldsymbol{P}(k-1)$.

2.2. Adaptive-Linear-Model-based Control algorithm.

2.2.1 Autoregressive Exogenous model

In general, a time-series is often modeled by integrating a degree of autoregression to potentially extract time dependencies that show a level of regularity across time. The integration is performed by explicitly introducing it as part of the input data, $\boldsymbol{x}(k)$, or implicitly by using time-varying learnable parameters, $\boldsymbol{\theta}(k)$, as is the case for ALMs. A general formulation is defined by (2.2.1)

$$\hat{y}(k) = f_{model}\left(\boldsymbol{y}_{k-1:k-d_y}, \boldsymbol{x}_{ex}(k), \boldsymbol{\theta}(k)\right) \tag{2.2.1}$$

where $d_y \geq 1$ is the maximum lag for the output $y(k)$ and $\boldsymbol{x}_{ex}(k)$ denotes the fixed size vector composed of m exogenous variables at instant k and of previous values up to lag dependencies $\left\{d_{ex}^{(1)}, \ldots, d_{ex}^{(m)}\right\}$; with the input data vector defined as $\boldsymbol{x}(k) = [\boldsymbol{y}_{k-1:k-d}, \boldsymbol{x}_{ex}(k)^T]^T$.

When explicit integration of autoregression is used in the model, as in (2.2.1), the resulting model is deemed an Autoregressive Exogenous (ARX) model. An ARX model establishes a partly recursive relation, which enables a forecasting estimation of the time series' output by using the recursion over a prediction horizon, $\gamma \epsilon \mathbb{N}$, by performing forward iterations of γ steps. This approach can be implemented if $\boldsymbol{\theta}(k)$ is assumed to remain constant over the prediction horizon and as long as a sufficiently statistically confident estimation of the exogenous variable $\widehat{\boldsymbol{x}}_{ex}(k+j)$ can be generated in each of the iterations up to the chosen time-horizon value, i.e., $0 \leq j \leq \gamma$. A 1-step forward iteration of the previous approach is described in (2.2.2)

$$\begin{aligned}\hat{y}(k+1|k) &= f_{model}\left(\hat{y}(k), \boldsymbol{y}_{k-1:k-d_y+1}, \widehat{\boldsymbol{x}}_{ex}(k+1), \boldsymbol{\theta}(k)\right) \\ &= f_{model}\left(f_{model}\left(\boldsymbol{y}_{k-1:k-d_y}, \boldsymbol{x}_{ex}(k), \boldsymbol{\theta}(k)\right), \boldsymbol{y}_{k-1:k-d_y+1}, \widehat{\boldsymbol{x}}_{ex}(k+1), \boldsymbol{\theta}(k)\right) \\ &= f_{model}^{(1)}\left(\boldsymbol{y}_{k-1:k-d_y}, \boldsymbol{x}_{ex}(k), \widehat{\boldsymbol{x}}_{ex}(k+1), \boldsymbol{\theta}(k)\right) \qquad (2.2.2)\end{aligned}$$

with $f_{model}^{(1)}$ denoting the result of the 1-step forward iteration.

By generalizing the process in (2.2.2) to a time-horizon $\gamma \geq 1$ through γ-step forward iterations, using the previous assumption over $\boldsymbol{\theta}(k)$, $\hat{y}(k+\gamma|k)$ can be obtained as shown in (2.2.3) where its dependence to past and estimated information can be observed

$$\hat{y}(k+\gamma|k) = f_{model}^{(\gamma)}\left(\boldsymbol{y}_{k-1:k-d_y}, \boldsymbol{x}_{ex}(k), \widehat{\boldsymbol{x}}_{ex}(k+1), \ldots, \widehat{\boldsymbol{x}}_{ex}(k+\gamma), \boldsymbol{\theta}(k)\right) \qquad (2.2.3)$$

with $f_{model}^{(\gamma)}$ denoting the result of the γ-step forward iteration.

One of the useful properties of specializing LMs to ARX models is their practicality when implemented, since using the representation (2.2.3) shows that another LM can be used to explicitly define $\hat{y}(k+\gamma|k)$ as a function of $\boldsymbol{y}_{k-1:k-d}$, $\{\hat{\boldsymbol{x}}_{ex}(k+1), \dots, \hat{\boldsymbol{x}}_{ex}(k+\gamma)\}$, $\boldsymbol{x}_{ex}(k)$ and $\boldsymbol{\theta}(k)$, as described in (2.2.4)

$$\hat{y}(k+\gamma|k) = \boldsymbol{\theta}_{\gamma}(k)[\boldsymbol{y}_{k-1:k-d}, \boldsymbol{x}_{ex}(k)^T, \hat{\boldsymbol{x}}_{ex}(k+1)^T, \dots, \hat{\boldsymbol{x}}_{ex}(k+\gamma)^T]^T \tag{2.2.4}$$

with $\boldsymbol{\theta}_{\gamma}(k)$ denoting a matrix, which in practical terms defines $f_{model}^{(\gamma)}$ in this case, resulting from bounded-length-input process, $I_{\boldsymbol{M}}$, carried out γ times. In more detail, $I_{\boldsymbol{M}}$ depends on $\boldsymbol{\theta}(k)$ and previous computed matrices $\boldsymbol{\theta}_{\gamma-1}(k), \dots, \boldsymbol{\theta}_{\gamma-d_{\boldsymbol{M}}}(k)$, where $d_{\boldsymbol{\theta}} = min(\gamma, d_y)$.

2.2.2 Generalized-Predictive-Control algorithm

The model described in (2.2.4) is of special interest in the are of control theory when the exogenous variables are user-defined across the prediction horizon, i.e., they are manipulated variables. This has been explored in [46]-[47] resulting in the well-known Generalized Predictive Controller (GPC) algorithm, which has become one of the most popular predictive control algorithms with a wide variety of applications.

In the context of the GPC algorithm, a model (2.2.5) composed of m manipulated variables, $\{u_1(k), \dots, u_m(k)\}$, is used over a prediction horizon γ with a control horizon $\gamma_c \leq \gamma$. The output forecast $\hat{y}(k+j|k)$ made by the GPC for $j \leq \gamma$ is interchangeably expressed by (2.2.6) and (2.2.7)

$$\hat{y}(k) = \boldsymbol{a}_0 \boldsymbol{y}_{past}(k) + \boldsymbol{b}_0 \boldsymbol{u}_{past}(k) \tag{2.2.5}$$

$$\hat{\boldsymbol{y}}_{future}^{(\gamma)}(k) = \boldsymbol{A}_{\gamma} \boldsymbol{y}_{past}(k) + \boldsymbol{B}_{\gamma} \boldsymbol{u}_{past}(k) + \boldsymbol{H}_{\gamma} \boldsymbol{u}_{future}^{(\gamma_c)}(k) \tag{2.2.6}$$

$$\hat{\boldsymbol{y}}_{future}^{(\gamma)}(k) = \boldsymbol{A}_{\gamma}\boldsymbol{y}_{past}(k) + \boldsymbol{B}_{\gamma}\boldsymbol{u}_{past}(k) + \boldsymbol{H}'_{\gamma}\boldsymbol{u}(k-1) + \boldsymbol{G}_{\gamma}\Delta\boldsymbol{u}_{future}^{(\gamma_c)}(k) \tag{2.2.7}$$

with $[\boldsymbol{a}_0, \boldsymbol{b}_0] = \boldsymbol{\theta}(k)$,

$$\boldsymbol{A}_{\gamma} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \vdots \\ \boldsymbol{a}_{\gamma}^T \end{bmatrix}, \boldsymbol{B}_{\gamma} = \begin{bmatrix} \boldsymbol{b}_1^T \\ \vdots \\ \boldsymbol{b}_{\gamma}^T \end{bmatrix}, \boldsymbol{a}_j = I_A\left(\boldsymbol{a}_{j-1}, \dots, \boldsymbol{a}_{j-d_{\boldsymbol{A}}(j)}\right), \boldsymbol{b}_j = I_B\left(\boldsymbol{a}_{j-1}, \dots, \boldsymbol{a}_{j-d_{\boldsymbol{A}}(j)}, \boldsymbol{b}_{j-1}, \dots, \boldsymbol{b}_{j-d_{\boldsymbol{A}}(j)}\right)$$

$$\hat{\boldsymbol{y}}_{future}^{(\gamma)}(k) = \begin{bmatrix} \hat{y}(k+1|k) \\ \vdots \\ \hat{y}(k+\gamma|k) \end{bmatrix}, \boldsymbol{y}_{past}(k) = \begin{bmatrix} y(k-1) \\ \vdots \\ y(k-d) \end{bmatrix}, \boldsymbol{u}_{past}(k) = \begin{bmatrix} \boldsymbol{u}_{past-1}(k) \\ \vdots \\ \boldsymbol{u}_{past-m}(k) \end{bmatrix},$$

$$\boldsymbol{u}_{past-i}(k) = \begin{bmatrix} u_i(k-1) \\ \vdots \\ u_i(k-e_i) \end{bmatrix}, \boldsymbol{u}_{future}^{(\gamma_c)}(k) = \begin{bmatrix} \boldsymbol{u}(k) \\ \vdots \\ \boldsymbol{u}(k+\gamma_c-1) \end{bmatrix}, \boldsymbol{u}(k+j) = \begin{bmatrix} u_1(k+j) \\ \vdots \\ u_m(k+j) \end{bmatrix},$$

$$\boldsymbol{u}(k-1) = \begin{bmatrix} u_1(k-1) \\ \vdots \\ u_m(k-1) \end{bmatrix}, \quad \Delta\boldsymbol{u}_{future}^{(\gamma_c)}(k) = \begin{bmatrix} \Delta\boldsymbol{u}(k) \\ \vdots \\ \Delta\boldsymbol{u}(k+\gamma_c-1) \end{bmatrix}, \quad \Delta\boldsymbol{u}(k+j) = \begin{bmatrix} \Delta u_1(k+j) \\ \vdots \\ \Delta u_m(k+j) \end{bmatrix},$$

$\Delta u_i(k+j) = u_i(k+j) - u_i(k+j-1)$, $d_{\boldsymbol{A}}(j) = min\left(j, d_y\right)$, $\boldsymbol{A}_{\gamma} \in \mathbb{R}^{\gamma \times d_y}$, $\boldsymbol{B}_{\gamma} \in \mathbb{R}^{\gamma \times m\left(d_u^{(1)} + \cdots + d_u^{(m)}\right)}$, $\boldsymbol{H}_{\gamma} \in \mathbb{R}^{\gamma \times (m\gamma_c)}$, $\boldsymbol{G}_{\gamma} \in \mathbb{R}^{\gamma \times (m\gamma_c)}$ and $\boldsymbol{H}'_{\gamma} \in \mathbb{R}^{\gamma \times m}$.

In (2.2.6)-(2.2.7) the matrices $\boldsymbol{A}_{\gamma}$, $\boldsymbol{B}_{\gamma}$ are calculated through the previously mentioned iterative process, $I_{\boldsymbol{\theta}}$, which in the context of the GPC algorithm is separated into two processes, $I_{\boldsymbol{A}}$ and $I_{\boldsymbol{B}}$, defined in [75]. Also, the matrices $\boldsymbol{H}_{\gamma}$ are block lower triangular matrices as described in (2.2.8)-(2.2.9), where $\boldsymbol{h}_j, \boldsymbol{g}_j \in \mathbb{R}^{1 \times m}$. In more detail, $\boldsymbol{h}_1 = \boldsymbol{b}_0$ and for $j > 1$, $\boldsymbol{h}_j$ can be extracted from $\boldsymbol{B}_{\gamma}$ by taking the following m elements located in the ith row: $\left\{1, d_u^{(1)} + 1, \dots, d_u^{(1)} + \cdots + d_u^{(m-1)} + 1\right\}$. Furthermore, $\boldsymbol{g}_i = \sum_{i=1}^{j} \boldsymbol{h}_{\mathrm{i}}$ and $\boldsymbol{H}'_{\gamma} = \left[\boldsymbol{g}_1^T, \dots, \boldsymbol{g}_{\gamma}^T\right]^T$.

$$\boldsymbol{H}_{\gamma} = \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{h}_2 & \boldsymbol{h}_1 & \boldsymbol{0} & \ddots & \vdots \\ \boldsymbol{h}_3 & \boldsymbol{h}_2 & \boldsymbol{h}_1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{h}_{\gamma} & \boldsymbol{h}_{\gamma-1} & \boldsymbol{h}_{\gamma-2} & \cdots & \boldsymbol{h}_{\gamma-\gamma_c+1} \end{bmatrix} \tag{2.2.8}$$

$$\boldsymbol{G}_{\gamma} = \begin{bmatrix} \boldsymbol{g}_1 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{g}_2 & \boldsymbol{g}_1 & \boldsymbol{0} & \ddots & \vdots \\ \boldsymbol{g}_3 & \boldsymbol{g}_2 & \boldsymbol{g}_1 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{g}_{\gamma} & \boldsymbol{g}_{\gamma-1} & \boldsymbol{g}_{\gamma-2} & \cdots & \boldsymbol{g}_{\gamma-\gamma_c+1} \end{bmatrix} \tag{2.2.9}$$

In addition, the equivalences between (2.2.4) and some of the expressions linked to (2.2.6) are: $\boldsymbol{y}_{k-1:k-d} = \boldsymbol{y}_{past}(k)^T$, $\boldsymbol{x}_{ex}(k) = \boldsymbol{u}_{past}(k)$, $\hat{\boldsymbol{x}}_{ex}(k+j) = \boldsymbol{u}_{past}(k+j)$. Consequently, in the j-step forward iteration of the GPC algorithm elements in $\hat{\boldsymbol{x}}_{ex}(k+j)$ are either known or user-defined, where the latter type of elements is precisely $\boldsymbol{u}_{future}^{(j)}(k)$.

By using current and past information of the system the GPC structure can be used to determine appropriate increments of the exogenous values, $\Delta\boldsymbol{u}_{future}^{(\gamma_c)}(k)$, so that the model's output can closely track a vector of (desired) set points $\boldsymbol{y}_{desired}^{(\gamma)}(k) \in \mathbb{R}^{\gamma}$, i.e. $\hat{\boldsymbol{y}}_{future}^{(\gamma)}(k) \approx \boldsymbol{y}_{desired}^{(\gamma)}$. The exogenous values (manipulated variables) are computed by minimizing the loss function shown in (2.2.10) which has a closed-form solution (2.2.11) due to the linearity and the unconstrained structure of GPC.

$$\begin{aligned} \mathcal{L}_{GPC}(k) &= \sum_{j=1}^{\gamma} q_y^{(j)} e(k+j|k)^2 + \sum_{j=0}^{\gamma_c-1} \Delta\boldsymbol{u}(k+j)^T q_u^{(j)} \Delta\boldsymbol{u}(k+j) \\ &= \left(\boldsymbol{y}_{desired}^{(\gamma)}(k) - \hat{\boldsymbol{y}}_{future}^{(\gamma)}(k)\right)^T \boldsymbol{Q}_y \left(\boldsymbol{y}_{desired}^{(\gamma)}(k) - \hat{\boldsymbol{y}}_{future}^{(\gamma)}(k)\right) + \\ &\quad \Delta\boldsymbol{u}_{future}(k)^T \boldsymbol{Q}_u \Delta\boldsymbol{u}_{future}(k) \end{aligned} \tag{2.2.10}$$

$$\Delta \boldsymbol{u}_{future}(k) = \left(\boldsymbol{G}_{\gamma}^{T}\boldsymbol{Q}_{y}\boldsymbol{G}_{\gamma} + \boldsymbol{Q}_{u}\right)^{-1}\boldsymbol{G}_{\gamma}^{T}\boldsymbol{Q}_{y}(\boldsymbol{y}_{desired}^{(\gamma)}(k) - \boldsymbol{A}_{\gamma}\boldsymbol{y}_{past}(k) - \boldsymbol{B}_{\gamma}\boldsymbol{u}_{past}(k) - \boldsymbol{H}_{\gamma}'\boldsymbol{u}(k-1)) \quad (2.2.11)$$

where $e(k+j|k) = \left[\boldsymbol{y}_{desired}^{(\gamma)}(k)\right]_{j} - \hat{y}(k+j|k)$; $\boldsymbol{Q}_{y} \in \mathbb{R}^{\gamma\times\gamma}$ and $\boldsymbol{Q}_{u} \in \mathbb{R}^{m\gamma_c\times m\gamma_c}$ are positive semi-definite block diagonal matrices used to assign relevance to future estimated errors and penalize large changes in the manipulated variables.

2.2.3 GPC-based algorithm with variable prediction horizon and feasible solutions

The standard GPC algorithm considers a fixed prediction horizon γ; however, the chosen value for γ directly affects all the manipulated variables due to the interdependence created by the GPC solution (2.2.11). In this regard, a variable prediction horizon could potentially increase the flexibility of the approach and even consider practical constraints, an idea explored in [48]-[49] where theoretical advantages were demonstrated. Inspired by these works and aiming to achieve a higher flexibility with the prediction horizon, a simple and effective variable prediction horizon for the GPC is proposed and implemented heuristically.

First, it is important to highlight that in practical implementations, like most predictive-based algorithms, the GPC algorithm uses a receding-horizon approach, meaning that at each sampling instant the vector $\Delta \boldsymbol{u}_{future}(k)$ is recomputed. Furthermore, the computational cost of generating $\Delta \boldsymbol{u}_{future}(k)$ in (2.2.11) is mostly caused by $\left(\boldsymbol{G}_{\gamma}^{T}\boldsymbol{Q}_{u}\boldsymbol{G}_{\gamma} + \boldsymbol{Q}_{u}\right)^{-1}$, whose computational complexity is $O((m\gamma_c)^3)$ when using the Gauss-Jordan method or $O((m\gamma_c)^{2+\alpha})$, with $0.8 \leq \alpha \leq 0.81$, for more advanced and implementable methods. Hence, the total complexity of (2.2.11) is $O\left((m\gamma_c)^{2+\alpha} + \gamma(d + e_1 + \cdots + e_m + m + \gamma_c)\right)$, with $0.8 \leq \alpha \leq 1$.

Based on the previous complexity analysis and using the approach in [47], the control horizon γ_c can be set equal to 1 in order to decrease the computational cost of calculating the next value for $\Delta\boldsymbol{u}(k)$, producing an optimal "mean-level" controller [47]. In more detail, only a one-time increment is performed in the manipulated variables, after which no further changes are made, causing $\boldsymbol{u}(k+j)$ to remain constant across the prediction horizon, i.e., $\boldsymbol{u}(k+j) = \boldsymbol{u}(k),\ \forall j \geq 1$. Based on the previous setting and (2.2.7), the matrix $\boldsymbol{G}_\gamma$ can be replaced by $\left[\boldsymbol{g}_1^T \quad \cdots \quad \boldsymbol{g}_\gamma^T\right]^T$, significantly decreasing the computational complexity of (2.2.11) to $O\left(m^{2+\beta} + \gamma(d + e_1 + \cdots + e_m + m)\right)$. In general, for stable systems with possible dead-time, making $\gamma_c = 1$ can generate an acceptable solution since a new $\Delta\boldsymbol{u}(k)$ is computed in each iteration when the receding-horizon approach is used, a common and widely accepted approach for predictive control algorithms.

The general approach of the heuristic starts by setting all elements of matrix $\boldsymbol{Q}_y$ in (2.2.10) to zero except for the last element, set to 1, simplifying the optimization problem to (2.2.12) and leading to a corresponding explicit solution expressed in (2.2.13) with a time complexity of $O(d + e_1 + \cdots + e_m + m)$.

$$\boldsymbol{g}_\gamma \Delta\boldsymbol{u}(k) = \left[\boldsymbol{y}_{desired}^{(\gamma)}(k)\right]_\gamma - \boldsymbol{a}_\gamma \boldsymbol{y}_{past}(k) - \boldsymbol{b}_\gamma \boldsymbol{u}_{past}(k) - \boldsymbol{g}_\gamma \boldsymbol{u}(k-1) \tag{2.2.12}$$

$$\Delta\boldsymbol{u}(k) = \frac{\boldsymbol{g}_\gamma^T}{\boldsymbol{g}_\gamma \boldsymbol{g}_\gamma^T}\left(\left[\boldsymbol{y}_{desired}^{(\gamma)}(k)\right]_\gamma - \boldsymbol{a}_\gamma \boldsymbol{y}_{past}(k) - \boldsymbol{b}_\gamma \boldsymbol{u}_{past}(k) - \boldsymbol{g}_\gamma \boldsymbol{u}(k-1)\right) \tag{2.2.13}$$

As observed in (2.2.13) the calculation of $\Delta\boldsymbol{u}(k)$ is of low time complexity but considers only the last set point in the prediction horizon γ; however, such simplicity can be exploited by exploring possible contiguous values for the prediction horizon located in a user-defined set $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{n_{max}}\}$ which would lead to a solution set $\mho_\Delta = \{\Delta\boldsymbol{u}^{\gamma_1}(k), \Delta\boldsymbol{u}^{\gamma_2}(k), \ldots, \Delta\boldsymbol{u}^{\gamma_{n_{max}}}(k)\}$, representing the control increment vectors obtained at prediction horizon values ranging from γ_1

to $\gamma_{n_{max}}$. Using $\mathfrak{V}_\Delta$ manipulated-variables values, $\boldsymbol{u}^{(\gamma_i)}(k)$, can be created and subsequently used for a selection process based on practical constraints and predefined metrics.

The details of the previous heuristic are described next. Let us denote the set of physically feasible manipulated variables as $\mathcal{U}$. Then, assuming the variables are physically independent of each other and defined over continuous closed intervals (as in many control applications) the set $\mathcal{U}$ can be described by a hyperrectangle defined by the physical constraints, $(\boldsymbol{u}_{min}, \boldsymbol{u}_{max})$, of the manipulated variables; also, let τ_j denote $\left(\left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_j - \boldsymbol{a}_j \boldsymbol{y}_{past}(k) - \boldsymbol{b}_j \boldsymbol{u}_{past}(k) - \boldsymbol{g}_j \boldsymbol{u}(k-1)\right)$. From this point two cases will be considered, $m = 1$ and $m > 1$.

In the first case, with a single manipulated variable, the constraints $(\boldsymbol{u}_{min}, \boldsymbol{u}_{max})$ are applied on each of the single-element solutions, $\boldsymbol{u}^{(j)}(k)$, $i \in \Gamma$. This is performed by computing $\boldsymbol{u}_{sat}^{(j)} = SAT\left(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}\right)$, where $SAT(\cdot)$ is a function that saturates $\boldsymbol{u}^{(j)}(k)$ so that $\boldsymbol{u}_{min} \leq \boldsymbol{u}_{sat}^{(j)} \leq \boldsymbol{u}_{max}$, which becomes relevant when dealing with infeasible solutions due to the likely model mismatch. Then, a prediction horizon γ^*, and consequently a $\boldsymbol{u}_{sat}^{(\gamma^*)}$, is selected according to (2.2.14).

$$\gamma^* = \underset{j \in \Gamma}{\operatorname{argmin}} \left| \tau_j - \boldsymbol{g}_j \boldsymbol{u}_{sat}^{(j)} \right| \tag{2.2.14}$$

If there exists more than one element $j \in \Gamma$ that minimizes $\left|\tau_j - \boldsymbol{g}_j \boldsymbol{u}_{sat}^{(j)}\right|$, the minimum element is selected for γ^*. This selection is justified by the fact that smaller elements of Γ are more likely to result in smaller errors, since mismatches between $\boldsymbol{M}_{VWRLS}(k)$ and the optimal parameters $\boldsymbol{M}^*(k)$ as well as unmodeled effects are propagated to $\boldsymbol{A}_j$, $\boldsymbol{B}_j$, $\boldsymbol{H}_j$ and $\boldsymbol{G}_j$ through their iterative construction, which is the basis for computing τ_j and the set $\mathfrak{V}_\Delta$. Hence, the larger the

value of j, the greater the likelihood of a large error in the forecast, i.e., between $y(k+j)$ and $\hat{y}(k+j|k)$.

In the second case, $m > 1$, when using multiple manipulated variables and if any of the solutions obtained using the pseudo-inverse is not feasible, i.e., $\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}$, a solution might still exist. If the hyperplane defined by the normal vector and bias pair $(\boldsymbol{g}_j, \tau_j)$ intersects the hyperrectangle $\mathcal{U}$ it is possible to convert the vector $\boldsymbol{u}^{(j)}(k)$ into a vector $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \in \mathcal{U}$ with the property: $\boldsymbol{g}_j \boldsymbol{u}^{(j)}(k) = \boldsymbol{g}_j \boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$. Determining $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$ can be interpreted as finding a point in the hyperplane that is inside the boundaries of the hyperrectangle, as shown in Fig. 2.2.1.

Examining Fig. 2.2.1, it can be seen that $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$ can have multiple values, none or only one. When more than one value exists the closest value to $\boldsymbol{u}^{(j)}(k)$ is selected to achieve the smallest feasible change $\Delta\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$, in the 2-norm sense, with $\boldsymbol{g}_j \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) = 0$. In this way, feasible solutions are reduced to points in the intersection between the hyperplane $(\boldsymbol{g}_j, \tau_j)$ and $\mathcal{U}$; $\Delta\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$ would be the smallest vector from $\boldsymbol{u}^{(j)}(k)$ to one of the points in such intersection. The previous solution-finding approach is performed iteratively using the following procedure. First, the lower-dimensional hyperface closest to the intersection is determined by identifying in $\boldsymbol{u}^{(j)}(k)$ the set of components, $\mathcal{F}_1^c$, that cause it to be outside of $\mathcal{U}$ as specified in (2.2.15).

$$\mathcal{F}_1^c = \left\{ l \,|(1 \le l \le m) \wedge \left(\left[\boldsymbol{u}^{(j)}(k)\right]_l > [\boldsymbol{u}_{max}]_l \vee \left[\boldsymbol{u}^{(j)}(k)\right]_l < [\boldsymbol{u}_{min}]_l \right) \right\} \tag{2.2.15}$$

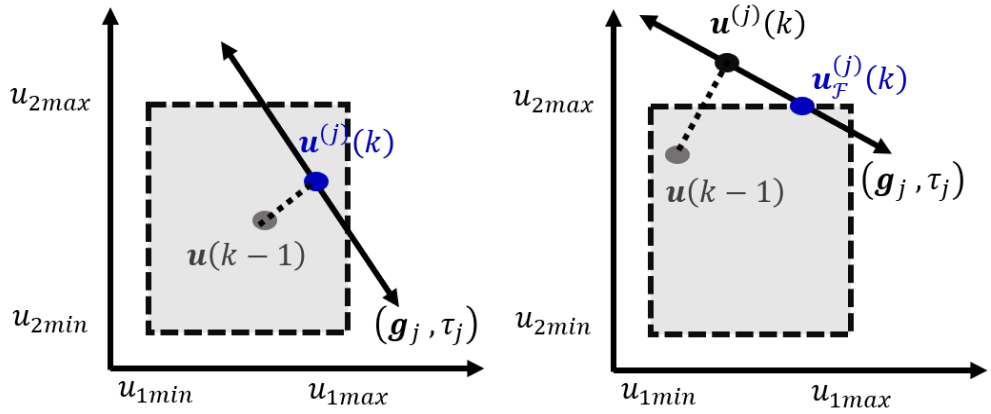


Fig. 2.2.1. Two-dimensional representation of the possible infinite set of solutions that arise in the proposed approach when $\boldsymbol{u}^{(j)}(k) \in \mathcal{U}$ (left) and $\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}$ (right).

Next, a hyperplane of dimension $m - card(\mathcal{F}_1^c)$ is defined by using the pair $\left(\breve{\boldsymbol{g}}_j^{(1)}, \breve{\tau}_j^{(1)}\right)$, where $\breve{\boldsymbol{g}}_j^{(1)}$ is the original vector $\boldsymbol{g}_j$ with components $l \in \mathcal{F}_1^c$ removed and $\breve{\tau}_j^{(1)} = \tau_j - \sum_{l \in \mathcal{F}_1^c} \left[\boldsymbol{u}_{sat}^{(j)}\right]_l [\boldsymbol{g}_j]_l$. Then, assuming at least one of the components of $\breve{\boldsymbol{g}}_j$ is non-zero, the vector $\Delta \boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k)$ that connects $\boldsymbol{u}^{(j)}(k)$ to the closest point in $\left(\breve{\boldsymbol{g}}_j^{(1)}, \breve{\tau}_j^{(1)}\right)$ is computed using (2.2.16) and (2.2.17).

$$\Delta \breve{\boldsymbol{u}}_{\mathcal{F}_1}^{(j)} = \frac{\breve{\boldsymbol{g}}_j^{(1)^T}}{\breve{\boldsymbol{g}}_j^{(1)} \breve{\boldsymbol{g}}_j^{(1)^T}} \breve{\tau}_j^{(1)} \tag{2.2.16}$$

$$\left[\Delta \boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k)\right]_l = \left[\Delta \breve{\boldsymbol{u}}_{\mathcal{F}_1}^{(j)}\right]_l, \forall l \notin \mathcal{F}_1^c, \quad \left[\Delta \boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k)\right]_l = \left[\boldsymbol{u}_{sat}^{j}\right]_l, \forall l \in \mathcal{F}_1^c \tag{2.2.17}$$

Finally, a modified solution $\boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k) = \boldsymbol{u}^{(j)}(k) + \Delta \boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k)$ can be calculated. If $\boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k) \notin \mathcal{U}$ then the steps (2.2.15)-(2.2.17) are repeated, but replacing $(\boldsymbol{g}_j, \tau_j)$ and $\boldsymbol{u}^{(j)}(k)$ by $\left(\breve{\boldsymbol{g}}_j^{(1)}, \breve{\tau}_j^{(1)}\right)$ and $\boldsymbol{u}_{\mathcal{F}_1}^{(j)}(k)$, respectively; in this way, the constraints already considered, $\left[\boldsymbol{u}_{sat}^{(j)}\right]_l \forall l \in \mathcal{F}_1^c$, are not violated, and new sets $\mathcal{F}_i^c$ are used in further iterations.

The iterative procedure ends after a number of iterations no higher than m, since at least one component is removed from $\boldsymbol{u}^{(j)}(k)$ in each iteration until a feasible solution $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$ is achieved, as depicted in Fig. 2.2.2. Also, it should be noted that in each iteration i, $\Delta\breve{\boldsymbol{u}}_{\mathcal{F}_i}^{(j)}$ is the smallest vector that fulfills $\Delta\breve{\boldsymbol{u}}_{\mathcal{F}_i}^{(j)}\breve{\boldsymbol{g}}_j^{(i)} = \breve{\tau}_j^{(i)}$ due to the right-hand side of (2.2.16) being the Moore–Penrose inverse, and there will be no need to consider previous constraints since increments $\Delta\breve{\boldsymbol{u}}_{\mathcal{F}_i}^{(j)}$ will be along the constrained gradient descent direction.

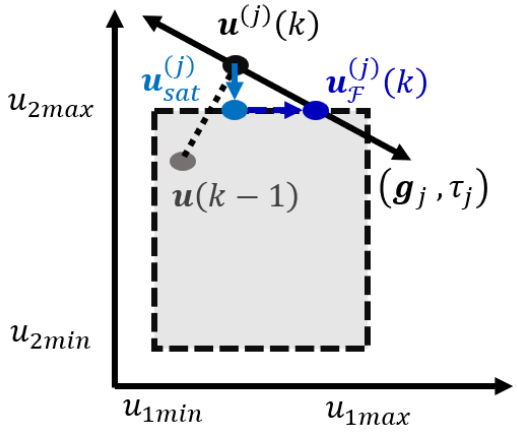


Fig. 2.2.2. Two-dimensional representation of the iterative process to generate $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$.

Once the iterative procedure is carried out for each possible value of j, the set of feasible prediction horizons is constructed $\Gamma_{\mathcal{F}} = \left\{ j | j \in \Gamma \wedge \exists \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) : \boldsymbol{u}_{max} \leq \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leq \boldsymbol{u}_{max} \right\}$, and from it the prediction horizon γ^* is selected using (2.2.18).

$$\gamma^* = \underset{j \in \Gamma_{\mathcal{F}}}{\operatorname{argmin}} \left| \tau_j - \boldsymbol{g}_j \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \right| \tag{2.2.18}$$

If there is no intersection between any hyperplane $(\boldsymbol{g}_j, \tau_j)$ and the hyperrectangle $\mathcal{U}$, implying infeasible inputs for any value in Γ, then γ^* is selected based on (2.2.14).

2.2.4 Adaptive Predictive Control Algorithm

An Adaptive Predictive Control (APC) algorithm is proposed by integrating the ALM model defined in Section 2.2.2 and the GPC-based control approach in Section 2.2.3, resulting in a control algorithm that provides online predictions of the system while adapting to new operating conditions. Potential advantages of APC algorithms have already been shown in a variety of applications [50]-[53] where promising results were shown through simulations.

The proposed APC is characterized by the following properties: it adapts its learning rate in an online fashion, which reflects the dynamics of the possibly time-varying nonlinear system; it uses a variable prediction horizon algorithm, increasing the controller's ability to deal with time-varying systems; it implements a method to transform physically infeasible solutions, in the predictive formulation, into feasible solutions (when they exist); and it is computationally inexpensive to implement with respect to the maximum prediction horizon and the dimensionality of the input. A graphical representation of the proposed APC can be observed in Fig. 2.2.3.

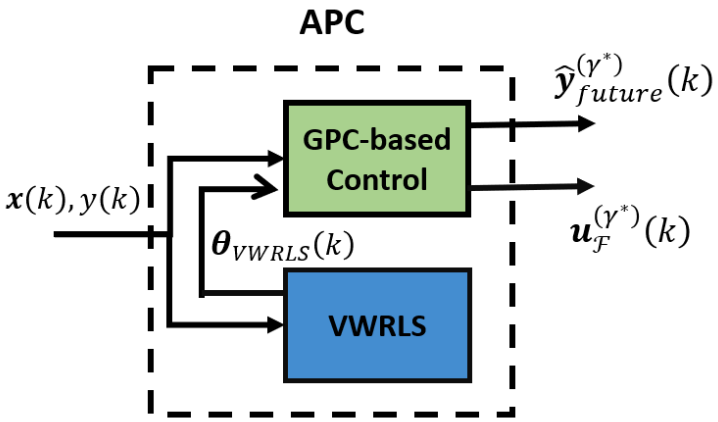


Fig. 2.2.3. High-level graphical representation of the proposed APC integrating the VWRLS and the GPC-Based control as the Adaptive and Predictive algorithms, respectively.

When implementing the VWRLS (or the WRLS) algorithm, in addition to constraining the minimum value of the forgetting factor through the choice of p_{old}, as discussed in Section 2.2.2, practical implementation issues must be addressed. In particular, the following potential interconnected issues are considered as most relevant: potential mismatches between the estimated and the optimal model parameters, lack of diversity in the data generated while in deployment, and numerical instability. These issues, combined or independent of each other, could slow down the convergence of $\boldsymbol{\theta}_{VWRLS}(k)$; degrading the performance of the control algorithm and leading to computational instability in the matrix $\boldsymbol{P}(k)$, as discussed in [54]. Next, two independent strategies are proposed to handle these implementation issues within the proposed APC framework.

The first strategy, intended to mitigate the first two issues, is based on the realization that, with respect to $\mathcal{L}_{VWRLS}(k)$, mismatches between the estimated and optimal parameters, $\boldsymbol{\theta}_{opt}(k)$, although not measured directly can be detected whenever there is a steady-state error, which can be defined over a possibly weighted time window considering the last k_{ss} errors. Therefore, by monitoring the steady-state error the following strategy is developed. First, if there is a solution $\boldsymbol{u}^{(j)}$ (or $\boldsymbol{u}_{\mathcal{F}}^{(j)}$) then, with probability one, there exist an infinite number of solutions, since a unique solution occurs when a vertex of the hyperrectangle $\mathcal{U}$ intersects the plane $(\boldsymbol{g}_j, \tau_j)$. Hence, to increase information diversity, it is proposed that one of these solutions be selected at random, promoting variability in some of the features in $\boldsymbol{x}_{ex}(k)$, which promotes more stability in $\boldsymbol{P}(k)$ by decreasing its eigenvalues' magnitudes.

The selection can be implemented by using a randomly weighted average of all intersection points between the 'edges' of $\mathcal{U}$ and the hyperplane $(\boldsymbol{g}_j, \tau_j)$, generating a solution $\Delta\boldsymbol{u}_{ran}^{(j)}$ within the convex hull defined by the intersection points. From a practical point of view, identifying the

intersection points for the case of m manipulated variables would imply a $O(m2^{m-1})$ time complexity, which is manageable for small m; however, for $m \geq 4$ a more sophisticated algorithm such as the one described in [55] can be implemented.

In the context of the numerical stability of $\boldsymbol{P}(k)$, VWRLS can handle the situation more effectively than WRLS due to its flexibility, caused by the dynamic relevance given to new data; however, in practical implementations this does not imply that the eigenvalues of $\boldsymbol{P}(k)$, $\boldsymbol{\mu_P}(k)$, will be upper bounded at all sample instants. The latter phenomena is the result of the fact that there is no an upper bound for the degree of time collinearity in the matrix $\boldsymbol{X}_{k_1:k_2}$, which is implicitly used in the loss functions $\mathcal{L}_{WRLS}(k)$ and $\mathcal{L}_{VWRLS}(k)$. The time collinearity, coupled with the inherently finite resolution of computers, can lead to very large eigenvalues $\boldsymbol{\mu_P}(k)$.

Because of the potential instability of $\boldsymbol{P}(k)$, the second proposed strategy focuses on assuring all eigenvalues $\boldsymbol{\mu_P}(k)$ are below a threshold, μ_{max}, that is user-defined and hardware-dependent. The last strategy is implemented by tracking the easy-to-compute trace of $\boldsymbol{P}(k)$, compare it to $n\mu_{max}$ and, if larger than the latter, implementing a saturation-like operation over $\boldsymbol{\mu_P}(k)$ to produce an upper-bound eigenvalue matrix $\boldsymbol{P}_{ube}(k)$. In more detail, if $Tr\big(\boldsymbol{P}(k)\big) \geq n\mu_{max}$, then a multiple of the identity matrix $c_\mu \boldsymbol{I}$ is "injected" into $\boldsymbol{P}(k)$ by making $\boldsymbol{P}_{ube}(k) = \boldsymbol{D}(k)\left(c_\mu \boldsymbol{I} + \text{diag}\big(\boldsymbol{\mu_P}(k)\big)^{-1}\right)^{-1}\boldsymbol{D}^T(k)$, where $c_\mu \geq 1/\mu_{max}$ and $\boldsymbol{D}(k)\text{diag}\big(\boldsymbol{\mu_P}(k)\big)\boldsymbol{D}^T(k)$ is the spectral decomposition of $\boldsymbol{P}(k)$.

It is worth noting that the calculation of $\boldsymbol{P}_{ube}(k)$ can be performed avoiding matrix inversion by carrying out (2.1.9)-(2.1.10) n times with $\lambda = 1$ and using each of the n column vectors of $c_\mu \boldsymbol{I}$ instead of the vector $\boldsymbol{x}(k)$; the latter equivalence is due to the Woodbury matrix

inversion [56], over which the Recursive Least Squares algorithm is based upon, and due to each column vector of $c_\mu \boldsymbol{I}$ being zeros except in exactly one of its elements.

A summarized pseudocode of the proposed APC algorithm is depicted in Algorithm 2.2.1.

Algorithm 2.2.1: Adaptive Predictive Control

Input: $y(k), \boldsymbol{u}(k)$
Initialize: $\boldsymbol{\theta}_{VWRLS}(0)(0), \boldsymbol{P}(0), \boldsymbol{x}(0), \Gamma = \{\gamma_1, \gamma_2, \dots, \gamma_{n_{max}}\}$
$\lambda(k) \leftarrow \boldsymbol{Variable\ forgetting\ factor}\ (\Delta\tau_{min}, A_{\Delta\tau}, y(k), \boldsymbol{\theta}_{VWRLS}(k-1), \boldsymbol{x}(k), e_{min}, e_{max}, e_{nl})$
$\boldsymbol{\theta}_{VWRLS}(k), \boldsymbol{P}(k) \leftarrow VWRLS\left(\lambda(k), y(k), \boldsymbol{x}(k), \boldsymbol{P}(k-1), \boldsymbol{\theta}_{VWRLS}(k-1)\right)$
If $Tr\big(\boldsymbol{P}(k)\big) \geq n\mu_{max}$**:**
  $\boldsymbol{P}(k) \leftarrow \boldsymbol{Information\ diversity}\ \big(\boldsymbol{P}(k), c_\mu\big)$
end If
$\boldsymbol{a}_0, \boldsymbol{b}_0 \leftarrow \boldsymbol{\theta}_{VWRLS}(k), e(k) \leftarrow \left(\left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_0 - y(k)\right)$
$Flag\_rand\_sol \leftarrow \boldsymbol{steady\ state\ error\ detection}\big(e(k), \dots, e(k-k_{ss})\big)$
$\Gamma_{\mathcal{F}} \leftarrow \{\}, \Gamma_{\mathcal{F}^c} \leftarrow \{\}$
$\boldsymbol{A}_{\gamma_{n_{max}}}, \boldsymbol{B}_{\gamma_{n_{max}}}, \boldsymbol{G}_{\gamma_{n_{max}}} \leftarrow I_{\boldsymbol{M}}\big(\boldsymbol{a}_0, \boldsymbol{b}_0, \gamma_{n_{max}}\big)$
For $j = \gamma_1$ to $\gamma_{n_{max}}$
  $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{g}_j \leftarrow \boldsymbol{Extract}\left(\boldsymbol{A}_{\gamma_{n_{max}}}, \boldsymbol{B}_{\gamma_{n_{max}}}, \boldsymbol{G}_{\gamma_{n_{max}}}\right)$
  $\tau_j \leftarrow \left(\left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_j - \boldsymbol{a}_j \boldsymbol{y}_{past}(k) + \boldsymbol{b}_j \boldsymbol{u}_{past}(k) + \boldsymbol{g}_j \boldsymbol{u}(k-1)\right)$
  $\Delta\boldsymbol{u}^{(j)}(k) \leftarrow \frac{\boldsymbol{g}_j{}^T}{\boldsymbol{g}_j \boldsymbol{g}_j{}^T} \tau_j^{(c)}$
  $\boldsymbol{u}^{(j)}(k) \leftarrow \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}^{(j)}(k)$
  If $(m > 1)$
    If $\big(\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}\big)\boldsymbol{or}(Flag\_rand\_sol = 1)$
      $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{Feasible\ solution}\left(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}, \boldsymbol{g}_j, \tau_j, Flag\_rand\_sol, e(k), \dots, e(k-k_{ss})\right)$
      If $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \neq Null$:
        $\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}$
      Else
        $\Gamma_{\mathcal{F}^c} \leftarrow \Gamma_{\mathcal{F}^c} \cup \{j\}, \boldsymbol{u}_{sat}^{(j)} \leftarrow SAT\big(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}\big)$
      end If
    Else
      $\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}, \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{u}^{(j)}(k)$
    end If
  Else
    If $\big(\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}\big)$
      $\Gamma_{\mathcal{F}^c} \leftarrow \Gamma_{\mathcal{F}^c} \cup \{j\}, \boldsymbol{u}_{sat}^{(j)} = SAT\big(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}\big)$
    Else
      $\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}, \boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{u}^{(j)}(k)$
    end If
  end If
end for
If $\Gamma_{\mathcal{F}} \neq \{\}$:
  $\gamma^* = min\, \Gamma_{\mathcal{F}}$

$\boldsymbol{u}(k) \leftarrow \boldsymbol{u}_{\mathcal{F}}^{(\gamma^*)}$
Else
$\gamma^* = \underset{i \in \Gamma_{\mathcal{F}^c}}{\operatorname{argmin}} \left| \tau_j - \boldsymbol{g}_j \boldsymbol{u}_{sat}^{(j)} \right|$
$\boldsymbol{u}(k) \leftarrow \boldsymbol{u}_{sat}^{\gamma^*}$
end If

2.3 Industrial application of Adaptive-Linear-Model-based Control Algorithm

2.3.1 Context of implementation

Data center (DC) energy consumption has attracted a lot of attention in recent years. According to [57], DC energy consumption ranges from 1.1% to 1.5% of total global electricity consumption, with this proportion showing a tendency to increase [58]. A significant portion of this energy utilization is devoted to cooling systems that aim to keep server temperatures within a safe region, necessary to avoid damage to servers. Traditionally in DCs, cooling infrastructure is either room-based or row-based [59]-[60]. However, in recent years rack-mountable cooling units have been introduced to cope with the increasing demand for high performance computing (HPC). These new architectures bring servers and cooling units closer to each other with an aim to decrease cooling infrastructure energy consumption [60].

In addition to reducing energy consumption, maintaining a stable temperature inside a data center is crucial since oscillations in air temperature, even by 1 or 2 degrees, increase the probability of server failures [61]. These oscillations are an inherent characteristic of the ON/OFF or PID controllers which have been widely used in cooling infrastructure [62]. Due to the proximity of the rack-mounted cooling units to the servers, any variation in airflow created by these controllers, as a response to changes in workload, will be experienced immediately by the servers, which will consequently lead to higher server failure rates.

Within the previous scope, the APC algorithm defined in Section 2.2.4 is implemented in a rack-mountable cooling unit with limited computational capacity and developed by an industrial partner, the Computing Infrastructure Research Centre (CIRC) (now FYELABS). The goal of this implementation is to utilize a data-driven control method with hardware limitations that can adapt to changes in the system, such as addition and removal of servers. Hence, the proposed APC is chosen due to being implementable on a low-cost and memory limited, off the shelf general purpose microcontroller. Furthermore, the APC is extended to take monetary costs into consideration by adding a projected gradient-based algorithm so that, unlike other low complexity controllers, it can address power consumption and operating costs.

2.3.2 Hardware and physical system description

The test bed considered for the APC implementation consists of a single rack containing 20 servers with an average maximum power consumption of 250W across all servers, and a rack-mounted cooling unit located at the top of the rack which uses air as the cooling medium. The cooling unit consists of a heat exchanger and a set of five identical compact industrial fans. The controlled variable is the rack's temperature, measured using a sensor of 0.06°C resolution and located in front of the 12th server, the hottest point in front of the rack. The manipulated variables are the water flow in the heat exchanger and the PWM signals of the fans. A schematic of the rack can be observed in Fig. 2.3.1.

The water flow in the heat exchanger is regulated by an on/off valve, whose aperture is controlled by a local feedback loop model-based algorithm that generates electrical pulses to manipulate the aperture. Hence, the input to the water flow regulation algorithm is the desired value of water flow computed by the temperature controller. The water flow, with a maximum

value near 21L/min, is measured by a sensor with average resolution of 1.06L/min whose value is fed back to the water regulation algorithm. The fans, with a maximum power consumption of 168W per unit, are directly manipulated by the controller through 8-bit resolution PWM signals at 488Hz.

Water is supplied by a branch of the building's water system, and its temperature is regulated by an outside controller using cooling tower technology. Since the outside controller is not designed for delivering a constant water temperature, there are changes in the water inlet temperature of the heat exchanger in the rack, which can have significant impact on the system. Therefore, the water inlet temperature is considered a disturbance for the system.

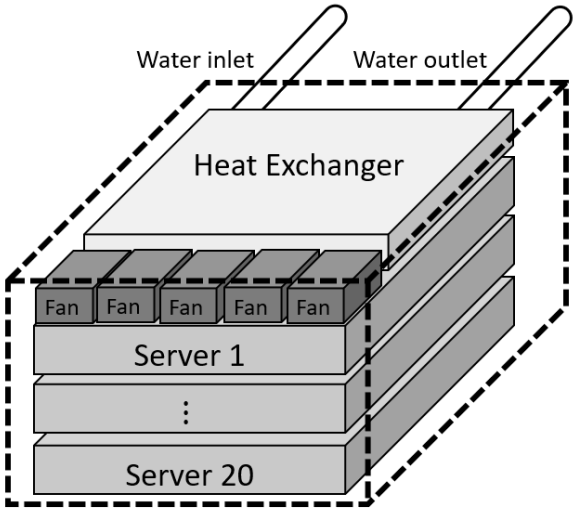


Fig. 2.3.1. Schematic of the rack configuration and cooling system location.

An Arduino Mega, a low-cost general-purpose microcontroller, is used to implement the proposed APC. The microcontroller is characterized by having 8KB of SRAM memory, a 256KB Flash memory and a 16MHz crystal oscillator.

2.3.3 Implementation Challenges in Real-Time APC and Monetary Optimization

One of the limitations with the APC's adaptive algorithm is that, even when the error is close to zero, the parameters $\boldsymbol{\theta}_{VWRLS}(k)$ do not necessarily represent the true dynamics of the system, due to the limited amount of online information. The latter can in turn have a significant impact on the performance of the predictive algorithm, especially for long prediction horizons. This issue can be handled by using one of the following approaches: constraining $\boldsymbol{M}_{VWRLS}(k)$ to be in a specific space or establishing constraints for $\boldsymbol{g}_j$ in the predictive algorithm. In this implementation, the latter is chosen.

Typically, the output of a dynamic system is physically constrained, i.e., $y_{min} \leq y(k) \leq y_{max}, \forall k$. Hence, in the GPC-based algorithm the following constraint can be imposed: $\hat{y}_{free}^{(c)}(k+j|k) = SAT\big(\boldsymbol{a}_j \boldsymbol{y}_{past}(k) + \boldsymbol{b}_j \boldsymbol{u}_{past}(k) + \boldsymbol{g}_j \boldsymbol{u}(k-1); y_{min}; y_{max}\big)$; where $y_{max} > y_{min}$ and $\hat{y}_{free}^{(c)}(k+j|k)$ represents the estimated (free) evolution of the system at time instant j, when no change in the input is implemented, so that the constrained estimated output error is $\tau_j^{(c)} = \left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_j - \hat{y}_{free}^{(c)}(k+j|k)$. In addition, since (2.2.12) captures in $\boldsymbol{g}_j$ the physical effect that each manipulated variable has on the output, it is possible to constrain the values of $\boldsymbol{g}_j$ by setting $\boldsymbol{g}_j^{(c)} = SAT\big(\boldsymbol{g}_j, \boldsymbol{g}_{min}, \boldsymbol{g}_{max}\big)$.

Even when $\boldsymbol{g}_{min}$ and $\boldsymbol{g}_{max}$ are not known, due to physical constraints most industrial systems do have a minimum and maximum gain for each input variable and it is information that can be easily obtained or estimated. Consequently, by using these minimum and maximum possible gains with possibly a margin of error (to overestimate) it is possible to establish a lower bound for the true $\boldsymbol{g}_{min}$ and an upper bound for the true $\boldsymbol{g}_{max}$. The result is a reduction in the

negative effects of transitory lack of information and large perturbations that dramatically change the value of $\boldsymbol{\theta}_{VWRLS}(k)$ in a sudden manner and propagate to $\boldsymbol{g}_j$.

For cases in which the signs of the gains of the manipulated variables, with respect to the output, are known and constant, constraining $\boldsymbol{g}_j$ can avoid counterintuitive control actions ($\boldsymbol{u}_{\mathcal{F}}^{(j)}(k)$) that, even though they do not necessarily prevent the desired set-points from being achieved, can waste excessive energy. To elaborate further, such cases can occur for a multiple input system in which the output behavior of one variable can be attributed to multiple inputs, and limited information about the inputs can cause a false attribution of positive gain effects to input variables with negative gains.

By considering the economic aspect linked to manipulated variable usage and exploiting the flexibility of (2.2.12), it is possible to implement an algorithm that minimizes the monetary cost rate $C_{\$}\big(\boldsymbol{u}(k)\big)$. Assuming the monetary cost function is differentiable, a gradient descent algorithm restricted to be orthogonal to $\boldsymbol{g}_j^{(c)^T}$ can be implemented, with a learning rate $\alpha_{\$}$. The resulting direction for minimizing $C_{\$}\big(\boldsymbol{u}(k)\big)$ is then given by

$$-\nabla C_{\$}\big(\boldsymbol{u}(k)\big) + \frac{\boldsymbol{g}_j^{(c)^T}}{\boldsymbol{g}_j^{(c)}\boldsymbol{g}_j^{(c)^T}}\boldsymbol{g}_j^{(c)}\nabla C_{\$}\big(\boldsymbol{u}(k)\big) \tag{2.3.1}$$

The use of (2.3.1) is suggested only when a feasible solution has been found and the system output has settled around the desired set-point, so that no additional restrictions are imposed on $\boldsymbol{u}(k)$ for the transient response, which would also increase the diversity in the input and subsequently benefit the VWRLS algorithm.

A summarized pseudocode of the practical implementation of the APC can observed in Algorithm 2.3.1.

Algorithm 2.3.1: Adaptive Predictive Control for practical implementation

Input: $y(k)$, $\boldsymbol{u}(k)$

Initialize: $\boldsymbol{\theta}_{VWRLS}(0)(0), \boldsymbol{P}(0), \boldsymbol{x}(0), \Gamma = \{\gamma_1, \gamma_2, \dots, \gamma_{n_{max}}\}$

$\lambda(k) \leftarrow \boldsymbol{Variable\ forgetting\ factor}\ (\Delta\tau_{min}, A_{\Delta\tau}, y(k), \boldsymbol{\theta}_{VWRLS}(k-1), \boldsymbol{x}(k), e_{min}, e_{max}, e_{nl})$

$\boldsymbol{\theta}_{VWRLS}(k), \boldsymbol{P}(k) \leftarrow VWRLS\left(\lambda(k), y(k), \boldsymbol{x}(k), \boldsymbol{P}(k-1), \boldsymbol{\theta}_{VWRLS}(k-1)\right)$

If $Tr\left(\boldsymbol{P}(k)\right) \geq n\mu_{max}$**:**

$\boldsymbol{P}(k) \leftarrow \boldsymbol{Information\ diversity}\ \left(\boldsymbol{P}(k), c_\mu\right)$

end If

$\boldsymbol{a}_0, \boldsymbol{b}_0 \leftarrow \boldsymbol{\theta}_{VWRLS}(k)$,

$e(k) \leftarrow \left(\left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_0 - y(k)\right)$

$Flag\_rand\_sol \leftarrow \boldsymbol{steady\ state\ error\ detection}\left(e(k), \dots, e(k-k_{ss})\right)$

$\Gamma_{\mathcal{F}} \leftarrow \{\}$, $\Gamma_{\mathcal{F}^c} \leftarrow \{\}$

$\boldsymbol{A}_{\gamma_{n_{max}}}, \boldsymbol{B}_{\gamma_{n_{max}}}, \boldsymbol{G}_{\gamma_{n_{max}}} \leftarrow I_{\boldsymbol{M}}\left(\boldsymbol{a}_0, \boldsymbol{b}_0, \gamma_{n_{max}}\right)$

For $j = \gamma_1$ to $\gamma_{n_{max}}$

$\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{g}_j \leftarrow Extract\ \left(\boldsymbol{A}_{\gamma_{n_{max}}}, \boldsymbol{B}_{\gamma_{n_{max}}}, \boldsymbol{G}_{\gamma_{n_{max}}}\right)$

$\boldsymbol{g}_j^{(c)} = SAT\left(\boldsymbol{g}_j, \boldsymbol{g}_{min}, \boldsymbol{g}_{max}\right)$

$\hat{y}_{free}^{(c)}(k+j|k) = SAT\left(\boldsymbol{a}_j \boldsymbol{y}_{past}(k) + \boldsymbol{b}_j \boldsymbol{u}_{past}(k) + \boldsymbol{g}_j^{(c)} \boldsymbol{u}(k-1), y_{min}, y_{max}\right)$

$\tau_j^{(c)} \leftarrow \left(\left[\boldsymbol{y}_{desired}^{(\gamma_{n_{max}})}(k)\right]_j - \hat{y}_{free}^{(c)}(k+j|k)\right)$

$\Delta\boldsymbol{u}^{(j)}(k) \leftarrow \frac{\boldsymbol{g}_j^{(c)^T}}{\boldsymbol{g}_j^{(c)} \boldsymbol{g}_j^{(c)^T}} \tau_j^{(c)}$

$\boldsymbol{u}^{(j)}(k) \leftarrow \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}^{(j)}(k)$

If $(m > 1)$:

If $\left(\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}\right)\boldsymbol{or}(Flag\_rand\_sol = 1)$

$\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{Feasible\ solution}\left(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}, \boldsymbol{g}_j^{(c)}, \tau_j^{(c)}, Flag\_rand\_sol, e(k), \dots, e(k-k_{ss})\right)$

If $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \neq Null$:

$\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}$

Else

$\Gamma_{\mathcal{F}^c} \leftarrow \Gamma_{\mathcal{F}^c} \cup \{j\}$, $\boldsymbol{u}_{sat}^{(j)} \leftarrow SAT\left(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}\right)$

end If

Else

$\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}$, $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{u}^{(j)}(k)$

end If

Else

If $\left(\boldsymbol{u}^{(j)}(k) \notin \mathcal{U}\right)$

$\Gamma_{\mathcal{F}^c} \leftarrow \Gamma_{\mathcal{F}^c} \cup \{j\}$, $\boldsymbol{u}_{sat}^{(j)} = SAT\left(\boldsymbol{u}^{(j)}(k), \boldsymbol{u}_{min}, \boldsymbol{u}_{max}\right)$

Else

$\Gamma_{\mathcal{F}} \leftarrow \Gamma_{\mathcal{F}} \cup \{j\}$, $\boldsymbol{u}_{\mathcal{F}}^{(j)}(k) \leftarrow \boldsymbol{u}^{(j)}(k)$

end If

end If

end for

If $\Gamma_{\mathcal{F}} \neq \{\}$:

$\gamma^* = min\ \Gamma_{\mathcal{F}}$

Else

$\gamma^* = \underset{i \in \Gamma_{\mathcal{F}^c}}{\operatorname{argmin}} \left|\tau_j^{(c)} - \boldsymbol{g}_j^{(c)} \boldsymbol{u}_{sat}^{(j)}\right|$

end If
If *Cost reduction is enabled* **and** $\Gamma_{\mathcal{F}} \neq \{\}$:

$\Delta \boldsymbol{u}_{\$} \longleftarrow -\nabla C_{\$}\left(\boldsymbol{u}_{\mathcal{F}}^{(\gamma^*)}(k)\right) + \frac{\boldsymbol{g}_j^{(c)^T}}{\boldsymbol{g}_j^{(c)}\boldsymbol{g}_j^{(c)^T}}\boldsymbol{g}_j \nabla C_{\$}\left(\boldsymbol{u}_{\mathcal{F}}^{(\gamma^*)}(k)\right)$

$\boldsymbol{u}(k) \longleftarrow \boldsymbol{Cost\ reduction}\left(\Delta \boldsymbol{u}_{\$}, \boldsymbol{u}_{\mathcal{F}}^{(\gamma^*)}(k), \alpha_{\$}, e(k)\right)$

Else

$\boldsymbol{u}(k) \longleftarrow \boldsymbol{u}_{sat}^{\gamma^*}$

end If

It should be highlighted that for the proposed VWRLS a proof of convergence is not available. However, the variable forgetting factor implemented belongs to a family of RLS algorithms described in [63] where, under some excitation and boundedness assumptions for $\boldsymbol{x}(k)$, it is shown that for systems with time varying parameters and a bounded disturbance, RLS with a variable forgetting factor will have a bounded tracking error. This, together with the random solution selection, the lower bound on the forgetting factor, λ_{min}, from Section 2.1.3 as well as the experimental results shown in the following sections, validate the effectiveness of the algorithm.

2.3.4 APC Experimental Simulations

The proposed APC defined in Algorithm 2.3.1, except for the monetary compensation, is compared with a standard APC via simulation, replacing the proposed predictive algorithm with the original version of the GPC, both using VWRLS as the adaptive algorithm. The proposed controllers were tested in a MATLAB simulation environment. The simulated system has similar characteristics to the physical system described in Section 2.3.1 and it is composed of independent linearized subsystems of PWM signals, water flow and water inlet temperature; the water inlet temperature is considered as a disturbance. For the standard APC the parameters of the GPC's loss function (2.2.10) were set to $\gamma_c = 1$, $\boldsymbol{Q}_y = \boldsymbol{Q}_u = \boldsymbol{I}$. Additionally, a PI controller designed with the

Split-Range strategy (PI-SR), described in [64]-[65] and characterized by a hierarchy of manipulated-variables activation sequences, was simulated to provide a comparison to an alternative low-complexity control algorithm.

The VWRLS parameters for both APC controllers were set as follows: $e_{min} = 0.05$, $e_{max} = 0.2$, $e_{nl} = 0.01$, $p_{\text{old}} = 0.1$, $\Delta\tau_{min} = 350$, $A_{\Delta\tau} = 100$, and $\mu_{max} = 1000$. Also, the VWRLS considered eight previous values of the output and inputs, i.e., $d = e_1 = e_2 = 8$, for a total of 24 learnable parameters. For both controllers, the maximum prediction horizon $\gamma_{n_{max}}$ was set to 24. This value was tuned to optimize the performance of the APC with standard GPC. In addition, γ_1 was set to 8 and the output's constraints were set to $y_{min} = 15$, $y_{n_{max}} = 35$.

Since the constraints for $\boldsymbol{g}_j$, described in Section 2.3.3, decreased the performance of the standard APC in simulations, it was implemented only in the proposed APC, where $\boldsymbol{g}_{min}$ and $\boldsymbol{g}_{max}$ were determined experimentally and set to $[-0.2/255, -0.2/27]$ and $[-10/255, -10/27]$, respectively. Also, the random solution selection from Section 2.2.4 was implemented using the simple algorithm of iterating across each edge in $\mathcal{U}$, since the space dimensionality considered was low, two in the current implementation. The random solution selection was not compatible with the standard GPC, therefore it was not implemented in that setting.

For the simulated system some additional physical restrictions were implemented. The output temperature was discretized to a resolution of 0.06°C, identical to the resolution of the sensors used in the physical system described in Section 2.3.2. Also, the water inlet temperature variable was set to a constant value of 12°C plus a sinusoidal function with a period of 300s and bounded random amplitude to test the system under perturbation conditions. The water flow and PWM signal values calculated by the controllers were discretized to 0.02L/min and 1 unit,

respectively, and for both controllers their values were constrained to [100, 255] for PWM and [10, 27] for the water flow. The amplitude of the water temperature sinusoidal component followed a half-normal distribution with mean absolute value μ of 0.4°C and standard deviation σ of 0.3°C. Finally, a sampling time of 5 seconds was used since it represented the minimum stable sampling time that could be used in the real implementation.

One relevant aspect resulting from the physical constraints in the simulated system is the increased time for the VWRLS algorithm to converge, due to the resolution of the PWM and the sensor readings which decrease the frequency information in the signals. In addition, the APC with standard GPC does not identify which values $(\boldsymbol{g}_j, \tau_j)$ generate infeasible solutions. In this aspect, since the proposed APC first identifies and discards solutions that could lead to this problem, it is expected to provide better regulation.

For the PI-SR control the procedure defined in [65] was implemented, resulting in internal parameter values $v^* = 0.2838$, $\alpha_1 = 2.142, \alpha_2 = 0.93$, $K_c = 16.50$ and $\tau_I = 70$. For this scheme, the water flow was used as the first manipulated variable in the PI-SR hierarchy. The results of the simulations are shown from Fig. 2.3.2 to Fig. 2.3.4. The Mean Squared Error of the simulation results for Fig. 2.3.2 can be found in Table 2.3.1.

From Fig. 2.3.2 it can be observed that the PI-SR and proposed APC algorithms have similar performance, with both showing improved performance over the APC with standard GPC. However, PI-SR shows more oscillations around the operating point than the APC controllers, possibly due to the activation of the second manipulated variable, as observed in Fig. 2.3.4. Similar behavior was observed with respect to the water flow manipulated variable for both APC controllers, but more short-term oscillations were generated by the proposed APC as identified in Fig. 2.3.3, mostly attributed to the variable prediction horizon and the random solution selection.

Also, while the APC with standard GPC took more than 200s to show reasonable performance, the proposed APC stabilized in less than 100s.

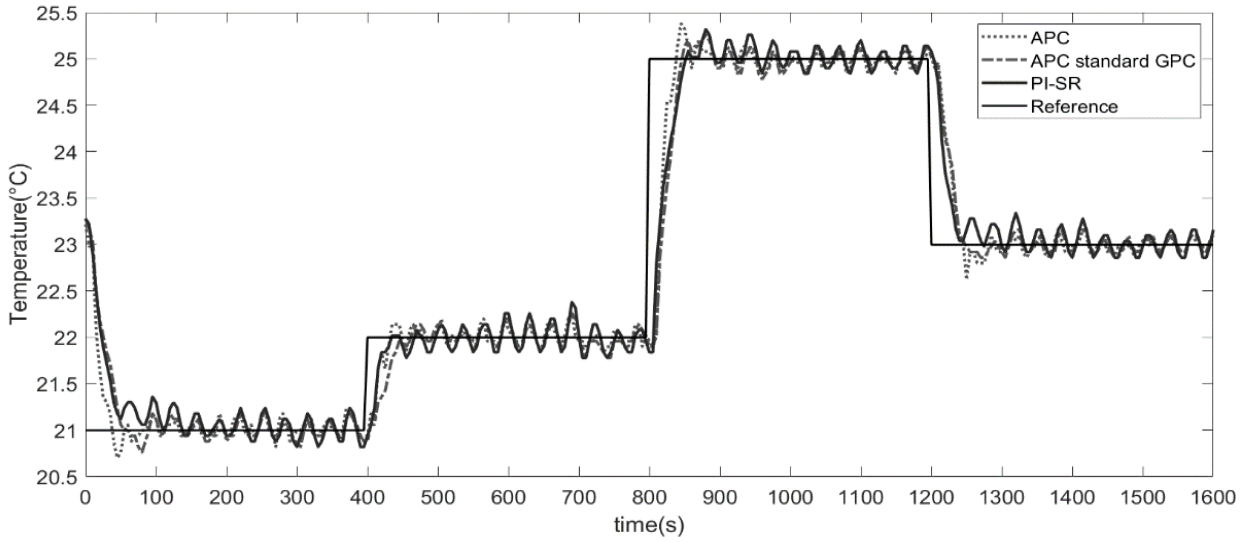


Fig. 2.3.2. Performance of APC with standard GPC, PI-SR and proposed APC. The random sinusoidal amplitude parameters of the water temperature are set to μ=0.4°C and σ=0.3°C.

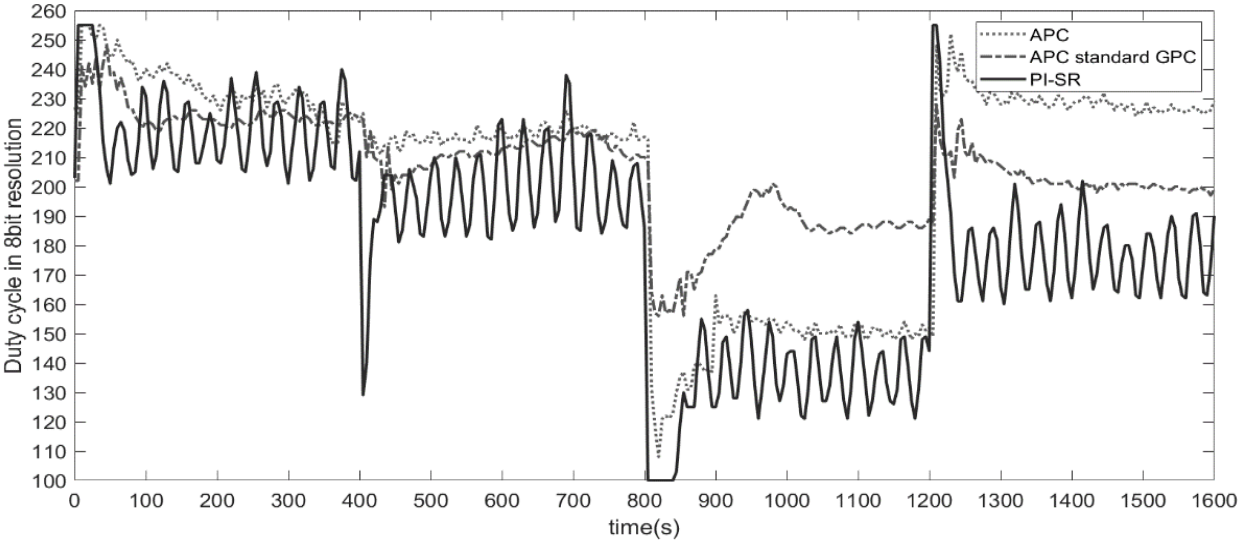


Fig. 2.3.3. PWM values for the APC with standard GPC, PI-SR and proposed APC. Resolution of 1, within the range [100,255] (8-bit representation).

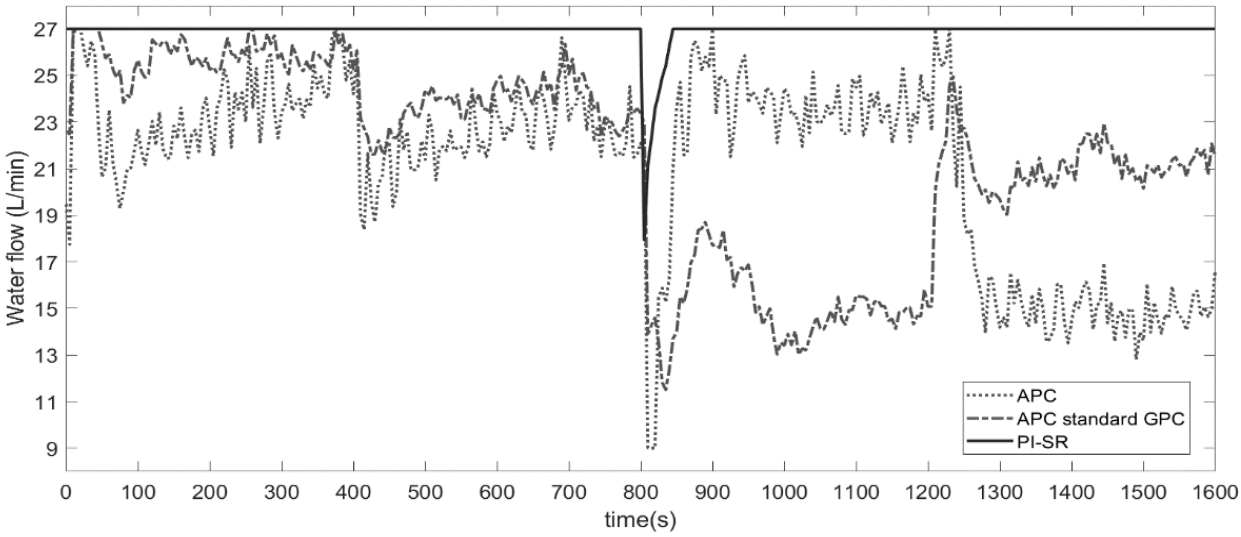


Fig. 2.3.4. Water flow values for the APC with standard GPC, PI-SR and proposed APC. Resolution of 0.02, within the range [10,27].

Table 2.3.1. RMSE performance of the controllers in the simulation.

Controller	RMSE
APC standard GPC	0.5095
APC	0.4670
PI-SR	0.4778

It is important to note that while stability cannot be guaranteed for the manipulated values generated by the proposed APC, for the current stable system the VWRLS algorithm when facing model mismatch will use the error values, $e(k|k-1) = y(k) - \hat{y}(k|k-1)$ to make corrections. In addition, the constraints $\boldsymbol{g}_{min}$ and $\boldsymbol{g}_{max}$ used in the GPC-based algorithm will result in the model remaining in a region more consistent with the physical properties of the system, independently of how large the error values are. Furthermore, one of the main advantages of the proposed APC approach in the current implementation is that precise tuning was not required to have competitive performance.

The proposed APC controller showed two major advantages over the standard APC in the presented simulations. First, it can discard values for manipulated variables that are outside of the physical constraints, whereas such values implicitly affect the standard APC. Second, the proposed APC can create more diversity in the values of the manipulated variables by selecting semi-random solutions when a threshold of steady-state error is reached, which leads to a more accurate model computed by the adaptive algorithm.

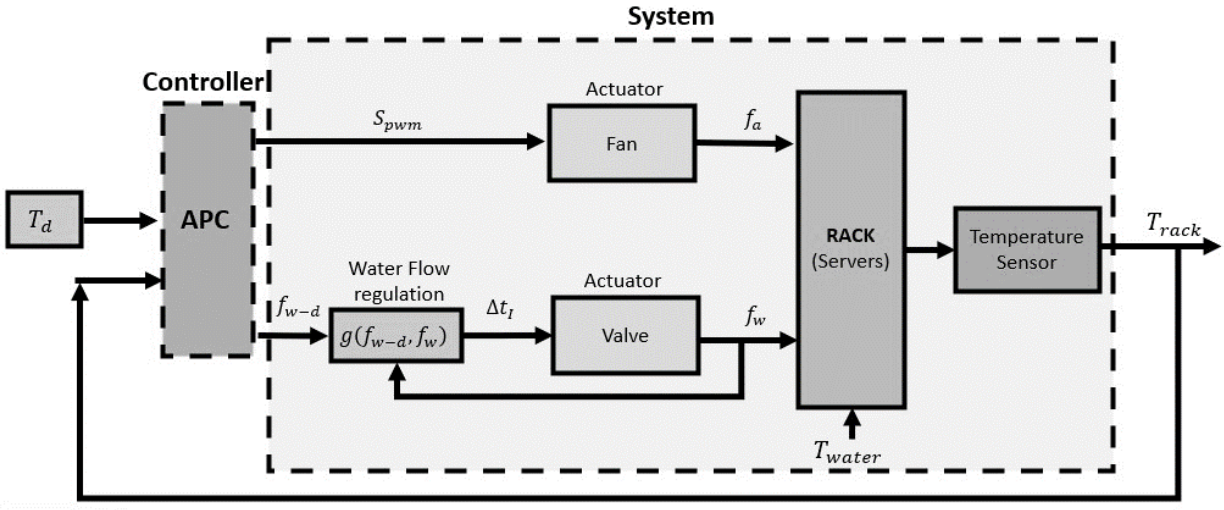
2.3.5 APC Experiments in a rack-mounted cooling unit

The proposed APC defined in Algorithm 2.3.1 (Section 2.3.3) was used to perform a set of experiments on the physical system described in Section 2.3.2. Both the proposed APC and the water flow regulation algorithm were implemented on the low-cost microcontroller installed in the cooling system, since the memory space proved to be more than enough for their implementation. Given the memory and CPU speed constraints (8KB SRAM and 16MHz), a computationally expensive iterative controller implementation would have been infeasible for the desired sampling time (5s). In contrast, the proposed APC required approximately the same memory space as a standard unconstrained APC with control horizon of 1. In more detail, the implementation of Algorithm 2.3.1, including the monetary cost reduction strategy, required approximately 60% of the SRAM for static variables and 30% for non-static variables, resulting in 90% total memory usage.

It is important to note that even though the water flow regulation algorithm was encoded in the same microcontroller, this algorithm was transparent to the proposed APC since its parameters are not used for the controller design and it represented less than 2% of the memory used. Hence, the water flow regulation algorithm was considered part of the controlled physical

system. Also, the acquisition times for the water flow and temperatures were close to 4.5s, which made 5s the minimum achievable stable sampling time in the microcontroller. A general schematic of the implementation of the APC is shown in Fig. 2.3.5.

The parameters for the APC were set as follows: $\hat{e}_{min} = 0.045$, $\hat{e}_{max} = 0.2$, $\hat{e}_z = 0.001$, $p_{\min} = 0.1$, $w_{min} = 150$ and $\mu_{min} = 0.1$. The eight previous values of outputs and inputs yield a total of 24 coefficients. Also, $\gamma_{min} = 3$, $\gamma_{max} = 14$, $y_{min} = 20$, $y_{max} = 40$, and $\boldsymbol{g}_{min}$ and $\boldsymbol{g}_{max}$ values were set to $[-0.2/255, -0.2/30]$ and $[-10/255, -10/30]$, respectively. The water flow and PWM values calculated by the APC were discretized to 0.02L/min and 1unit, respectively. Finally, the manipulated variables were constrained to [35,255] for PWM and [9,21]L/min for water flow. The minimum achievable stable sampling time of 5 seconds was used for these experiments and their results can be observed in Fig. 2.3.6 - Fig. 2.3.8.



Algorithm 2	Block diagram	Variable's physical description
$y_{desired}^{(y_{n_{max}})}(k)$	T_d	Desired temperature
$y(k)$	T_{rack}	System's temperature
$[\boldsymbol{u}(k)]_1$	S_{pwm}	Pulse Width Modulation signal (8bit resolution)
$[\boldsymbol{u}(k)]_2$	f_{w-d}	Desired water flow
	f_w	Water flow
	f_a	Air flow
	T_{water}	Water inlet temperature

Fig. 2.3.5. Block diagram representation of the system being controlled and the controller.

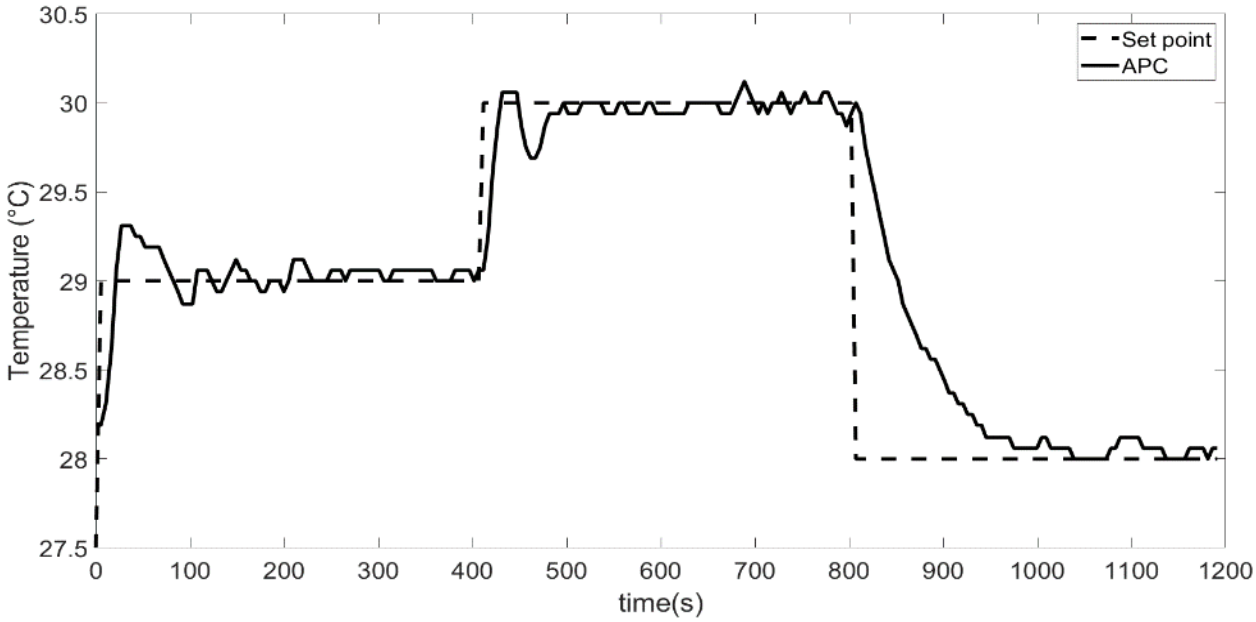


Fig. 2.3.6. Proposed APC performance with the top 12 servers on, the bottom servers off and the air ducts of the latter blocked.

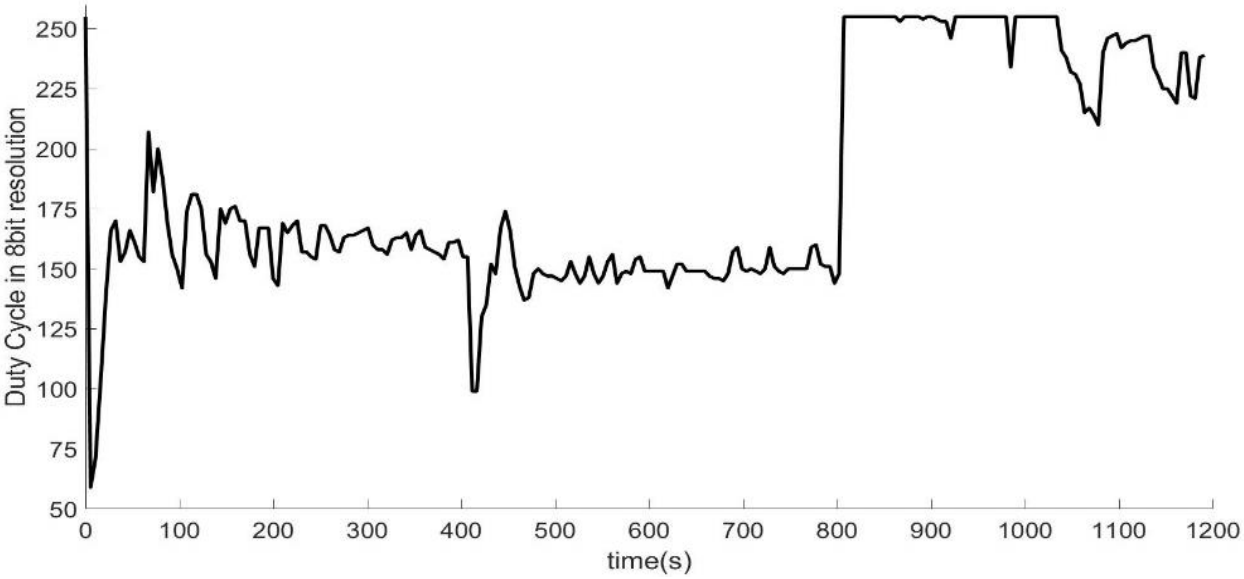


Fig. 2.6.7. PWM manipulation of APC. Resolution of 1, within the range [35,255] (8-bit representation).

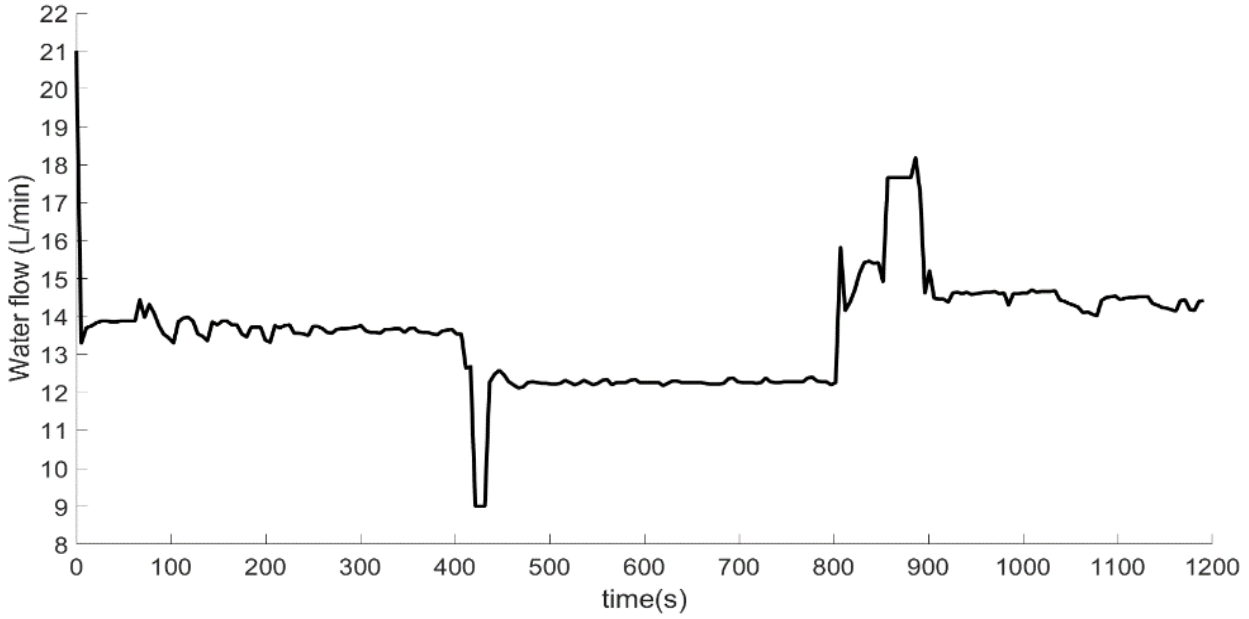


Fig. 2.3.8. Water flow manipulation of APC. Resolution of 0.02, within the range [9,21].

In Fig. 2.3.6, after the initial 400s when the VWRLS algorithm has obtained more information about the system, it can be observed that the APC's performance tends to improve, generating less overshoot. The decreased overshoot can be mainly attributed to the forecasting performed by the predictive algorithm of APC. In addition, it is worthwhile noticing that in Fig. 2.3.7 and Fig. 2.3.8 the variations in steady-state found in the manipulated variables are partially caused by the water inlet temperature oscillation, and the APC is capable of incorporating this effect through parameter adaptation, implying consistent results with those obtained in simulation.

Additional experiments were conducted to test the monetary cost reduction algorithm in Section 2.3.3 and described in Algorithm 2.3.1. The results from the latter and the conditions for implementation are explained next. Assuming a case in which both water and energy have associated costs, the monetary cost function of the manipulated variables has the form $C_{\$}(\boldsymbol{u}(k)) = \boldsymbol{b}_{\$}\boldsymbol{u}^{j}$, where $\boldsymbol{b}_{\$}$ contains the cost rate of water per litre/min and an estimate of the associated cost rate for the energy spent for fans, $\boldsymbol{b}_{\$} = \left[5.94(10^{-6})\ cents/s \quad 6.340(10^{-3})\ cents/\left(\frac{L}{min}s\right)\right]$.

The latter values were computed assuming 7.3 cents/kWh and 3.84 CAD/m^3 as fixed utility prices, based on information from the province of Ontario, Canada [66]-[67].

The monetary cost reduction algorithm was activated only when the system output approximately matched the desired set-point (± 0.06) for four consecutive iterations. The performance of the APC with the monetary cost reduction, $APC_{\$}$, is shown in Fig. 2.3.9 - Fig. 2.3.11, and the cumulative monetary reduction through time is observed in Fig. 2.3.12. From them, it is possible to observe that the $APC_{\$}$ generates a reduction near 15% of $C_{\$}(\boldsymbol{u}(k))$ when compared to the APC. Considering that a typical large DC has from hundreds to thousands of rack units, the savings of an estimated 3700CAD per rack per year are significant. It is important to note that both algorithms, $APC_{\$}$ and APC, are similar in general performance. However, since the former slowly changes the state of the system by making small adjustments to $\boldsymbol{u}(k)$, it can become more vulnerable to disturbance effects caused by the water temperature, as observed in Fig. 2.3.9. Despite this, when the system changes the desired set-point the algorithm that minimizes $C_{\$}(\boldsymbol{u}(k))$ is not active until it returns to the set-point.

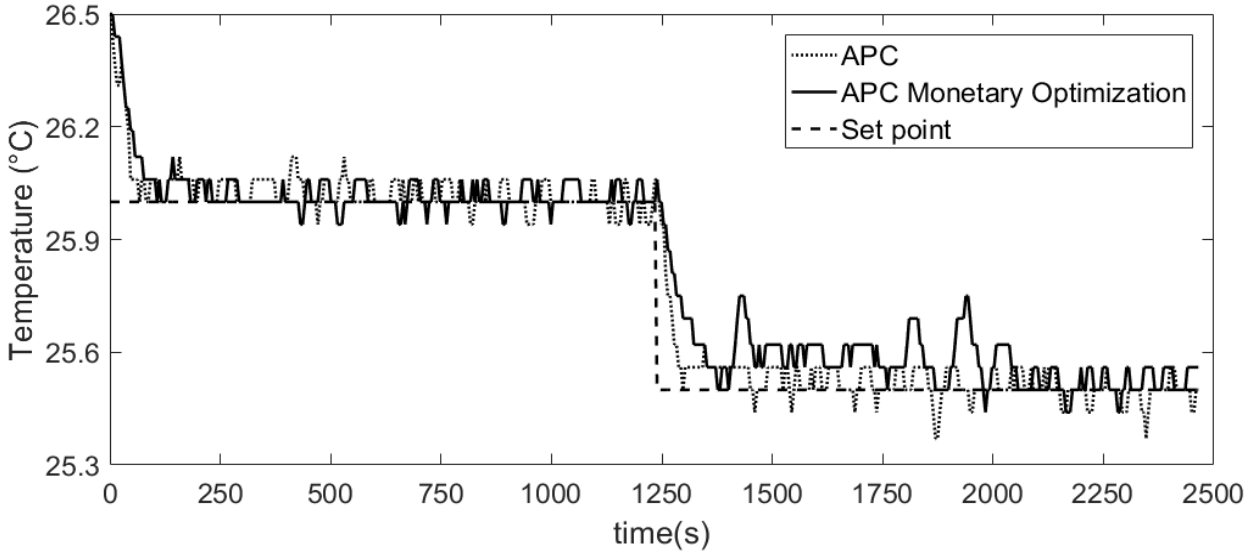


Fig. 2.3.9. APC and APC with Monetary optimization performance

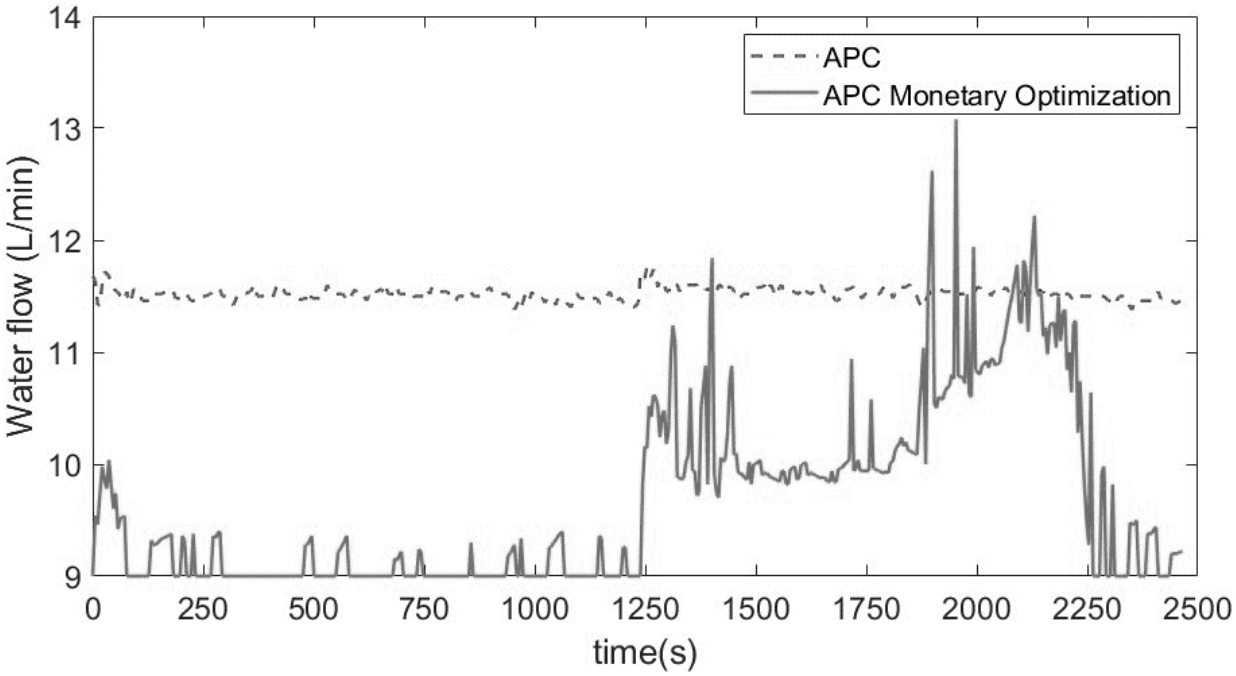


Fig. 2.3.10. Water flow manipulation of APC and APC with Monetary optimization.

Resolution of 0.02, within the range [9,21].

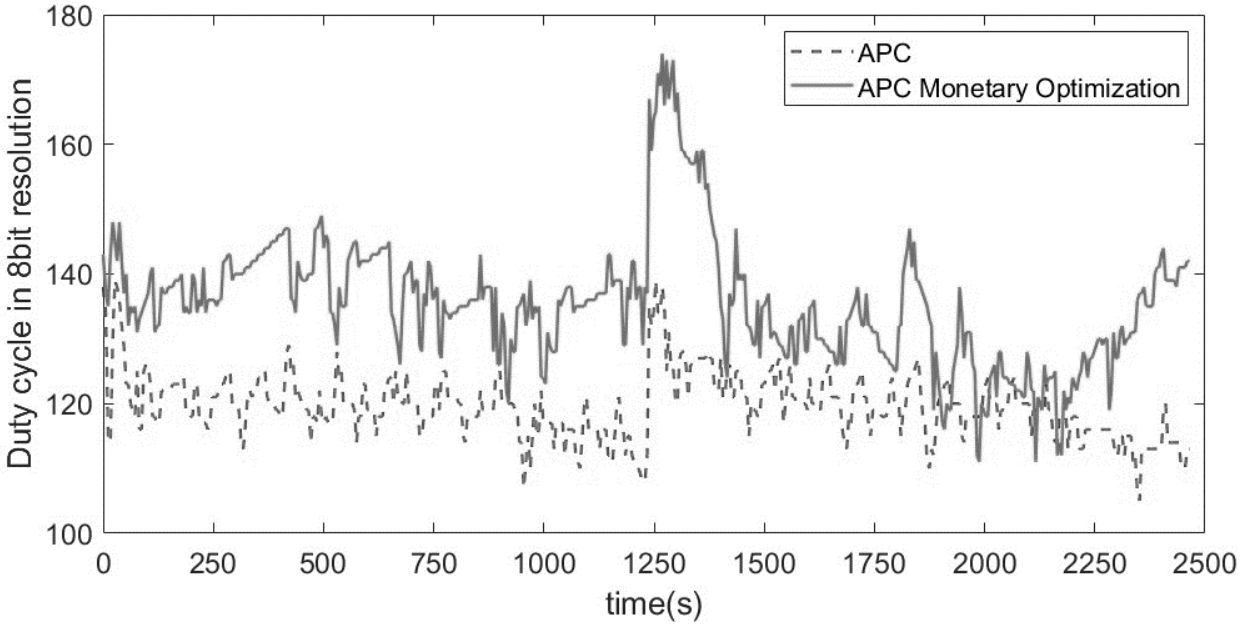


Fig. 2.3.11. PWM manipulation of APC and APC with Monetary optimization. Resolution of 1, within the range [35,255] (8-bit representation).

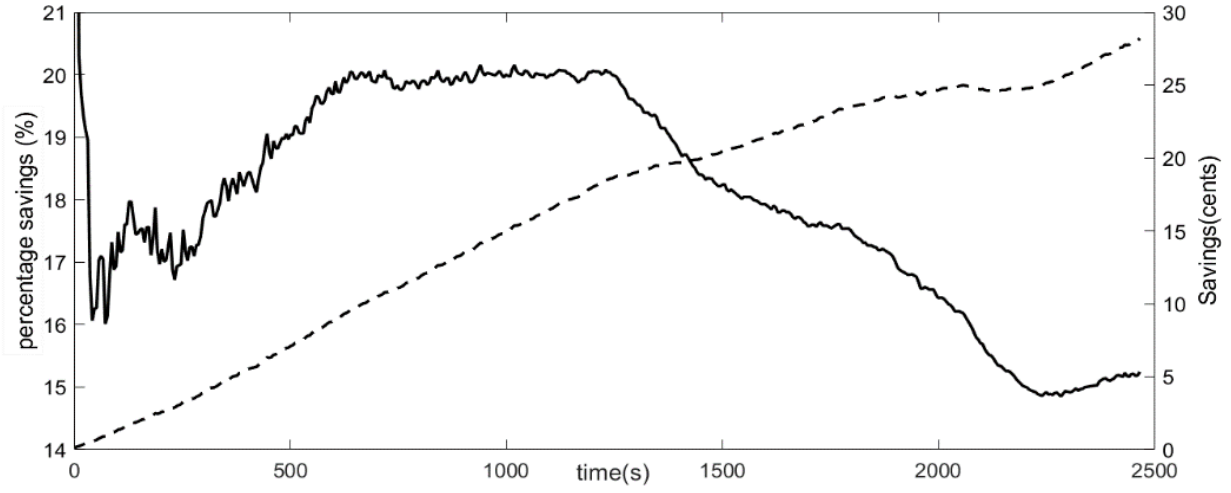


Fig. 2.3.12. Savings of the APC with $C_{\$}(\boldsymbol{u}(k))$ enabled, with respect to $C_{\$}(\boldsymbol{u}(k))$ disabled.

The behavior of the manipulated variables for the APC$_{\$}$ are consistent with the minimization objective defined by $\boldsymbol{b}_{\$}$, since the energy price of the water flow is more expensive per unit than that linked to the fans.

The Mean Squared Error of the results for Fig. 2.3.5 and Fig. 2.3.9 can be found in Table 2.3.2 as Experiment 1 and Experiment 2, respectively.

Table 2.3.2. RMSE performance of the controllers in the simulation

	Controller	**RMSE**
Experiment 1	APC	0.3650
Experiment 2	APC	0.0814
	APC$_{\$}$	0.1088

Chapter 3

Neural-Network-based Models for Time Series

In this chapter a brief introduction to Neural Network models intended for time series is presented. Feedforward and Recurrent Neural Network models are covered, emphasizing the LSTM recurrent architecture. In addition, the backpropagation (BP) algorithm, used in these models, is concisely presented using compact matrix notation to facilitate its readability and interpretation. For reference in later chapters standard equations for each architecture, linked to the algorithm, are expressed; these equations are also used to describe the models' capabilities and limitations.

3.1 Feedforward and Recurrent Neural Networks

In recent years, the field of time series has been through a gradual but important transformation, caused in large advances in the machine learning (ML) area. Among these, Neural Networks (NN) have emerged as an effective alternative to linear models, specifically due to the capability to model nonlinearities in time-dependent data.

When compared to Adaptive Linear Models (ALM) studied in previous chapters, NN models carry out a more direct approach when used to handle nonlinearities, by trying to capture their effects on the output, $\boldsymbol{y}(k)$, using nonlinear functions, $\sigma(\cdot)$, usually referred to as activation

functions or hidden neurons. Feedforward neural networks (FNN), one of the most popular architectures, have shown remarkable results beyond time series modelling and across many scientific applications [23]-[26]. One of the simplest FNN architectures, with one hidden layer, can be observed in (3.1.1)-(3.1.2) (with its corresponding graphical representation in Fig. 3.1.1):

$$\boldsymbol{h}^{(1)}(k) = \sigma\big(\boldsymbol{W}_0\boldsymbol{x}(k)\big) \tag{3.1.1}$$

$$\widehat{\boldsymbol{y}}(k) = \boldsymbol{W}_1\boldsymbol{h}^{(1)}(k) \tag{3.1.2}$$

where $\boldsymbol{h}^{(1)}(k) \in \mathbb{R}^r$, $\boldsymbol{W}_0 \in \mathbb{R}^{r\times n}$, $\boldsymbol{W}_1 \in \mathbb{R}^{m\times r}$, $\sigma(\cdot)$ is applied elementwise and $\boldsymbol{x}(k)$ is assumed to contain a constant '1' as its last element to introduce a bias. These equations, describing the NN architecture, will be referred to from now on as forward equations.

An FNN with more complex architecture is shown in Fig. 3.1.2 and defined by the forward equations (3.1.3)-(3.1.4); here, L hidden layers are used, $\boldsymbol{h}^{(0)}(k) = \boldsymbol{x}(k)$ and $\boldsymbol{W}_l \in \mathbb{R}^{r_{l+1}\times r_l}$. When more than a few hidden layers are used, such models are referred to as Deep Neural Networks (DNN) or Deep Learning models.

$$\boldsymbol{h}^{(l)}(k) = \sigma\left(\boldsymbol{W}_{l-1}\boldsymbol{h}^{(l-1)}(k)\right), \forall l \geq 1 \tag{3.1.3}$$

$$\widehat{\boldsymbol{y}}(k) = \boldsymbol{W}_L\boldsymbol{h}^{(L)}(k) \tag{3.1.4}$$

For the case of most FNN models intended for regression the computation of the learnable parameters, $\boldsymbol{W}_l$, is performed by minimizing the MSE loss function over a training set, $\mathcal{L}_{MSE}^{(train)} = \sum_{j=1}^{k_{train}}\|\boldsymbol{e}(j)\|^2$, while tracking the MSE of a validation set, $\mathcal{L}_{MSE}^{(tval)} = \sum_{j=k_{train}+1}^{k_{val}}\|\boldsymbol{e}(j)\|^2$, as in Section 2.1.1. As FNNs (and NNs in general) are nonlinear models, the loss function is minimized using any of the iterative GD-based algorithms, with a vanilla version described in (3.1.5).

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha_i \frac{\partial L_{MSE}^{(train)}}{\partial \boldsymbol{\theta}} \tag{3.1.5}$$

where $\boldsymbol{\theta} = Vec([\boldsymbol{W}_0, \dots, \boldsymbol{W}_L])$ is the set of learnable parameters and α_i is the (possibly varying) learning rate at the ith iteration .

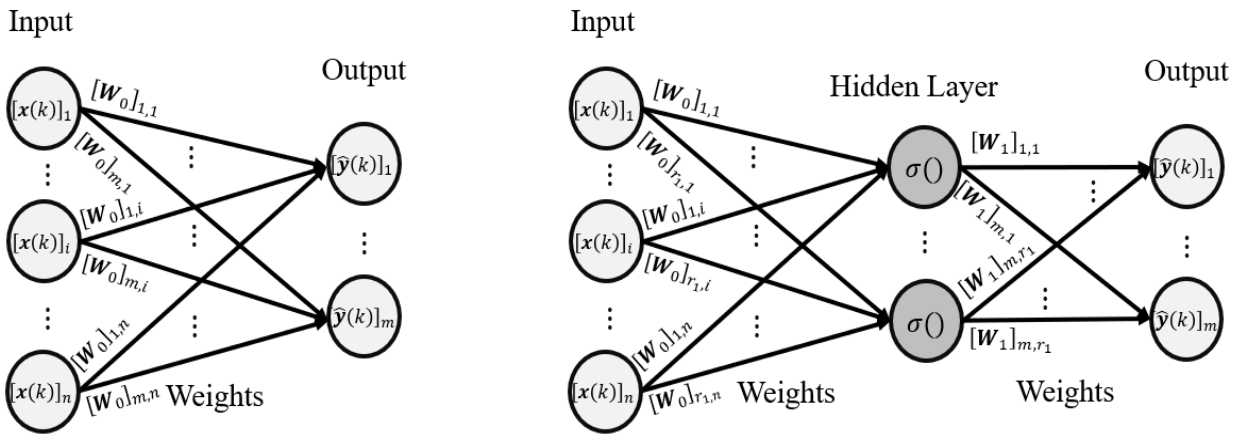


Fig. 3.1.1. Graphical representation of a Linear Model (left) and Neural Network with one hidden layer (right).

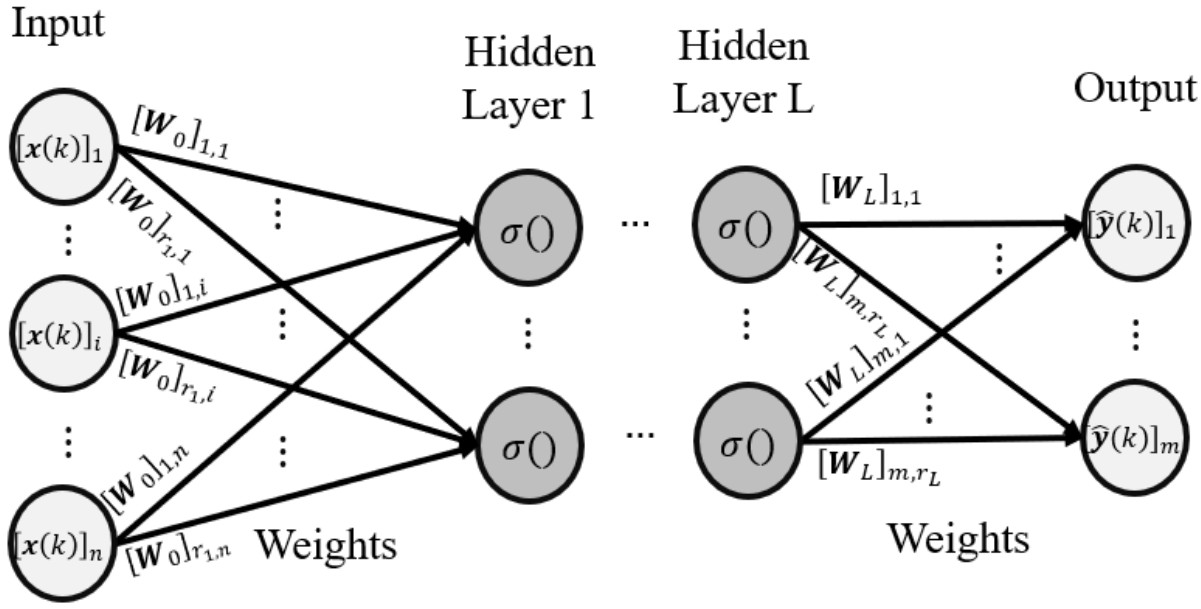


Fig. 3.1.2. Graphical representation of a Neural Network with L hidden layers.

When (3.1.5) is used (or one of its variations) on an NN model as defined in (3.1.3)-(3.1.4), the changes, $\partial \mathcal{L}_{MSE}^{(train)}/\partial \boldsymbol{W}$, in the front layers (rightmost in Fig. 3.1.2) influence the changes in the back layers (leftmost in Fig. 3.1.2), creating the well-known backpropagation effect which results from the chain-rule application to $\partial \mathcal{L}_{MSE}^{(train)}/\partial \boldsymbol{\theta}^{(p)}$. The associated BP equations for the architecture (3.1.3)-(3.1.4) (henceforth referred to as backward equations) are defined in (3.1.6)-(3.1.7).

$$\boldsymbol{\delta}_{k_0 k_f}^{(l)} = \left(\boldsymbol{W}_{l+1}^{T} \boldsymbol{\delta}_{k_0 k_f}^{(l+1)}\right) \circ \dot{\boldsymbol{H}}_{k_0 k_f}^{(l+1)} \tag{3.1.6}$$

$$\boldsymbol{W}_{l}^{(p+1)} = \boldsymbol{W}_{l}^{(p)} + \alpha_p \boldsymbol{\delta}_{k_0 k_f}^{(l)} \left(\boldsymbol{H}_{k_0 k_f}^{(l)}\right)^{T} \tag{3.1.7}$$

where $\boldsymbol{H}_{k_0 k_f}^{(l)} = \left[\boldsymbol{h}^{(l)}(k_0), \dots, \boldsymbol{h}^{(l)}(k_f)\right]$, $\dot{\boldsymbol{H}}_{k_0 k_f}^{(l)} = \left[\dot{\boldsymbol{h}}^{(l)}(k_0), \dots, \dot{\boldsymbol{h}}^{(l)}(k_f)\right]$, $\dot{\boldsymbol{h}}^{(l)}(k) = \dot{\sigma}\left(\boldsymbol{z}^{(l-1)}(k)\right)$, $\dot{\sigma}\left(\boldsymbol{z}^{(l)}(k)\right) = d\sigma\left(\boldsymbol{z}^{(l)}(k)\right)/d\boldsymbol{z}^{(l)}(k)$, $\boldsymbol{z}^{(l)}(k) = \boldsymbol{W}_l \boldsymbol{h}^{(l)}(k)$, $\boldsymbol{\delta}_{k_0 k_f}^{(l)} = \partial \mathcal{L}_{MSE}^{(train)}/\partial \boldsymbol{z}_{k_0 k_f}^{(l)}$; $\boldsymbol{\delta}_{k_0 k_f}^{(L)} = \left[\boldsymbol{e}(k_0), \dots, \boldsymbol{e}(k_f)\right]$, $\boldsymbol{H}_{k_0 k_f}^{(0)} = \left[\boldsymbol{x}(k_0), \dots, \boldsymbol{x}(k_f)\right]$; "$\circ$" denotes the Hadamard product (elementwise multiplication); and the term $\boldsymbol{\delta}_{k_0 k_f}^{(l)}$ is the so-called propagated error across the network, received at the lth hidden layer. A high-level graphical representation of the training process when using the forward and backward equations is shown in Fig. 3.1.3.

Even though an FNN can model some nonlinear components in the data, it still has limitations regarding learning time dependencies in a time series, since it does not consider the interactions between previous and current inputs [68]-[69]. A widely used approach to overcome this limitation in the context of time series is the augmented-input approach, $\boldsymbol{x}_{aug}(k) = Vec\left(\boldsymbol{X}_{k-j:k}\right)$, by which the previous j values in $\boldsymbol{x}(k)$ are directly introduced to the network in order to extract their interactions [68]-[69].

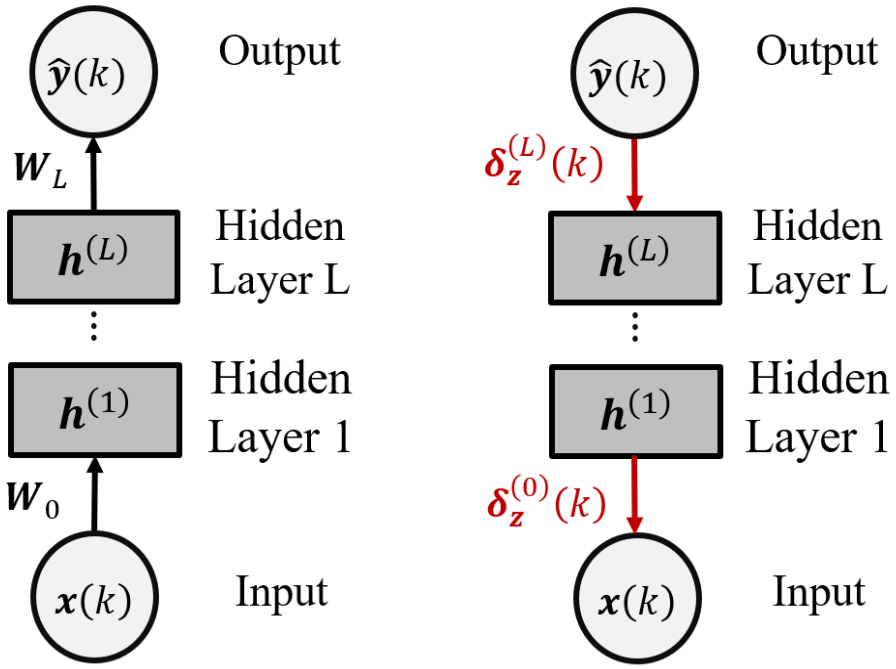


Fig. 3.1.3. Graphical representation of an FNN training process: forward pass (left) and backward pass (right).

One of the disadvantages of the augmented-input approach is the potential dimensionality increase in the layers' weights, as well as larger number of layers, needed for the FNN to extract temporal information. Furthermore, the increase in model size can also make the minimization process more complex due to the increase of the search space of the learnable parameters [70]. In addition, in this approach the time dependency of $\boldsymbol{y}(k)$ on the previous input information is limited to exactly the j previous time instances contained in $\boldsymbol{x}_{aug}(k)$. The previous process can also be interpreted as the FNN trying to model the mean of the output $\boldsymbol{y}(k)$ conditioned on previous inputs, i.e., $E\big(\boldsymbol{y}(k)\big|\boldsymbol{x}(k),..,\boldsymbol{x}(k-j)\big)$, by encoding it in its weights $\boldsymbol{W}_l$. This potentially requires a large number of layers when nonlinear complex time dependencies exist in the data, since the first few layers create a linear combination of the input elements.

One option to model nonlinear time dependencies in the area of time series, following an NN-based model, is the Recurrent Neural Network (RNN) model, a type of NN that tries to extend the limits of FNN models in capturing long-term dependencies. RNNs are characterized by using feedback connections within hidden layers through recurrent matrices, $\boldsymbol{U}_l \in \mathbb{R}^{r_l \times r_l}$, [71]-[72] (see Fig. 3.1.4). Through this connection, the output $\widehat{\boldsymbol{y}}(k)$ becomes dependent not only on the current input $\boldsymbol{x}(k)$ but also on the extracted information from previous input values $\boldsymbol{x}(k-j)$. The vanilla RNN forward equations of an L-hidden layer architecture, resulting from using (3.1.5) to minimize $\mathcal{L}_{MSE}^{(train)}$, are defined in (3.1.8)-(3.1.9).

$$\boldsymbol{h}^{(l)}(k) = \sigma\left(\boldsymbol{W}_{l-1}\boldsymbol{h}^{(l-1)}(k) + \boldsymbol{U}_l\boldsymbol{h}^{(l)}(k-1)\right), \forall l \geq 1 \tag{3.1.8}$$

$$\widehat{\boldsymbol{y}}(k) = \boldsymbol{W}_L\boldsymbol{h}^{(L)}(k) \tag{3.1.9}$$

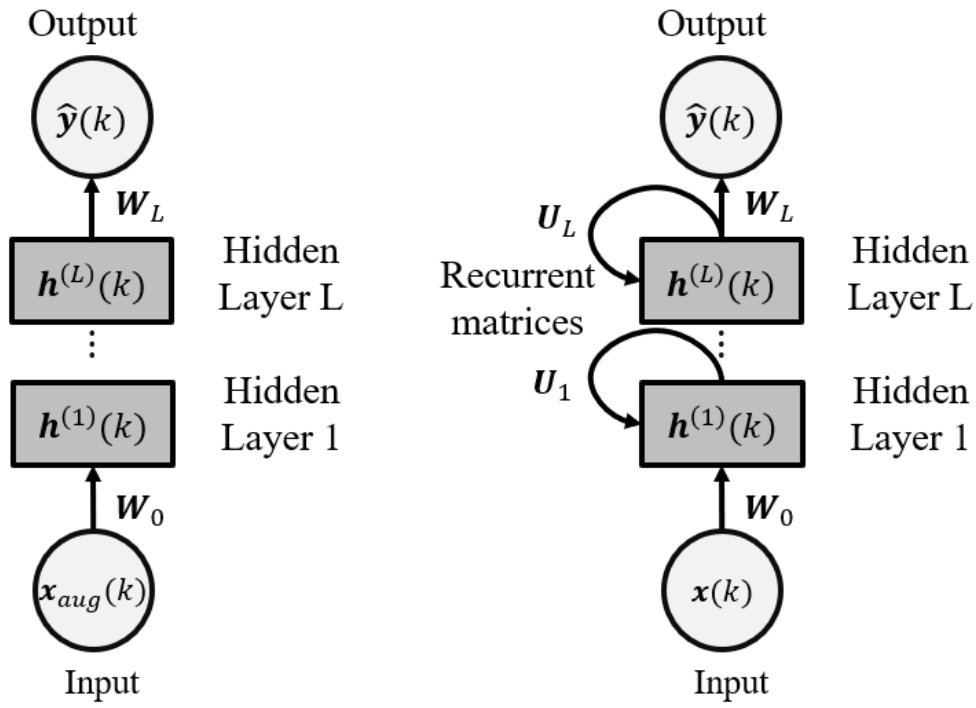


Fig. 3.1.4. Graphical representation of an FNN with augmented input (left) and RNN (right)

The vanilla RNN backward equations, resulting from using (3.1.5) to minimize $\mathcal{L}_{MSE}^{(train)}$, are described in (3.1.10)-(3.1.14), where $\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{h}^{(l)}(k)$, $\boldsymbol{\delta}_{\boldsymbol{h}}^{(L)}(k) = \boldsymbol{e}(k)$ and $\boldsymbol{h}^{(0)}(k) = \boldsymbol{x}(k)$.

$$\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k) = \boldsymbol{W}_l^T \left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l+1)}(k) \circ \dot{\boldsymbol{h}}^{(l+1)}(k)\right) + \boldsymbol{U}_l^T \left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k+1) \circ \dot{\boldsymbol{h}}^{(l)}(k+1)\right) \tag{3.1.10}$$

$$\boldsymbol{\delta}_{\boldsymbol{W}}^{(l)}(k) = \left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l+1)}(k) \circ \dot{\boldsymbol{h}}^{(l+1)}(k)\right) \boldsymbol{h}^{(l)}(k)^T \tag{3.1.11}$$

$$\boldsymbol{\delta}_{\boldsymbol{U}}^{(l)}(k) = \left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k+1) \circ \dot{\boldsymbol{h}}^{(l)}(k+1)\right) \boldsymbol{h}^{(l)}(k)^T \tag{3.1.12}$$

$$\boldsymbol{W}_l^{(p+1)} = \boldsymbol{W}_l^{(p)} + \alpha_p \sum_{j=k_0}^{k_f} \boldsymbol{\delta}_{\boldsymbol{W}}^{(l)}(j) \tag{3.1.13}$$

$$\boldsymbol{U}_l^{(p+1)} = \boldsymbol{U}_l^{(p)} + \alpha_p \sum_{j=k_0}^{k_f} \boldsymbol{\delta}_{\boldsymbol{W}}^{(l)}(j) \tag{3.1.14}$$

In the previous equations, (3.1.10) is the propagated error through time across the RNN, which is why backpropagation for RNN models is referred to as backpropagation through time (BPTT) [73].

RNN models have been successfully applied to problems in which nonlinear time dependencies need to be modeled accurately in forecasting settings [74]-[76]. However, when their vanilla architecture is used, (3.1.8)-(3.1.9), they experience practical limitations during their training phase, namely the Vanishing Gradient (VG) and Exploding Gradient (EG) problems [77]-[78]. These problems are linked to the magnitude of the gradients $\partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{W}_l^{(p)}$ used to update the weights in the NN. As its name suggests, the VG problem arises when the magnitude of the gradient is so small that changes in the weights become negligible during the training process, limiting the capability of the model to learn long-term dependencies and/or making the convergence extremely slow. On the other hand, the EG problem occurs whenever the magnitude

of the gradient becomes large enough to create large oscillations in the learnable parameters, $\boldsymbol{\theta}$, potentially leading to numerical instabilities.

In more depth, one of the main causes of the VG and EG problems is the consecutive matrix multiplication associated to the propagated gradients across layers [76], $\delta_{k_0 k_f}^{(l)}$, in multilayer FNNs and/or the propagated gradients through time in RNNs, $\delta_{\boldsymbol{h}}^{(l)}(k)$. These propagations can be observed in (3.1.15)-(3.1.16) where the propagated gradients' effects across, respectively, two layers and two-time instances, are explicitly shown.

$$\begin{aligned}\boldsymbol{\delta}_{k_0 k_f}^{(l)} &= \left(\boldsymbol{W}_{l+1}^T\left(\left(\boldsymbol{W}_{l+2}^T\boldsymbol{\delta}_{k_0 k_f}^{(l+2)}\right)\circ\dot{\boldsymbol{H}}_{k_0 k_f}^{(l+2)}\right)\right)\circ\dot{\boldsymbol{H}}_{k_0 k_f}^{(l+1)}\\ &= \left(\boldsymbol{W}_{l+1}^T\boldsymbol{W}_{l+2}^T\boldsymbol{\delta}_{k_0 k_f}^{(l+2)}\right)\circ\left(\boldsymbol{W}_{l+1}^T\dot{\boldsymbol{H}}_{k_0 k_f}^{(l+2)}\right)\circ\dot{\boldsymbol{H}}_{k_0 k_f}^{(l+1)}\end{aligned} \tag{3.1.15}$$

$$\begin{aligned}\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k) &= \boldsymbol{W}_l^T\left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l+1)}(k)\circ\dot{\boldsymbol{h}}^{(l+1)}(k)\right)\\ &\quad+\boldsymbol{U}_l^T\left(\left(\boldsymbol{W}_l^T\left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l+1)}(k+1)\circ\dot{\boldsymbol{h}}^{(l+1)}(k+1)\right)\right)\circ\dot{\boldsymbol{h}}^{(l)}(k+1)\right)\\ &\quad+\boldsymbol{U}_l^T\left(\left(\boldsymbol{U}_l^T\left(\boldsymbol{\delta}_{\boldsymbol{h}}^{(l)}(k+2)\circ\dot{\boldsymbol{h}}^{(l)}(k+2)\right)\right)\circ\dot{\boldsymbol{h}}^{(l)}(k+1)\right)\end{aligned} \tag{3.1.16}$$

From (3.1.15)-(3.1.16) it can be observed that since the eigenvalues of the matrices $\{\boldsymbol{W}_l, \boldsymbol{U}_l\}$ are not bounded, the consecutive products can lead to exponential growth or decay in their eigenvalues and consequently their elements. Also, in the context of VG, whenever the magnitude of the activation function's derivative, $|\dot{\sigma}(\cdot)|$, is less than 1 the matrices $\dot{\boldsymbol{H}}_{k_0 k_f}^{(l)}$ and vectors $\dot{\boldsymbol{h}}^{(l)}(k)$ will contain elements smaller than 1 in magnitude, decreasing the magnitude of the propagated gradients due to the element-wise multiplication and hence potentially promoting VG effects.

It is important to highlight that in RNNs, due to the recurrent connections, the gradient propagation across time, even when a single layer is used, becomes equivalent to that observed in an NN with several hidden layers, as seen in Fig. 3.1.5-Fig. 3.1.6, where the forward and backward components of the training process are depicted. This is sometimes referred to as RNN unrolling [79] and shows the similarity between RNNs and DNNs.

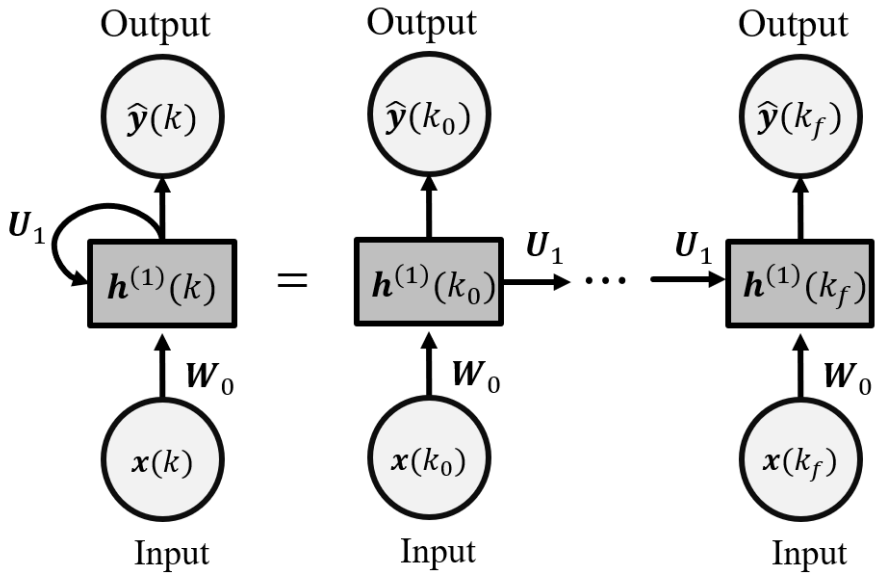


Fig. 3.1.5. High-level graphical representation of a one-hidden-layer RNN and its 'unrolled' equivalency during the forward part of the training process.

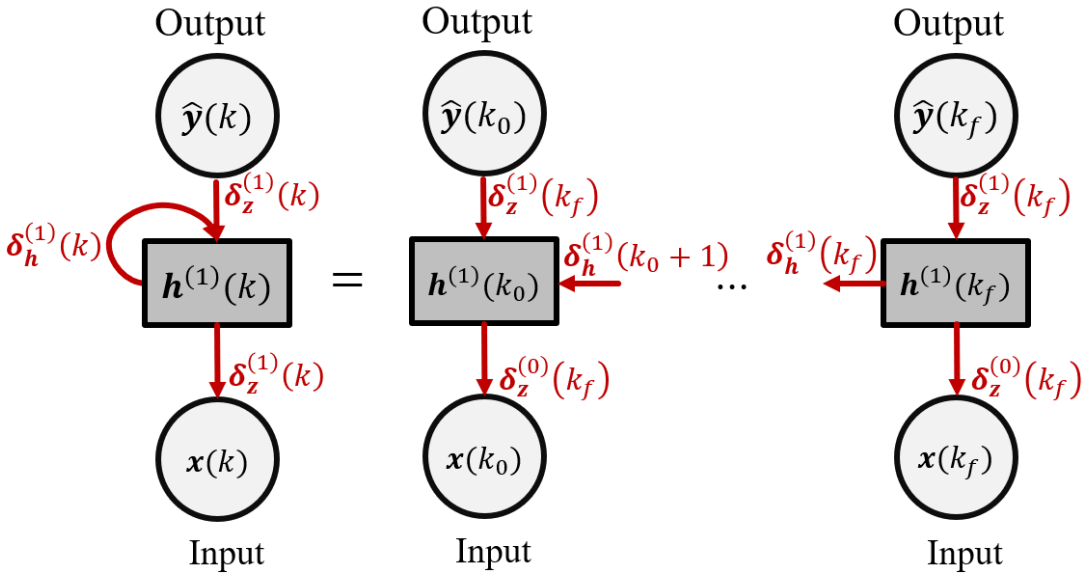


Fig. 3.1.6. High-level graphical representation of a one-hidden-layer RNN and its 'unrolled' equivalency during the backward part of the training process.

Due to the similarity between RNNs and DNNs in terms of the gradient propagation, practical measures are often taken to mitigate the VG and EG problems, one of which is to restrict the process of BPTT to relatively small time windows, a process known as Truncated Back Propagation Through Time (TBPTT) [80]-[81]. By implementing TBPTT, not only are the VG and EG problems diminished but also the computational overhead/auxiliary-memory associated to the BPTT (3.1.10)-(3.1.12), caused by the hidden-state related values ($\boldsymbol{h}^{(l)}(k)$, $\dot{\boldsymbol{h}}^{(l)}(k)$), is decreased [27].

3.2 LSTM

Even though TBPTT facilitates the use of vanilla RNN, the presence of VG and EG problems can limit its potential [27]. In order to overcome the previous limitations and improve the performance of RNNs, an architecture known as Long Short-Term Memory (LSTM) was introduced in the late 90s [27]-[29], standing out due to its potential to exploit long-term

dependencies and producing competitive results in a wide range of applications [30]-[34]. The practical success of LSTMs has been mostly attributed to its capability to mitigate the EG and VG problems [30], [82]. This capability is mostly associated to an 'internal' vector state $\boldsymbol{c}(k)$, often referred to as cell units, that partly depends on its own previous immediate value $\boldsymbol{c}(k-1)$.

Since the LSTM was first introduced variations with different levels of success have been proposed [82]-[83]; however, the most common single-layer LSTM architecture (3.2.1)- (3.2.6) can be mostly described by four different single-layer RNNs of equal dimensions, $\{\boldsymbol{a}(k), \boldsymbol{i}(k), \boldsymbol{f}(k), \boldsymbol{o}(k)\}$, referred to as 'gate units' and the previously mentioned internal state $\boldsymbol{c}(k)$. The gate units are interconnected in an element-wise fashion, they depend on the same hidden states, $\boldsymbol{h}(k-1)$, and are used to regulate the 'flow' of information across time in the network. Additionally, a linear relation is used to create the recursive temporal dependence in the cell units $\boldsymbol{c}(k)$, responsible for creating a flow of information from previous inputs, $\boldsymbol{x}(k-j)$, into the current output, $\boldsymbol{y}(k)$, during the training process [27]-[28].

$$\boldsymbol{a}(k) = \sigma_{th}(\boldsymbol{W_a x}(k) + \boldsymbol{U_a h}(k-1) + \boldsymbol{b}_a) \tag{3.2.1}$$

$$\boldsymbol{i}(k) = \sigma_{sig}(\boldsymbol{W_i x}(k) + \boldsymbol{U_i h}(k-1) + \boldsymbol{b_i}) \tag{3.2.2}$$

$$\boldsymbol{f}(k) = \sigma_{sig}\big(\boldsymbol{W_f x}(k) + \boldsymbol{U_f h}(k-1) + \boldsymbol{b_f}\big) \tag{3.2.3}$$

$$\boldsymbol{o}(k) = \sigma_{sig}(\boldsymbol{W_o x}(k) + \boldsymbol{U_o h}(k-1) + \boldsymbol{b_o}) \tag{3.2.4}$$

$$\boldsymbol{c}(k) = \boldsymbol{f}(k) \circ \boldsymbol{c}(k-1) + \boldsymbol{i}(k) \circ \boldsymbol{a}(k) \tag{3.2.5}$$

$$\boldsymbol{h}(k) = \boldsymbol{o}(k) \circ \sigma_{th}\big(\boldsymbol{c}(k)\big) \tag{3.2.6}$$

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{W}_y \boldsymbol{h}(k) \tag{3.2.7}$$

where $\sigma_{sig}(z) = 1/(1 + e^{-z})$ and $\sigma_{th}(z) = tanh(z)$ are element-wise functions, $\boldsymbol{W}_*$, $\boldsymbol{U}_*$, belong to $\mathbb{R}^{n_h \times n}$, $\mathbb{R}^{n_h \times n_h}$, respectively and n_h denotes the number of hidden units in the LSTM layer.

Regarding (3.2.2)-(3.2.4), the regulation of information due to the gates can be interpreted in the following manner. First, the input gate, $\boldsymbol{i}(k)$, gives a degree of relevance to the activation gate $\boldsymbol{a}(k)$. Then, the forget gate, $\boldsymbol{f}(k)$, determines how much past information contained in the previous cell units' values, $\boldsymbol{c}(k-1)$, will be carried into the current iteration. Finally, the output gate, $\boldsymbol{o}(k)$, dynamically scales the nonlinear transformed cell state, $\sigma_{th}\big(\boldsymbol{c}(k)\big)$. Each of these gates have their own associated input and feedback matrices, $\boldsymbol{W}_*$ $\boldsymbol{U}_*$. A graphical representation of the LSTM architecture is shown in Fig. 3.2.1.

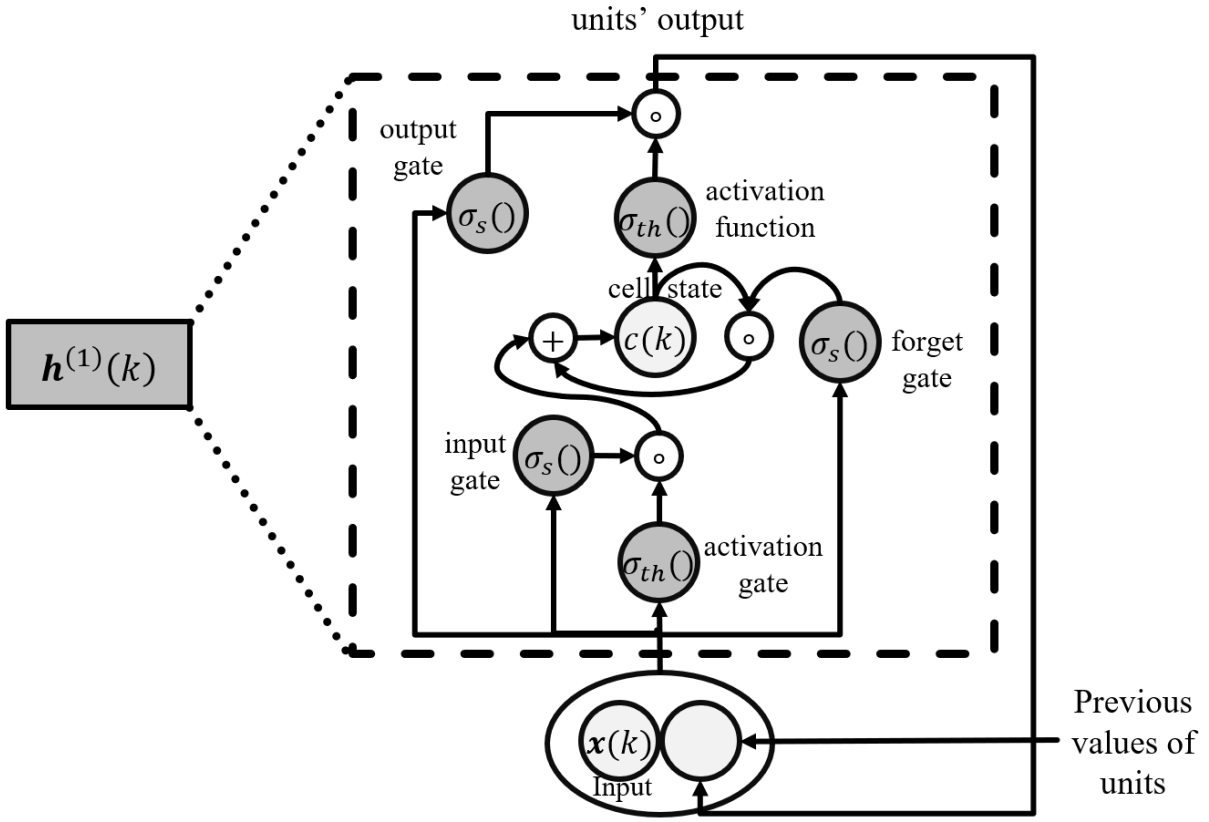


Fig. 3.2.1. A standard single-layer LSTM architecture, solid arrows represent matrix multiplication.

The backward equations resulting from implementing BP in the LSTM architecture, under the loss function $\mathcal{L}_{MSE}^{(train)}$, are expressed in (3.2.8)- (3.2.17).

$$\boldsymbol{\delta_h}(k) = \boldsymbol{W}_y^T \boldsymbol{e}(k) \tag{3.2.8}$$

$$\boldsymbol{\delta_o}(k) = \boldsymbol{\delta_h}(k) \circ \sigma_{th}\big(\boldsymbol{c}(k)\big) \tag{3.2.9}$$

$$\boldsymbol{\delta_c}(k) = \boldsymbol{\delta_h}(k) \circ \boldsymbol{o}(k) \circ \dot{\sigma}_{th}\big(\boldsymbol{c}(k)\big) + \boldsymbol{\delta_c}(k+1) \circ \boldsymbol{f}(k+1) \tag{3.2.10}$$

$$\boldsymbol{\delta_i}(k) = \boldsymbol{\delta_c}(k) \circ \boldsymbol{a}(k) \tag{3.2.11}$$

$$\boldsymbol{\delta_f}(k) = \boldsymbol{\delta_c}(k) \circ \boldsymbol{c}(k-1) \tag{3.2.12}$$

$$\boldsymbol{\delta_a}(k) = \boldsymbol{\delta_c}(k) \circ \boldsymbol{i}(k) \tag{3.2.13}$$

$$\boldsymbol{\delta_z}(k) = \boldsymbol{\delta_\sigma}(k) \circ \dot{\boldsymbol{\sigma}}\,(k) \tag{3.2.14}$$

$$\boldsymbol{\delta_{x_h}}(k) = \boldsymbol{V}^T \boldsymbol{\delta_z}(k) \tag{3.2.15}$$

$$\boldsymbol{\delta_V}(k) = \boldsymbol{\delta_z}(k) \boldsymbol{x_h}(k) \tag{3.2.16}$$

$$\boldsymbol{V} := \boldsymbol{V} + \alpha_p \sum_{j=k_0}^{k_f} \boldsymbol{\delta_V}(j) \tag{3.2.17}$$

where $\boldsymbol{\delta_h}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{h}(k)$, $\boldsymbol{\delta_o}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{o}(k)$, $\boldsymbol{\delta_c}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{c}(k)$, $\boldsymbol{\delta_a}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{a}(k)$, $\boldsymbol{\delta_i}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{i}(k)$, $\boldsymbol{\delta_z}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{z}(k)$, $\boldsymbol{\delta_z}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{x_h}(k)$, $\boldsymbol{\delta_V}(k) = \partial \mathcal{L}_{MSE}^{(train)} / \partial \boldsymbol{V}(k)$, $\boldsymbol{z}(k) = \boldsymbol{V}\boldsymbol{x_h}(k)$, $\boldsymbol{x_h}(k) = \begin{bmatrix} \boldsymbol{x}(k) \\ \boldsymbol{h}(k-1) \end{bmatrix}$, $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_a \\ \boldsymbol{V}_i \\ \boldsymbol{V}_f \\ \boldsymbol{V}_o \end{bmatrix}$, $\boldsymbol{V}_* = [\boldsymbol{W}_*, \boldsymbol{U}_*]$, $\boldsymbol{\delta_\sigma}(k) = \begin{bmatrix} \boldsymbol{\delta_a}(k) \\ \boldsymbol{\delta_i}(k) \\ \boldsymbol{\delta_f}(k) \\ \boldsymbol{\delta_o}(k) \end{bmatrix}$, $\dot{\boldsymbol{\sigma}}\,(k) = \begin{bmatrix} \dot{\sigma}_{th}\big(\boldsymbol{V}_a \boldsymbol{I}(k)\big) \\ \dot{\sigma}_{sig}\big(\boldsymbol{V}_i \boldsymbol{I}(k)\big) \\ \dot{\sigma}_{sig}\left(\boldsymbol{V}_f \boldsymbol{I}(k)\right) \\ \dot{\sigma}_{sig}\big(\boldsymbol{V}_o \boldsymbol{I}(k)\big) \end{bmatrix}$.

As observed in (3.2.10) the propagated error to the internal state $\boldsymbol{c}(k)$ has two components, the propagated error from the cell's output $\boldsymbol{\delta_h}(k)$ and the propagated error from the next state's value $\boldsymbol{\delta_c}(k+1)$. The latter component is referred to in the literature as the constant error carousel, since $\boldsymbol{\delta_c}(k+1)$ is multiplied by the forget gate value $\boldsymbol{f}(k+1)$ and no additional unconstrained matrix is present in this part of the operation. This process is one of the main reasons why this architecture can significantly mitigate the EG problem compared to vanilla RNNs, since the forget gate regulates the degree to which the incoming gradient is forgotten through this operation. In more detail, since $\mathbf{0}_{n_h \times 1} \leq \boldsymbol{f}(k) \leq \mathbf{1}_{n_h \times 1}$, the operation will not contribute to the generation of EG. It is important to highlight that even though LSTM architectures are more appropriate to handle gradient implementation problems, TBPTT is still used in practice during the training phase to further handle the VG and EG problems.

As expressed before, variations of LSTM have arisen through the years such as bidirectional LSTMs [82] and LSTM with peepholes [28], where the latter variation replaces the term $\boldsymbol{h}(k-1)$ by $\boldsymbol{c}(k-1)$ in (3.1.1)-(3.1.4), allowing the gates to have direct access to the constant error carousel when BP is implemented. Also, a simplified and very popular version of the LSTM architecture was proposed in [28], known as Gated Recurrent Unit (GRU), which uses one less gate than the LSTM, decreasing the number of parameters. These architectures have shown similar performance to the LSTM for various applications [30]-[34].

One of the main challenges when using LSTM networks is their complexity in terms of the number of parameters needed in their structure, which implies the need for more computational power to use them as well as the need for Graphics Processing Units (GPUs) to avoid slow training times [84]-[86]. Also, as is the case for most NN-based models, LSTMs are still considered black-box models due to their capability to approximate a wide range of functions but with low

interpretability when compared to LMs. For instance, two NN models with the same topology but significantly different learnable parameter values can generate similar results [87]-[88]. Consequently, there is not a simple link between the parameter values in these models and the function being approximated.

Chapter 4

E-LSTM: Extended LSTM

In this chapter an extended LSTM architecture designed to facilitate capturing long-term dependencies is proposed, named Extended Long-Short Term Memory (ELSTM). This extension is performed by explicitly increasing the connectivity of the states in the LSTM, $\boldsymbol{c}(k)$, to their own values at a lag of p time units, $\boldsymbol{c}(k-p)$. The increased connectivity in the E-LSTM aims to reduce the number of parameters in relation to the LSTM network, while achieving a similar performance to an LSTM model. In addition to describing the E-LSTM architecture, a performance comparison with the LSTM network and alternative models is provided, including the number of parameters needed in each model, through simulations using synthetic and real-world time-series data. It is important to clarify that a large amount of the material in this chapter has also been published in [89], where the results are more compactly presented.

4.1 E-LSTM Architecture

4.1.1 Motivation and conceptualization

As discussed in Chapter 3, LSTMs have produced competitive results in a wide range of applications [30]-[37], but at the price of using a large number of parameters in their architectures

[90]-[91]. For instance, in datasets containing short-term and long-term dependencies, LSTM networks might require on the order of several thousands of parameters [84]-[85] to extract the information about both dependence types, due to the temporal-explicit connectivity in the LSTM architecture being only immediate. For cases in which information about time dependencies can be obtained using statistical techniques, the temporal-explicit connectivity of standard LSTMs might create an inherent barrier to extract this information efficiently, which could be one of the potential causes behind the need for a large number of parameters to achieve an acceptable performance.

In previous years, approaches have been proposed to exploit long-term dependencies, some of which encompass stacked layers of NNs, RNNs or LSTMs designed to handle a variety of datasets [92]–[95] or to solve specific practical problems [96]-[97]. However, none of the strategies modify the inner mechanism of the LSTMs in terms of the temporal connectivity, leaving them susceptible to using a large number of parameters.

Clockwork RNN (CW-RNN), another well-established approach in the RNN field [98]–[100], is a model designed to decrease the number of parameters in its architecture, by reducing the connectivity between hidden units and dividing the network into modules that activate at different frequencies. For some datasets, this approach has shown to generate competitive results when compared to LSTM networks, while noticeably reducing the number of parameters.

In the context of exploiting statistical information more efficiently, an extension to the LSTM architecture is proposed, named Extended Long-Short Term Memory (E-LSTM). This architecture focuses on extending the temporal-explicit connectivity. The E-LSTM is designed to facilitate capturing long-term dependencies, under the assumption that the temporal location of those dependencies is known or that it can be estimated during data preprocessing, a more practical

assumption used in this chapter. The addition of the dependence information is performed by connecting the cell states in the LSTM, $\boldsymbol{c}(k)$, to their own value with p lags, $\boldsymbol{c}(k-p)$.

The increased temporal connectivity in the E-LSTM (as opposed to the reduced connectivity in the CW-RNN approach) aims to reduce the number of parameters while achieving a similar performance to an LSTM model. In this regard, an approach increasing the connectivity within the LSTM architecture has been previously proposed [93], but the modification was limited to linearly connect the current and immediately previous LSTM outputs.

4.1.2 Forward equations and conceptualization

As mentioned in Section 3.2, the propagated error to the cell states $\boldsymbol{c}(k)$ (3.2.10) is caused by two other propagated-error components: the LSTM output $\boldsymbol{\delta_h}(k)$ and the next state's value $\boldsymbol{\delta_c}(k+1)$. The latter component is of special relevance, since its effect is regulated by the forget gate $\boldsymbol{f}(k)$, controlling how much the incoming propagated error is diminished. Therefore, when an LSTM is used to model long-term dependencies, it can be challenging to identify to what extent it considers the effect of distant past values,

Even though $\boldsymbol{f}(k)$ mitigates the EG problem it can also exponentially decrease the effect of long-term dependencies, specifically, the effect of a previous cell state $\boldsymbol{c}(k-p)$ into the current state $\boldsymbol{c}(k)$. The latter can be observed in the first term on the right-hand side of (4.1.1), resulting from implementing backward substitution for (3.2.5).

$$\boldsymbol{c}(k) = \boldsymbol{c}(k-p) \circ \prod_{j=0}^{p-1} \boldsymbol{f}(k-j)$$

$$+\boldsymbol{i}(k) \circ \boldsymbol{a}(k) + \sum_{j=1}^{p-1} \left(\boldsymbol{i}(k-j) \circ \boldsymbol{a}(k-j) \circ \prod_{i=0}^{j-1} \boldsymbol{f}(k-i) \right) \tag{4.1.1}$$

Although forgetting previous information can be useful to prevent large accumulations from being created, due to the sum involving the terms of the form $\boldsymbol{i}(k-j) \circ \boldsymbol{a}(k-j)$, this could potentially erase relevant long-term information. The latter effect is more likely to occur when few hidden units are used, since the dimension of the forget gate vector is equal to the number of hidden units, n_h. Therefore, with smaller n_h it is less likely to have sufficiently large forget-gate values (close to one) in the multiplicative effect. The latter situation can imply that having more units (and consequently a larger number of parameters) might allow long-term dependencies to be captured in an LSTM network.

As an initial approach to oppose the exponentially decreasing effect of previous cell states, $\boldsymbol{c}(k-p)$, the recursive dependence defining the cell state (3.2.5) could be modified by creating a direct connection to a previous value, as expressed in (4.1.2). This modification could improve preserving/capturing long-term dependencies across a time series. In (4.1.2) a new additional forget gate, $\boldsymbol{f}_p(k)$, is used to dynamically regulate the effect of past information carried out by $\boldsymbol{c}(k-p)$ and follows the same mathematical structure defined by the original gates, as seen in (4.1.3).

$$\boldsymbol{c}(k) = \boldsymbol{f}(k) \circ \boldsymbol{c}(k-1) + \boldsymbol{f}_p(k) \circ \boldsymbol{c}(k-p) + \boldsymbol{i}(k) \circ \boldsymbol{a}(k) \tag{4.1.2}$$

$$\boldsymbol{f}_p(k) = \sigma_{sig}\left(\boldsymbol{W}_{f_p}\boldsymbol{x}(k) + \boldsymbol{U}_{f_p}\boldsymbol{h}(k-1)\right) \tag{4.1.3}$$

Although the term $\boldsymbol{f}_p(k) \circ \boldsymbol{c}(k-p)$ in (4.1.2) seems an appropriate generalization of the recursive equation in (3.2.5), it increases the likelihood of saturation in the hidden units, $\boldsymbol{h}(k) = \sigma_{sig}(\boldsymbol{c}(k))$. This potential saturation is linked to exponential growth in the values of $\boldsymbol{c}(k)$, which can occur when the sum of $\boldsymbol{f}(k)$ and $\boldsymbol{f}_p(k)$ is consistently greater than one across consecutive forward iterations. The latter cause can be eliminated by replacing both forget gates by constrained

versions, $\hat{\boldsymbol{f}}(k)$ and $\hat{\boldsymbol{f}}_p(k)$, as defined in (4.1.4) and resulting in the cell-state equation (4.1.5) that will be used as the core of the E-LSTM network.

$$\boldsymbol{c}(k) = \hat{\boldsymbol{f}}(k) \circ \boldsymbol{c}(k-1) + \hat{\boldsymbol{f}}_p(k) \circ \boldsymbol{c}(k-p) + \boldsymbol{i}(k) \circ \boldsymbol{a}(k) \tag{4.1.4}$$

$$\hat{\boldsymbol{f}}(k) + \hat{\boldsymbol{f}}_p(k) \leq \mathbf{1}_{n_h \times 1} \tag{4.1.5}$$

where $\hat{\boldsymbol{f}}(k) = \boldsymbol{f}(k) \circ \boldsymbol{w}_f(k)$, $\hat{\boldsymbol{f}}_p(k) = \boldsymbol{f}_p(k) \circ \boldsymbol{w}_{f_p}(k)$, and $\boldsymbol{w}_f(k)$, $\boldsymbol{w}_{f_p}(k)$ are n_h-dimensional dynamic normalizing factors, satisfying the constraints in (4.1.6)-(4.1.7).

$$\boldsymbol{w}_f(k), \boldsymbol{w}_{f_p}(k) \leq \mathbf{1}_{n_h \times 1} \tag{4.1.6}$$

$$\boldsymbol{w}_f(k) + \boldsymbol{w}_{f_p}(k) = \mathbf{1}_{n_h \times 1} \tag{4.1.7}$$

Among a variety of candidates for the normalizing factors $\boldsymbol{w}_f(k)$ and $\boldsymbol{w}_{f_p}(k)$ (constant functions, linear functions, neural networks, etc.), a simple normalization (4.1.8)-(4.1.9) using both forget gates is used. This normalization not only avoids additional parameters in the E-LSTM network but also causes the following two useful effects: competition for transmitting relevant information between short-term and long-term relations is directly promoted and the forget gate values directly influence each other during the training process, as will be seen in the next section.

$$\boldsymbol{w}_f(k) = \frac{\boldsymbol{f}(k)}{\boldsymbol{f}(k) + \boldsymbol{f}_p(k)} \tag{4.1.8}$$

$$\boldsymbol{w}_{f_p}(k) = \frac{\boldsymbol{f}_p(k)}{\boldsymbol{f}(k) + \boldsymbol{f}_p(k)} \tag{4.1.9}$$

where the divisions are elementwise.

Through the explicit temporal connectivity of the E-LSTM architecture, (4.1.4)-(4.1.9), the likelihood of forgetting relevant information in the distant past can be decreased by creating a 'bridge' to it. A high-level representation of this process is depicted in Fig. 4.1.1.

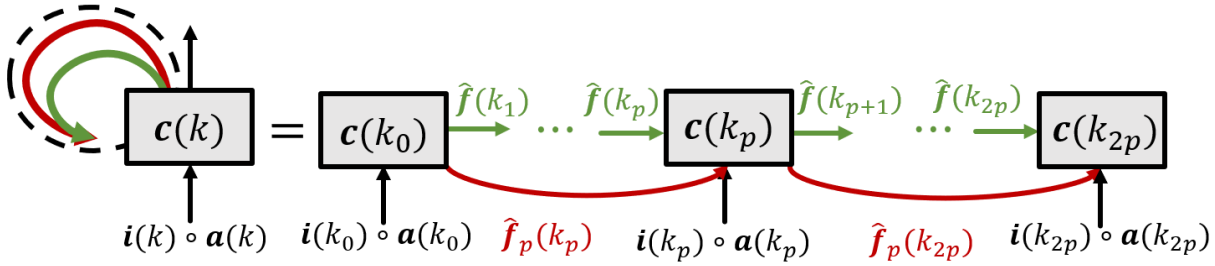


Fig. 4.1.1. Proposed E-LSTM network when "unrolled" through $2p + 1$ iterations.

4.1.3 Backward equations and analysis

Since the cell-state equation is different in the E-LSTM and an additional gate is used, existing BP will be modified (4.1.10)-(4.1.11) and a new BP is generated (4.1.12), which are linked to the forward equations described in (4.1.4)-(4.1.9).

$$\boldsymbol{\delta_c}(k) = \boldsymbol{\delta_h}(\boldsymbol{k}) \circ \boldsymbol{o}(k) \circ \dot{\sigma}_{th}\big(\boldsymbol{c}(k)\big) + \boldsymbol{\delta_c}(k+1) \circ \hat{\boldsymbol{f}}(k+1) + \boldsymbol{\delta_c}(k+p) \circ \hat{\boldsymbol{f}}_p(k+p) \quad (4.1.10)$$

$$\boldsymbol{\delta_f}(k) = \boldsymbol{\delta_c}(k) \circ \Big(\Big(2\boldsymbol{w}_f(k) - \boldsymbol{w}_f^2(k)\Big) \circ \boldsymbol{c}(k-1) - \boldsymbol{w}_{f_p}^2(k) \circ \boldsymbol{c}(k-p)\Big) \quad (4.1.11)$$

$$\boldsymbol{\delta}_{\boldsymbol{f}_p}(k) = \boldsymbol{\delta_c}(k) \circ \Big(\Big(2\boldsymbol{w}_{f_p}(k) - \boldsymbol{w}_{f_p}^2(k)\Big) \circ \boldsymbol{c}(k-p) - \boldsymbol{w}_f^2(k) \circ \boldsymbol{c}(k-1)\Big) \quad (4.1.12)$$

where (4.1.10) and (4.1.11) replace (3.2.10) and (3.2.12), respectively and (4.1.12) is used to update the learnable parameters of the new gate $f_p(k)$, i.e., $\boldsymbol{V}_{f_p}$. The power operation is applied element-wise in (4.1.11)-(4.1.12).

The remaining backpropagation equations linked to the LSTM remain the same for the E-LSTM, but the matrices $\boldsymbol{V}$, $\boldsymbol{\delta_\sigma}(k)$ and $\dot{\boldsymbol{\sigma}}\ (k)$, defined in Section 3.2, are modified as indicated next:

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_a \\ \boldsymbol{V}_i \\ \boldsymbol{V}_f \\ \boldsymbol{V}_{f_p} \\ \boldsymbol{V}_o \end{bmatrix}, \boldsymbol{\delta_\sigma}(k) = \begin{bmatrix} \boldsymbol{\delta_a}(k) \\ \boldsymbol{\delta_i}(k) \\ \boldsymbol{\delta_f}(k) \\ \boldsymbol{\delta_{f_p}}(k) \\ \boldsymbol{\delta_o}(k) \end{bmatrix} \text{ and } \dot{\boldsymbol{\sigma}}\ (k) = \begin{bmatrix} \dot{\sigma}_{th}\big(\boldsymbol{V}_a \boldsymbol{I}(k)\big) \\ \dot{\sigma}_{sig}\big(\boldsymbol{V}_i \boldsymbol{I}(k)\big) \\ \dot{\sigma}_{sig}\left(\boldsymbol{V}_f \boldsymbol{I}(k)\right) \\ \dot{\sigma}_{sig}\left(\boldsymbol{V}_{f_p} \boldsymbol{I}(k)\right) \\ \dot{\sigma}_{sig}\big(\boldsymbol{V}_o \boldsymbol{I}(k)\big) \end{bmatrix}.$$

When examining the propagated error (4.1.11), it is possible to verify that even when $\boldsymbol{f}(k)$ is close to zero the magnitude of $\boldsymbol{\delta_f}(k)$ might not necessarily reduce, since it is influenced by $\boldsymbol{w}_{f_p}^2(k)$, and consequently by $\boldsymbol{f}(k)$; an analogous situation occurs for $\boldsymbol{\delta_{f_p}}(k)$ in (4.1.12). In practical terms, this situation is desirable during the training process since it can aid $\boldsymbol{f}(k)$ and $\boldsymbol{f}_p(k)$ in avoiding local optima when they have transitory near-zero values.

4.1.4 Overhead analysis and training implementation

The number of learnable parameters in a single-layer E-LSTM network is expressed in (4.1.13).

$$n_{\boldsymbol{\theta}}^{(E-LSTM)} = 5\big(n_h + n_{input} + 1\big)n_h \tag{4.1.13}$$

The training of the E-LSTM is similar to that of the LSTM. However, due to the modified relationship in (4.1.4), the values associated with previous internal states, $\boldsymbol{c}(k-1)$, ..., $\boldsymbol{c}(k-p)$, used during forward and backward passes in BP need to be either explicitly stored or re-computed. In the first case, storing up to p previous values would imply an increase in the memory needed for the BP implementation, compared to a standard LSTM. Specifically, if the number of forward

iterations is equal to the sequence length, $\mathcal{S}$, then the total number of auxiliary variables used to store previous values of $\boldsymbol{c}(k)$ during the forward pass of the E-LSTM training is as expressed in (4.1.14).

$$n_{\overrightarrow{\Phi}}^{(GI-LSTM)} = \mathcal{S}n_h \tag{4.1.14}$$

If memory becomes a constraint for the BP implementation, due to large sequence lengths or a large number of hidden units, it may be advantageous to re-compute previous values of the cell state, changing the number of auxiliary variables to be that in (4.1.15).

$$n_{\overrightarrow{\Phi}}^{(GI-LSTM)} = pn_h \tag{4.1.15}$$

For the backward pass the overhead analysis can be performed similar to the forward pass. The resulting memory usage due to auxiliary internal states, $n_{\overleftarrow{\Phi}}^{(GI-LSTM)}$, is as described in (4.1.16) for the computation prioritization approach.

$$n_{\overleftarrow{\Phi}}^{(GI-LSTM)} = \mathcal{S}n_h \tag{4.1.16}$$

For the memory prioritization approach, the resulting auxiliary internal states is indicated by (4.1.17).

$$n_{\overleftarrow{\Phi}}^{(GI-LSTM)} = pn_h \tag{4.1.17}$$

For the training process, a validation set is used as one of the stopping criteria, specifically, using a threshold n_{fails} for the maximum number of consecutive fails on decreasing the validation-set loss function, $\mathcal{L}^{(val)}$, as defined in Algorithm 4.1.

Algorithm 4.1: E-LSTM training

Input: $\{(\boldsymbol{x}(1), \boldsymbol{y}(1)), \ldots, (\boldsymbol{x}(n), \boldsymbol{y}(n))\}$ //assumed to be normalized//

Set values:

n_h //number of hidden neurons

n_{ss} //number of subsequences

ss_{length} //training subsequence length

I_{max} //maximum number of iterations before stopping

n_{fails} //number of consecutive fails

$n_{fails-max}$ //max number of consecutive fails before stopping

$E_{train-min} \leftarrow inf$ //minimum training MSE

$E_{val-min} \leftarrow inf$ //minimum validation MSE

Initialize: $\boldsymbol{\theta}$ //random initialization of weights

Divide dataset: D_{train}, D_{val}, D_{test} //division keeping temporal order

for j = 1 to I_{max}

for l = 1 to n_{ss}

//----------Forward pass----------//

Extract: $D_{train}^{(l)}$ //Extract lth training subsequence from D_{train}

for k = 1 to ss_{length}

Compute: $\boldsymbol{a}(k), \boldsymbol{i}(k), \boldsymbol{w}_f(k), \boldsymbol{w}_{f_p}(k)\ \hat{\boldsymbol{f}}(k), \hat{\boldsymbol{f}}_p(k), \boldsymbol{o}(k), \boldsymbol{c}(k), \boldsymbol{h}(k)$

Compute: E_{train} // using $\mathcal{L}^{(train)}$ for the lth subsequence

//----------Backward pass----------//

for k = 1 to ss_{length}

Compute: $\boldsymbol{\delta_h}(k)$, $\boldsymbol{\delta_o}(k)$, $\boldsymbol{\delta_c}(k)$, $\boldsymbol{\delta_i}(k)$, $\boldsymbol{\delta_f}(k)$, $\boldsymbol{\delta}_{f_p}(k), \boldsymbol{\delta_a}(k)$, $\boldsymbol{\delta_z}(k)$, $\boldsymbol{\delta_z}(k)$, $\boldsymbol{\delta_\theta}(k)$

$\boldsymbol{\delta_\theta} \leftarrow \sum_{j=1}^{ss_{length}} \boldsymbol{\delta_\theta}(j)$

Update: $\boldsymbol{\theta}$ //using any optimizer designed for this purpose

//----------Validation Set Performance----------//

Compute: E_{val}//using D_{val} and $\mathcal{L}^{(val)}$while keeping temporal order

//----------Stopping criteria----------//

if $E_{val-min} > E_{val}$ **and** $E_{train-min} > E_{train}$**:** //Storing best performance and optimal parameters//

$\boldsymbol{\theta}_{opt} \leftarrow \boldsymbol{\theta}$, $E_{val-min} \leftarrow E_{val}$, $n_{fails} \leftarrow 0$

else:

$n_{fails} \leftarrow n_{fails} + 1$

if $n_{fails} > n_{fails-max}$://Maximum consecutive fails for reducing $E_{val-min}$ or $E_{train-min}$

break //ending main for loop

return: $\boldsymbol{\theta}_{opt}$ //Maximum number of iterations reached//

4.1.5 Lag dependence selection using the distance correlation

As indicated in Section 4.1.4, the relevance given to past information in (4.1.1) depends on the decay-rate variety in the elements of the forget gate, $\boldsymbol{f}(k)$, a situation mitigated in the E-LSTM architecture. However, due to the E-LSTM using two forget gates, if elements in the first of these are large enough to incorporate partial information of $\boldsymbol{c}(k-p)$, redundancy can occur. Consequently, appropriate selection for the lag value p is desirable. This selection process might be challenging when using techniques that rely on mathematical correlations (linear relations), since NN models focused on exploiting nonlinearities and such techniques would not be suited to spot the nonlinear relations in datasets.

Several selection techniques could be used to try to identify a value for p [103]–[105]. However, they are prone to miss nonlinear effects. Therefore, a hybrid approach for the lag selection designed, to consider the nonlinear nature of RNN networks, is employed. This is performed by combining a filter method and the distance correlation (DC) measure [106]–[108], a statistical measure used to identify statistical relations, not only linear relations, between paired multivariate variables.

The proposed hybrid approach is composed of two parts. First, an auxiliary linear regression model is constructed: $\widehat{\boldsymbol{y}}_{DC}(k) = \boldsymbol{\theta}_{DC}\boldsymbol{x}_{DC}(k)$, where $\boldsymbol{\theta}_{DC} \in \mathbb{R}^{1\times k_i}$ and $\boldsymbol{x}_{DC}(k) = Vec\left(\boldsymbol{X}_{k-k_i+1:k}\right)$ is an augmented input composed of the previous k_i input values. Using this model, linear relations between the desired output, $\boldsymbol{y}(k)$, and current-and-previous inputs, $\boldsymbol{X}_{k-k_i+1:k}$, are then removed by computing the residual values, i.e., $\boldsymbol{e}_{DC}(k) = \boldsymbol{y}(k) - \boldsymbol{\theta}_{DC}\boldsymbol{x}_{DC}(k)$. Second, a filter method is employed by computing the DC value of the paired residuals and lagged input

values, i.e., $\boldsymbol{e}_{DC}(k)$, $\boldsymbol{x}(k-j)$, for each lag $j \in \{0, \dots, k_i\}$, from which relevant nonlinear effects can be identified. The lag with the highest DC value is selected as the value for p.

4.2 Experimental Setup

To evaluate the performance and efficiency (in terms of the number of parameters) of the proposed E-LSTM network in relation to the standard LSTM, experiments using univariate synthetic and real time-series are performed. Additionally, the CW-RNN network and the Seasonal Auto Regressive Integrated Moving Average (SARIMA) model, a well-known linear model designed for data with seasonalities [101]-[102], are used as additional baselines to compare with the E-LSTM model. The CW-RNN is selected model due to its sparse structure approach to exploit long-term dependencies while reducing the number of parameters per unit.

4.2.1 Augmented-input Networks

One of the main purposes of an RNN is to identify relations among previous inputs, immediate and far in the past. However, simple linear input-output relationships, i.e., $x(k-p)$, $y(k)$, could require a large number of parameters for an RNN model, due to the several consecutive nonlinear operations employed within the model. As indicated in Section 3.1, directly presenting lagged input values into an NN model is a practice that can be used when handling time series to exploit possible relations [109]-[110]. Therefore, in order to assess the practical usefulness of the increased connectivity in the E-LSTM, the augmented-input approach is implemented in the CW-RNN, the standard LSTM and the proposed E-LSTM, resulting in augmented versions of these models, denoted as CW-RNN-A, LSTM-A and E-LSTM-A.

By using the augmented-input models, it might be possible to ease the need for a large number of parameters to learn linear relations, due to the linear transformation the augmented input goes through in the gate equations in (3.1.1)-(3.1.5) and (4.1.3). The augmentation is performed by adding a lagged input value, $\boldsymbol{x}(k-k_i)$, selected based on a simple correlation analysis, from which the augmented input is constructed $\boldsymbol{x}_{aug}(k) = [\boldsymbol{x}(k), \boldsymbol{x}(k-k_i)]$, where only one lag is selected to avoid potential redundancy caused by the recurrence relations in RNNs.

4.2.2 SARIMA implementation details

The selection for the SARIMA model hyperparameters and parameters is carried out in two stages for each dataset used. First, using the training set, a correlation analysis between the input and its lagged values is performed, using prior information associated with the maximum seasonality in each dataset to create an upper bound for a maximum lag. From the previous analysis, the highest (absolute) correlation value is selected for the seasonal components of the model (SA, MA). In the second stage, using the SARIMA performance over the validation set in each dataset, a search for appropriate values for the autoregressive, moving average and integrative components is carried out, varying values from one up to the seasonal values obtained in the first stage (SA, MA), turning on and off the presence of the integrative component. It is important to clarify that, even though it is possible to extract these linear relations first and then use the RNN models over the residuals, this approach is not used in this work, since the main goal across the experiments is to identify the capability of the networks, especially the proposed E-LSTM, as standalone models.

4.2.3 Synthetic datasets

Synthetic datasets are employed to have control over the effect of long-term dependencies and the size of the lag, denoted as p_l, while avoiding/controlling noisy measurements and outliers that are typically present in real-world datasets.

Two different pattern recognition aspects are tested through the synthetic datasets: the accuracy of the models on a time series that follows a nonlinear recurrent relationship and the detection capabilities of fixed length sequences that repeatedly (stochastically) appear.

For the first recognition aspect, a dataset is created with a nonlinear dependence located at a fixed lagged value p_l, where the nonlinearity is due to changing sign. This dataset, referred to as the Switching time series, is generated using two i.i.d. random sequences, $z(k) \in \mathbb{R}$ and $z_{sign}(k) \in \{1, 0\}$ following the distributions $N(0, 1)$ and $B(1, \rho_{swt})$; respectively, and two lags of $z(k)$ as shown in (4.2.1).

$$y(k) = a_1 z^2(k) + a_2 z(k-1) + a_{p_l} 2\big(z_{sign}(k-p_l) - 0.5\big) z^2(k-p_l) \qquad (4.2.1)$$

where a_1, a_2 and a_{p_l} are constant coefficients with values 0.25, 0.35 and 0.35, respectively; and ρ_{swt} is the switching probability. In (4.2.1) the term $2\big(z_{sign}(k-p_l) - 0.5\big) \in \{1, -1\}$ creates the switching-sign effect for the lagged variable $z^2(k-p_l)$.

The second dataset, referred to as Binary sequence, is motivated by the potential limitation expressed in Section 3.1 about (4.1.1), the forgetting effect of relevant information in a standard LSTM. The latter is explored by embedding replicas of two different binary patterns of length 28, b_1 and b_2, in a long sequence of bits, separated from each other by a constant length. The long

sequence is composed of 28-bit and 112-bit sequences whose individual values are obtained from the distribution $B(1, 0.5)$. The embedding of the patterns is described in Algorithm 4.2, where n_{points} is assumed to be a large multiple of the length of b_1 and b_2, $rand$ is a function generating samples from the uniform distribution on $[0, 1]$ and $generate(n_r)$ creates random binary sequences of length n_r.

Algorithm 4.2: Binary sequence construction

Input: b_1, b_2, n_{points} //Patterns and number of points//
Set: $n_s \leftarrow n_{points}/2length(b_1 + b_2)$ //number of 112-length subsequences//
$B_{seq} \leftarrow empty$ //the desired Binary sequence dataset//
for i = 1 to n_s
if: $rand$>0.5
$v_{112} \leftarrow generate(112),\ B_{seq} \leftarrow append(B_{seq}, v_{112})$
Else:
$v_{28}^1 \leftarrow generate(28), v_{28}^2 \leftarrow generate(28)$
$B_{seq} \leftarrow append(S, v_{28}^1, b_1, v_{28}^2, b_2)$
Output: B_{seq}

Examining Algorithm 4.2, with probability 0.5 a 112-bit sequence is created, containing patterns $b_1$and b_2 as well as short random sequences $v_{28}^{(1)}$ and $v_{28}^{(2)}$; placing $v_{28}^{(2)}$ between the patterns. The Binary sequence dataset represents one of many possible instances in which repetitive patterns are embedded among non-relevant data, which can be particularly challenging for linear models even when the pattern length is known.

4.2.4 Real-world datasets

In order to identify the performance of the proposed E-LSTM on real-world problems, four univariate time-series datasets were selected; categorized as small, medium and large sizes, with the last category containing two datasets.

The small-size dataset is the well-known Chicken Pox [111] time series, which represents the monthly occurrences of chicken pox in New York City, between the years 1931 and 1972. It is composed of 498 samples with an apparent yearly seasonality.

For the medium-size category, a popular dataset containing the monthly mean number of detected sunspots [112], from Jan 1749 to Dec 2019, is used. This dataset is characterized by showing a degree of seasonality occurring every 10 to 11 years and is composed of 3252 samples.

The large-size category is composed of temperature data for the city of Toronto, Canada [113], and the Power Consumption in the US Eastern Interconnection grid, reported by FirstEnergy Corp [114]. In both datasets hourly information is provided, spanning from Jan 2011 to Dec 2020 for the first dataset and from Dec 31st 2011 to Jan 1st 2018 for the second dataset. As expected, both datasets show a seasonality of 24 hours.

4.2.5 Implementation details

For the experiments performed over the datasets introduced in the previous section and using the RNN and SARIMA models, original subroutines were created using the MATLAB 2018a environment. Simulations using MATLAB-library-built LSTMs were performed for verification purposes, resulting in no significant performance difference with respect to the original subroutines. The Adam optimization algorithm [43], [115]-[116], a variation of the GD algorithm, was used for the training process in all experiments. Adam's hyperparameters were set to standard values, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. A small value for the learning rate, $\alpha = 0.001$, was chosen to mitigate potential EG issues during the training process, using up to 10000 epochs. Following the same training setting described in Algorithm 4.1, the validation-set loss function

value, $\mathcal{L}^{(val)}$, was used as one of the stopping criteria for the training process, specifically using a threshold $m_{fails} = 4000$ for the maximum number of consecutive fails on decreasing the error.

A similar training implementation was used for the CW-RNN, using the Adam algorithm with standard values, but using a smaller learning rate, $\alpha = 0.0003$, in order to avoid the EG problem. The number of modules, n_{mod}, a hyperparameter needed for the CW-RNN, was set to 7 for the Chicken Pox dataset due to its small size, and to 9 for the remaining datasets. Also, the frequency of activation of these modules was set as in [98], from 1 to $2^{n_{mod}}$, and using the same number of hidden units per module.

For selecting the number of hidden units in each of the four LSTM variants (including the E-LSTM) and the CW-RNNs an iterative process using the values from the set $\{2^0, 2^1, \ldots, 2^9\}$ was performed over the validation set, with 40 repetitions for each value; the value producing the best average validation-set performance was selected. Simulations were performed in the Beluga server cluster, operated by the Digital Research Alliance of Canada (formerly Compute Canada), using 2.4GHz CPUs.

4.3 Experimental Results and Analysis

The simulation results for each of the LSTM variants, CW-RNN and SARIMA models are presented. In all tables in this section performance indicators are provided such as average (μ) and standard deviation (σ) of the RMSE. The size of each model is selected based on the validation-set performance (minimum $\mu_{val} + \sigma_{val}$). Additionally, the following metrics are provided: number of hidden units and number of parameters (n_h, $n_{\boldsymbol{\theta}}$); average training time per iteration ($\bar{t}_{iter}$); average time and number of iterations to achieve the optimal loss function value ($\bar{t}^{(opt)}$, $\bar{I}_{iter}^{(opt)}$).

Also, the hyperparameter p for the E-LSTM is explicitly indicated next to the name of the network in the results and bold text is used to represent the network with best testing performance (minimum $\mu_{test} + \sigma_{test}$).

4.3.1 Synthetic datasets

For the synthetically created datasets, the lag associated with $\boldsymbol{x}_{aug}(k)$ in the augmented-input models was set equal to the lag dependence, i.e., $k_i = p_l$. The hyperparameter p in the E-LSTM and E-LSTM-A models was set equal to $p_l - 1$, due to the input to the network being the lagged desired output (lag 1). Also, for each of the two datasets 10 different sequences are used.

In the case of the Switching dataset, experiments varying the value for the switching-sign probability ρ_{swt} of $z_{sign}(k)$ were carried out, creating different instances. Results can be found in Table 4.3.1-Table 4.3.2 which correspond to ρ_{swt} taking the values 1 and 0.01, representing high-frequency and low-frequency switching behavior, respectively. Additionally, the lag-dependence p_l was set to 22 and 50 for the high-frequency and low-frequency switching datasets, respectively. A larger lag p_l was selected for the latter dataset instance to account for the higher linearity associated with a lower switching frequency. A time window of 100 was used for the backward and forward passes during training. No significant changes were found, in terms of performance among variants, for other values of ρ_{swt} between 0.01 and 1.

By assuming the only unpredictable value in (4.2.1) is the term $a_1 z^2(k)$ it is possible to establish a (not necessarily tight) lower bound on the minimum possible RMSE. This lower bound is also presented in Table 4.3.1-Table 4.3.2 and is used as a reference for relative comparison.

For the high-frequency Switching dataset the E-LSTM outperforms all other models (Table 4.3.1), outperforming the LSTM and CW-RNN by a significant margin, and it shows better parameter efficiency, as observed in Fig. 4.3.1. Also, the E-LSTM outperforms the LSTM-A despite the latter model using the lagged input directly where the nonlinear dependence occurs. This can be seen as the E-LSTM being able to model the nonlinear behavior of this time series better than the networks using input augmentation.

For the low-frequency Switching dataset, the augmented E-LSTM-A and LSTM-A showed similar performance (Table 4.3.1), with noticeably better performance than their standard versions and the remaining models. These results can be partly attributed to a lower nonlinear effect caused by the low-frequency switching (with switching probability of 0.01), in contrast with the results observed for the high-frequency Switching dataset. Also, the standard E-LSTM network still shows better parameter efficiency across different sizes, with respect to the standard LSTM and CW-RNN networks.

Table 4.3.1. Results for the Switching dataset ($k_i = 22$). SARIMA hyperparameters were set as: AR = MA=1 and SAR = SMA = 22, resulting in RMSE = 0.6811. Lower bound RMSE = 0.4227. Forward-backward pass length of 100.

Switching series 100% probability switching ($p_l = 22$)											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	0.6232	0.0980	0.6810	0.0085	0.6771	0.0182	128	66689	1.51	381.37	575.87
E-LSTM$_{22}$	**0.5894**	**0.0080**	**0.6198**	**0.0047**	**0.6060**	**0.0044**	**128**	**83329**	**1.97**	**69.82**	**137.55**
CWRNN	0.6718	0.0109	0.7025	0.0020	0.6938	0.0353	72	2412	0.32	4865.43	1556.94
LSTM-A	0.6014	0.0052	0.6204	0.0029	0.6156	0.0040	128	67201	1.51	69.32	104.67
E-LSTM$_{22}$-A	0.5855	0.0123	0.6167	0.0039	0.6071	0.0069	128	83969	1.98	54.82	108.54
CWRNN-A	0.5934	0.0155	0.6286	0.0028	0.6115	0.0077	288	40320	1.70	426.78	725.53

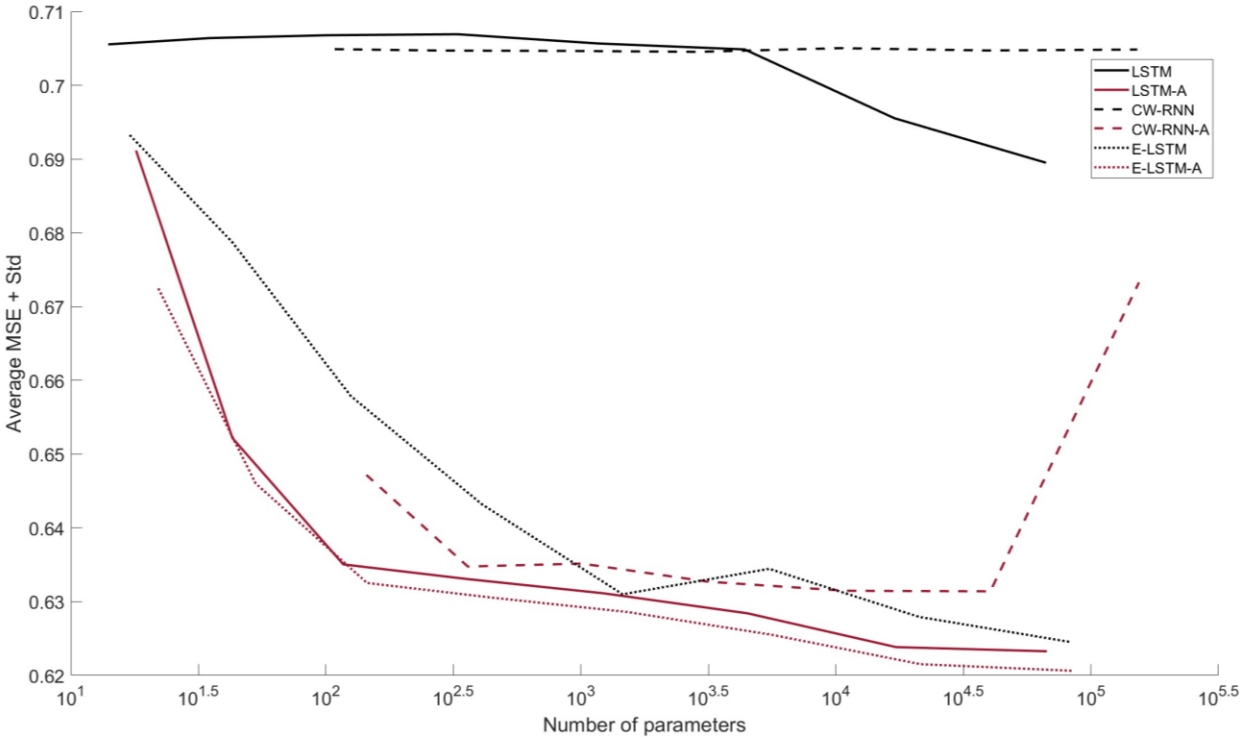


Fig. 4.3.1. Validation set performance across different sizes for the Switching-100 dataset.

Table 4.3.2. Results for the Switching dataset ($k_i = 50$). SARIMA hyperparameters were set as: AR = MA = 1 and SAR = SMA = 50, resulting in RMSE = 0.732. Lower bound RMSE = 0.4095. Forward-backward pass length of 100.

Switching series 1% probability switching ($p_l = 50$)											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	0.6988	0.0041	0.7675	0.0012	0.7258	0.0116	32	4385	0.53	510.15	270.38
E-LSTM$_{50}$	0.6484	0.0110	0.7252	0.0082	0.7207	0.0204	64	21185	1.19	459.35	546.63
CWRNN	0.6866	0.0090	0.7678	0.0029	0.7489	0.0141	144	10368	0.57	1791.42	1021.11
LSTM-A	0.6396	0.0111	0.6858	0.0041	0.6777	0.0181	8	361	0.41	413.72	169.63
E-LSTM$_{50}$-A	**0.6490**	**0.0064**	**0.6858**	**0.0052**	**0.6784**	**0.0096**	**4**	**145**	**0.67**	**2666.25**	**1786.39**
CWRNN-A	0.6360	0.0153	0.6984	0.0094	0.7346	0.0489	72	3168	0.32	2671.43	854.86

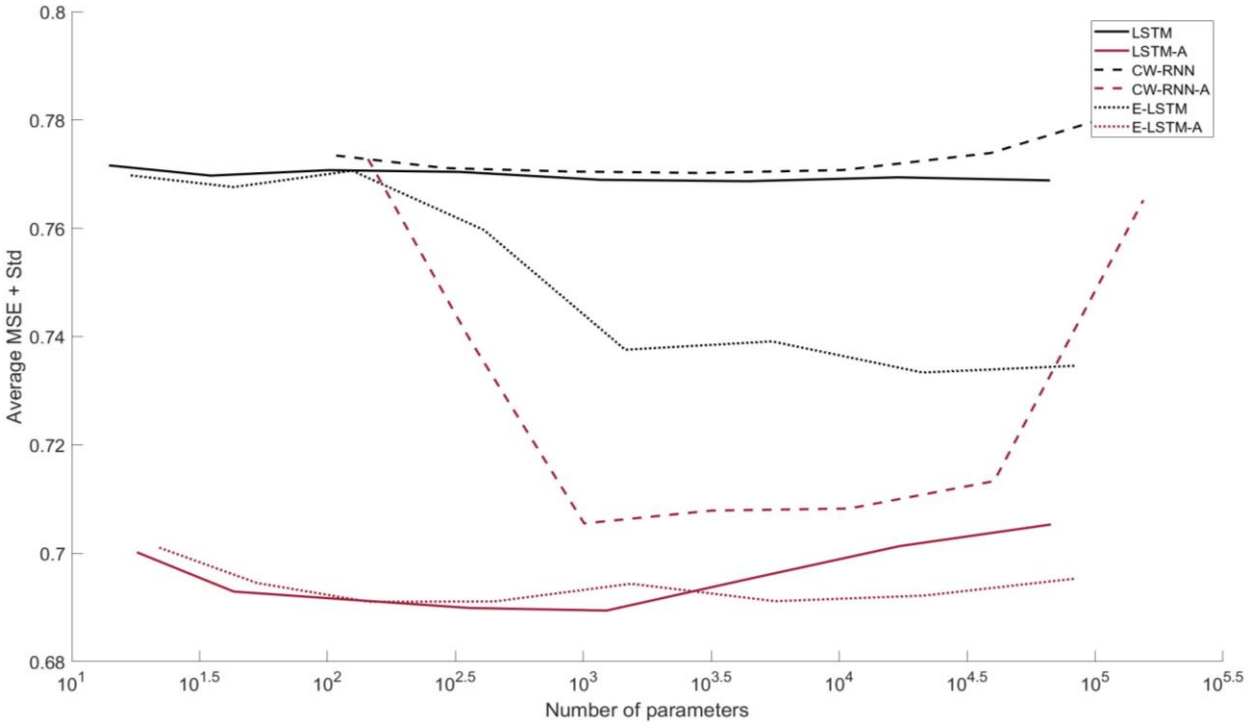


Fig. 4.3.2. Validation set performance across different sizes for the Switching-01 dataset. Forward-backward pass length of 100.

In the case of the Binary sequence (Table 4.3.3) the E-LSTM hyperparameter p was set to 29 to create a direct connection between elements in both 28-bit sequences, $v_{28}^{(1)}$ and $v_{28}^{(2)}$. Additionally, a time window of 1120 was used for the backward and forward passes during training. This window size was selected so that the resulting subsequences contain five instances, on average, of the 112-bit sequence (in which $v_{28}^{(1)}$ and $v_{28}^{(2)}$ are embedded), while keeping the subsequence length not too large. Similar to the Switching datasets, a (not necessarily tight) lower bound for the minimum RMSE was derived for the Binary Sequence dataset, shown in Table 4.3.3.

The results in Table 4.3.3 show again the E-LSTM as the network with the best testing performance. Even though it uses a larger number of parameters for this case, the alternative networks' performances stagnate more when the sizes are increased (Fig. 4.3.3).

Table 4.3.3. Results for the Binary Sequence dataset ($k_i = 29$). ARMA used lags from 1 to 112 for AR and MA components, resulting in RMSE = 0.6811. Lower bound RMSE = 0.4357. Forward-backward pass length of 1120.

Binary Sequence											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	0.4586	0.0037	0.4624	0.0039	0.4645	0.0041	8	329	1.488	1479.67	0.6116
E-LSTM29	**0.4314**	**0.0051**	**0.4520**	**0.0052**	**0.4582**	**0.0071**	**128**	**83329**	**7.242**	**39.02**	**0.0785**
CWRNN	0.4490	0.0086	0.4653	0.0070	0.4706	0.0078	144	10368	2.07	1774.885	1.0206
LSTM-A	0.4601	0.0073	0.4650	0.0070	0.4692	0.0074	8	361	1.503	1375.935	0.5745
E-LSTM29-A	0.4292	0.0078	0.4552	0.0118	0.4618	0.0131	128	83969	7.133	45.815	0.0908
CWRNN-A	0.4498	0.0115	0.4805	0.0051	0.4942	0.0093	288	40320	5.992	1027.735	1.7106

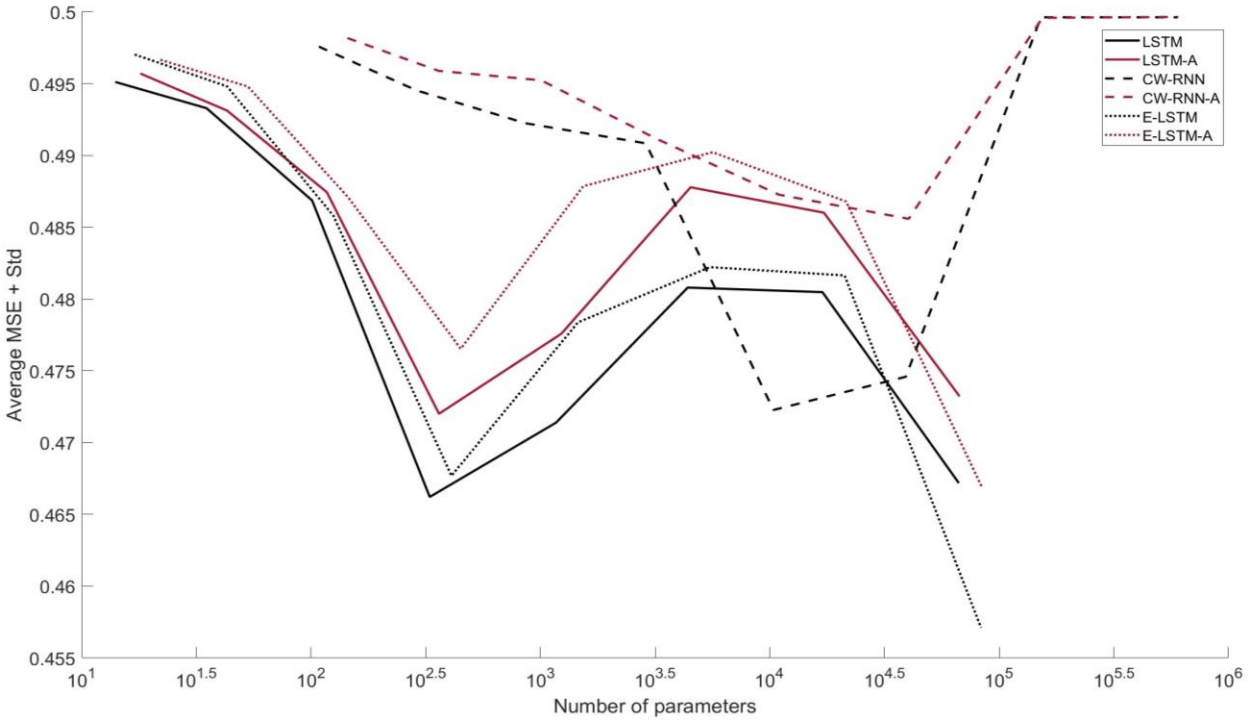


Fig. 4.3.3 Validation set performance across different sizes for the Binary sequence dataset.

Regarding the LSTM variants' performance on the synthetic datasets the E-LSTM and its augmented version required a similar or lower number of parameters to achieve a similar or better performance than the alternative networks, in some cases with the number of parameters being an order of magnitude smaller. Also, it should be highlighted that such performance is achieved without drastically increasing the CPU time needed for the training process.

4.3.2 Real-world datasets

For the experiments involving real-world datasets, the distance-correlation-based process described in Section 4.1.5 was used for the selection of the hyperparameter p of the proposed E-LSTM and E-LSTM-A models. Also, for the augmented variants, LSTM-A and E-LSTM-A, a correlation analysis on the training sets was performed to select the lag value k_i for the input augmentation. Differentiated datasets were used in all LSTM variants for the cases in which the integrative component of SARIMA was selected, since the differentiation due to this component is not directly performed in the LSTM variants; whenever this was the case, the correlation analysis was performed over the differentiated dataset.

Regarding the Chickenpox dataset (small size), the E-LSTM and its augmented variation showed a significant improvement over the other networks (Table 4.3.4). They reduced the number of parameters needed overall by orders of magnitude, while achieving better performance. Also, the E-LSTM-A was the top-performing network, using the least number of parameters and with consistent performance across different sizes (Fig. 4.3.4).

Table 4.3.4. Results for the Chickenpox dataset ($k_i = 24$). SARMA components used lags 1 to 4 for the AR and MA and a value of 12 for SAR and SMA, resulting in RMSE = 113.423. Forward-backward pass length of 200.

Chickenpox											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	n_{θ}	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	146.46	41.48	111.05	4.93	143.31	32.44	128	66689	0.168	8874.43	0.4141
E-LSTM24	216.48	9.76	114.20	4.70	121.36	7.33	2	43	0.0601	4701.95	0.0785
CWRNN	145.22	15.14	116.40	7.12	134.45	17.31	192	16320	0.09588	7494.76	0.1996
LSTM-A	127.88	48.71	104.69	8.04	144.93	40.59	128	67201	1.503	2122.30	0.8861
E-LSTM24-A	**201.42**	**0.84**	**100.72**	**1.75**	**108.99**	**1.53**	**1**	**22**	**0.17**	**8819.90**	**0.4165**
CWRNN-A	142.22	20.77	102.80	5.13	147.34	30.09	192	16800	0.0839	6845.23	0.1595

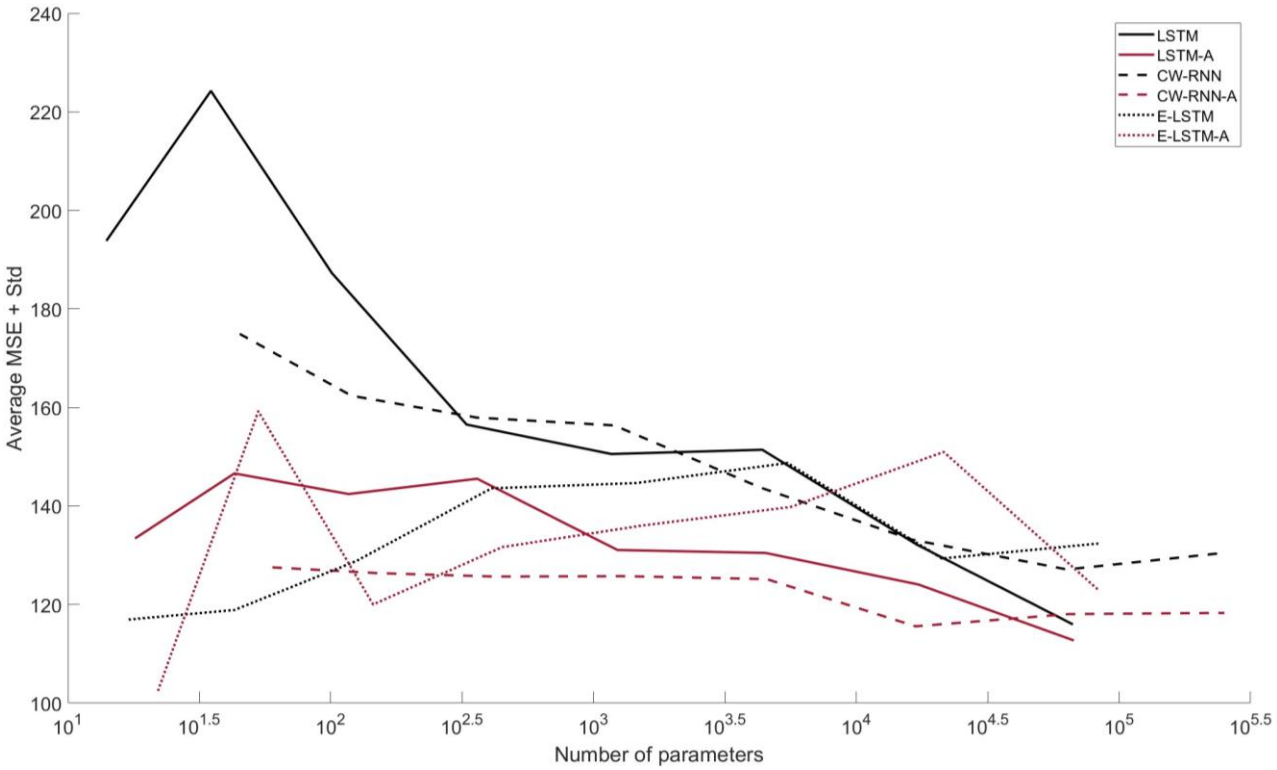


Fig. 4.3.4. Validation set performance across different sizes for the Chickenpox dataset.

For the experiments with the Sunspots dataset (medium size) the proposed E-LSTM showed similar performance to the LSTM (Table 4.3.5), and slightly outperformed the alternative networks; however, the SARIMA algorithm had very similar performance as the E-LSTM. The fact that a linear model, using less than 100 parameters, can achieve the best performance among the models might partly indicate why little improvement is generated by the E-LSTM architecture, as there might be little nonlinear time dependence in the data. Furthermore, when analyzing the performance curves of the networks in (Fig. 4.3.5) it can be noticed that those for the E-LSTM and LSTM networks have similar shape and values which might corroborate the marginal contribution of adding a mechanism to capture nonlinear long-term dependencies for this dataset.

Table 4.3.5. Results for the Sunspots dataset ($k_i = 12$). SARIMA hyperparameters used lags 1 to 34 for the AR and MA and a value of 127 for SAR and SMA, resulting in RMSE = 22.80. Forward-backward pass length of 288.

Sunspots											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	24.22	0.19	24.89	0.21	22.71	0.27	4	101	0.2079	3386.75	704.11
E-LSTM15	**24.22**	**0.21**	**24.93**	**0.18**	**22.67**	**0.18**	**4**	**125**	**0.2910**	**3179.8**	**925.32**
CWRNN	25.16	0.11	26.32	0.24	23.51	0.30	9	108	0.1022	4826.75	493.29
LSTM-A	24.15	0.20	25.00	0.20	22.83	0.24	4	117	0.2091	2616	547.01
E-LSTM15-A	24.09	0.23	25.15	0.27	23.00	0.32	4	145	0.2946	2184.2	643.47
CWRNN-A	25.00	0.19	26.10	0.42	23.47	0.30	9	144	0.1030	4527.98	466.38

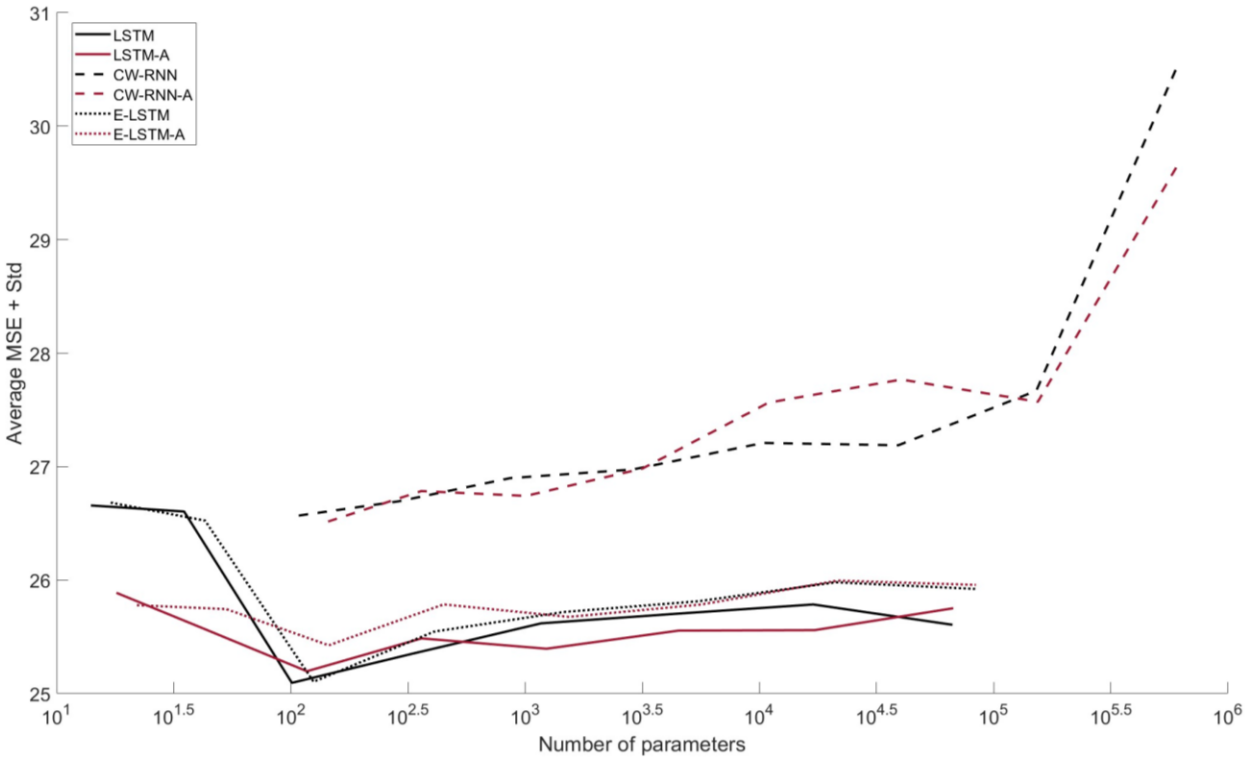


Fig. 4.3.5. Validation set performance across different sizes for the Sunspots dataset.

The results for the Power consumption dataset (Table 4.3.6) further supports the E-LSTM's capability to break potential walls the LSTM network might face, in terms of capturing long-term dependencies. This can be concluded by directly comparing the augmented and non-augmented versions side by side in Table 4.3.6 and analyzing how quickly the loss function value decreases with respect to the number of parameters for small network sizes in Fig. 4.3.6, where the performance curves of the E-LSTM and its augmented version are almost identical.

In the Toronto temperature dataset, the E-LSTM is the best-performing model, generating a small improvement over the LSTM and using less than half of the number of parameters in comparison (Table 4.3.7). Additionally, in terms of parameter efficiency, the performance behavior of the E-LSTM network seems to be more consistent across different network sizes,

providing more evidence that the LSTM needs a larger number of units to create variety in the behavior of the forget gates.

Table 4.3.6. Results for the Power Consumption dataset ($k_i = 24$) SARIMA hyperparameters used lags 1 to 14 for the AR and MA and a value of 24 for SAR and SMA, resulting in RMSE = 85.1136. Forward-backward pass length of 672.

Power consumption											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	65.00	1.00	63.75	0.58	65.58	1.40	32	4385	6.18	1452.18	2.493
E-LSTM24	65.16	1.54	63.08	0.53	63.71	0.63	128	83329	22.15	213.53	1.314
CWRNN	83.08	4.75	82.41	2.94	82.76	2.89	288	39168	18.87	1401.7	7.347
LSTM-A	65.35	0.92	63.61	0.42	64.74	0.57	32	4513	6.25	619.13	1.075
E-LSTM24-A	**65.08**	**1.20**	**62.98**	**0.52**	**63.47**	**0.63**	**64**	**21505**	**12.50**	**221.45**	**0.769**
CWRNN-A	76.64	2.23	73.09	0.67	73.58	0.95	288	40320	19.44	900.03	4.860

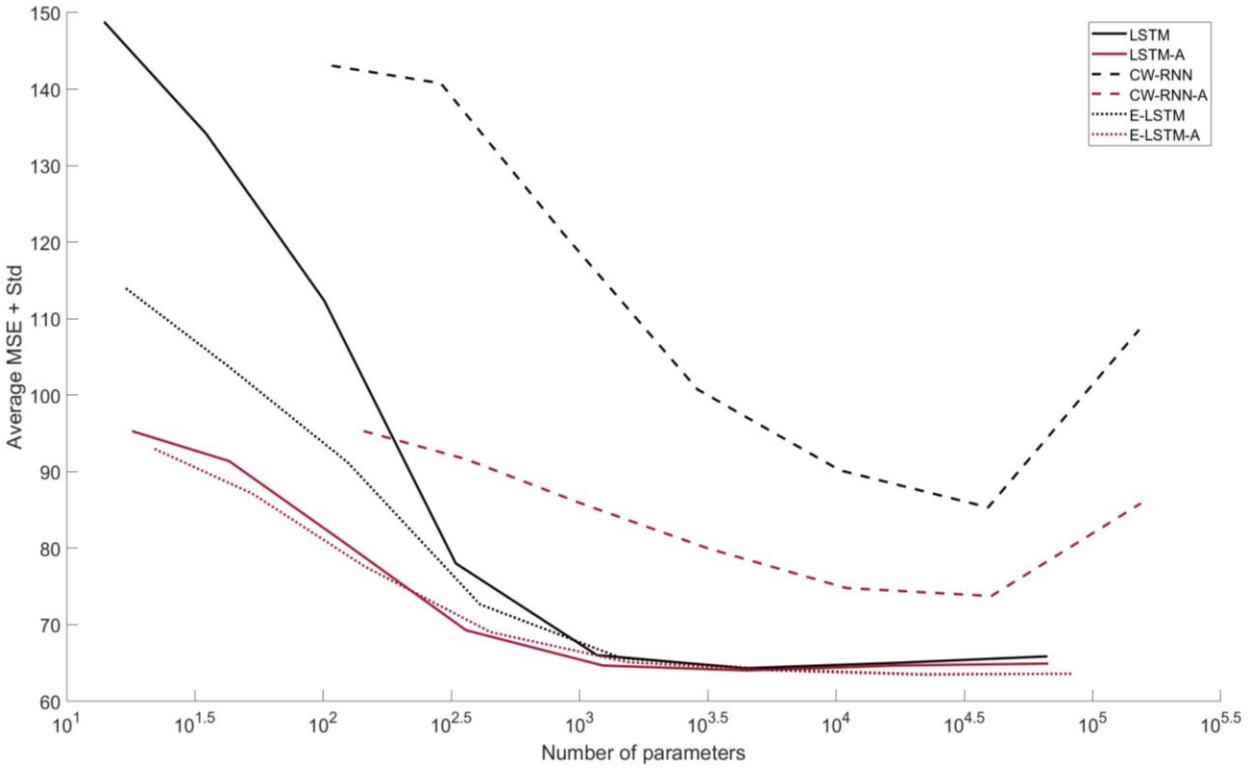


Fig. 4.3.6. Validation set performance across different sizes for the Power consumption dataset.

Table 4.3.7. Results for the Toronto temperature dataset ($k_i = 24$). SARMA components used lags 1 to 17 for the AR and MA and a value of 23 for SAR and SMA, resulting in RMSE = 0.6955. Forward-backward pass length of 672.

Toronto temperature											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	0.6751	0.0029	0.6618	0.0011	0.6684	0.0015	16	1169	7.217	692.68	1.3886
E-LSTM24	**0.6775**	**0.0023**	**0.6607**	**0.0020**	**0.6676**	**0.0014**	**8**	**409**	**9.961**	**1051.135**	**2.9084**
CWRNN	0.7443	0.0195	0.7178	0.0132	0.7343	0.017	144	10368	8.8	590.205	1.4427
LSTM-A	0.6722	0.0021	0.6631	0.0012	0.6715	0.0013	16	1233	7.323	262.41	0.5338
E-LSTM24-A	0.6776	0.0032	0.6631	0.0019	0.6702	0.0019	8	449	9.997	362.44	1.0065
CWRNN-A	0.7448	0.0020	0.7156	0.0026	0.7319	0.0024	18	360	3.108	377.53	0.3259

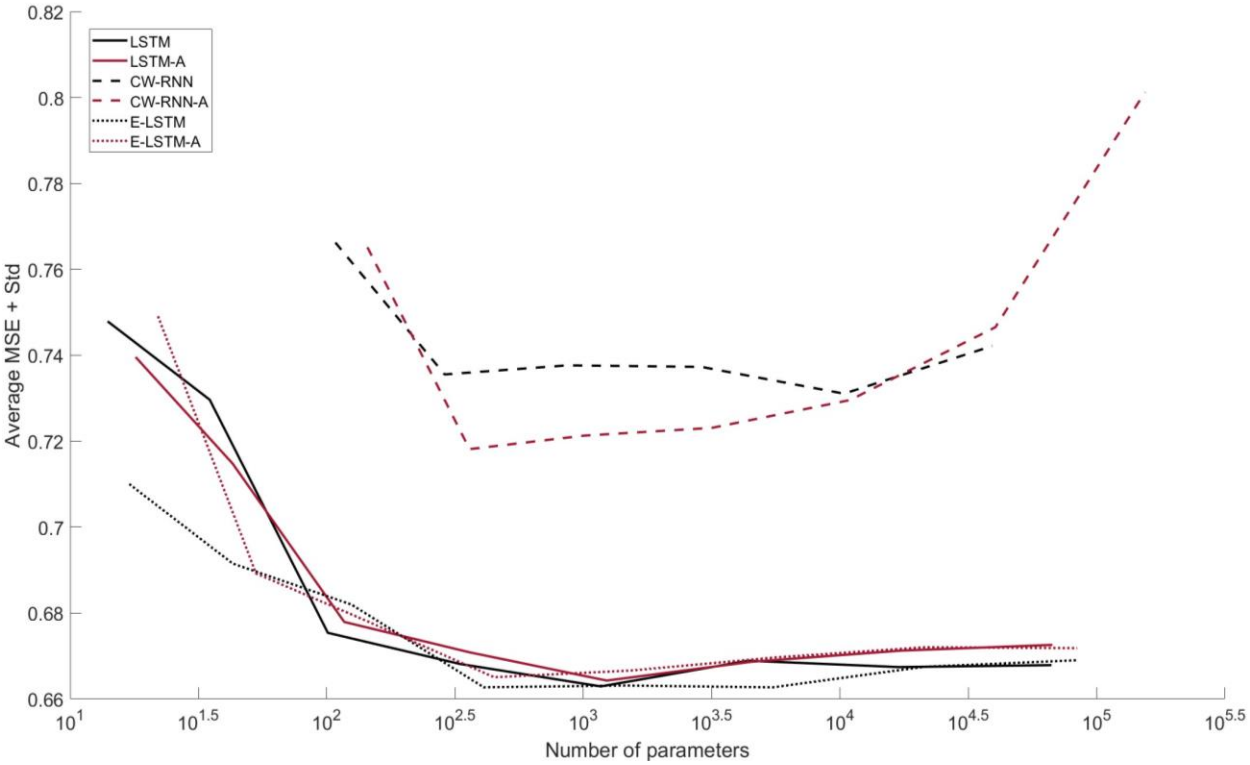


Fig. 4.3.7. Validation set performance across different sizes for the Toronto temperature dataset.

It is important to highlight that for the case of the large-size datasets (on the order of 10^5 datapoints) the E-LSTM and its augmented version were the best-performing models, when compared to the LSTM and the CW-RNN, while using a similar or lower number of parameters for the selected network sizes and showing a sharper increase in the performance curves for small network sizes.

From the previous real-world results, a general pattern that can be observed is that the proposed E-LSTM architecture appears to be a better option when compared to the LSTM and CW-RNN models, in terms of the number of parameters required to achieve similar performance, creating a significant reduction in the size of the model in some cases (by an order of magnitude). Furthermore, the increase in training time with the proposed variants remained reasonable across these experiments.

Chapter 5

GI-LSTM: Generalized and Interpretable LSTM

In this chapter a new mechanism that generalizes the E-LSTM connectivity approach is proposed, named Generalized Interpretable LSTM (GI-LSTM). This mechanism further increases the explicit recursive connectivity among cell states to directly compensate for exponentially weakening connectivity (EWC) across time, when needed. The GI-LSTM aims to extract long term dependencies more efficiently even when their precise location is unknown. In addition, due to the specific method used to create this connectivity, the GI-LSTM is embedded with an easy-to-use interpretability component (defined as being able to provide explanations in understandable terms to a human) that indicates the statistical relevance it gives to previous cell states.

5.1. GI-LSTM Architecture

5.1.1 Motivation for a generalization

As stated in Chapter 4, relevant information for long-term dependencies could be left uncaptured in the standard LSTM due to the lack of variety in the decay rate among forget gates, an issue partly addressed by the E-LSTM approach of increasing the explicit connectivity between cell states. Taking a step forward in this approach the connectivity among cell states with respect

to their previous values is increased, eliminating the need for specifying a previous location in time (in contrast to the E-LSTM) and replacing it by a flexible user-defined interval of previous values, which can be based on loose estimations of: maximum lag dependency, apparent seasonalities (as seen in human-driven phenomena) and other guessed temporal information. Also, by creating this higher connectivity the need for the DC measure is removed, resulting in reduced preprocessing time. This is particularly useful since the computation of the DC can be time consuming when trying to identify very long-term dependencies, even in univariate time series. Furthermore, the proposed GI-LSTM architecture enables a semi-local interpretation, as defined in [39], specifying which parts of the time series the network gives relevance to, within the user-defined time intervals.

5.1.2 Forward equations and conceptualization

To keep the efficiency in the number of weights achieved by the E-LSTM while extending the reach into past values, the new increased connectivity is performed by introducing dynamic 'memory groups', $\boldsymbol{m}_s(k) \in \mathbb{R}^{n_h}$, designed to create a balance between the explicit temporal connectivity and the number of parameters.

In more detail, the first memory group, $\boldsymbol{m}_1(k)$, contains explicit information of contiguous lagged cell states, starting at 1; the second memory group, $\boldsymbol{m}_2(k)$, contains information of non-contiguous lagged values of $\boldsymbol{m}_1(k)$, a structure followed by higher-order memory groups, $\boldsymbol{m}_s(k)$, up to a maximum number ς. Similar to the E-LSTM, a weighted forget gate, $\hat{\boldsymbol{f}}_s(k)$, is associated to each memory group to allow for dynamism during the information processing and to promote stability during the training phase.

$$\boldsymbol{N}_1(k) = [\boldsymbol{c}(k-1), \boldsymbol{c}(k-2), \dots, \boldsymbol{c}(k-q_1)] \tag{5.1.1}$$

$$\boldsymbol{m}_1(k) = \left(\boldsymbol{N}_1(k) \circ \boldsymbol{W}_{\boldsymbol{m}_1}\right)\mathbf{1}_{q_1 \times 1} \tag{5.1.2}$$

$$\boldsymbol{N}_s(k) = [\boldsymbol{m}_{s-1}(k-q_{s-1} \dots q_1), \boldsymbol{m}_{s-1}(k-2q_{s-1} \dots q_1), \dots, \boldsymbol{m}_{s-1}(k-q_s q_{s-1} \dots q_1)], s \geq 2 \tag{5.1.3}$$

$$\boldsymbol{m}_s(k) = \left(\boldsymbol{N}_s(k) \circ \boldsymbol{W}_{m_s}\right)\mathbf{1}_{q_s \times 1}, s \geq 2 \tag{5.1.4}$$

$$\| \left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{row_i} \|_1 = 1 \tag{5.1.5}$$

$$\boldsymbol{c}(k) = \boldsymbol{a}(k) \circ \boldsymbol{i}(k) + \hat{\boldsymbol{f}}_1(k) \circ \boldsymbol{m}_1(k) + \hat{\boldsymbol{f}}_2(k) \circ \boldsymbol{m}_2(k) + \cdots + \hat{\boldsymbol{f}}_\varsigma(k) \circ \boldsymbol{m}_\varsigma(k) \tag{5.1.6}$$

$$\hat{\boldsymbol{f}}_s(k) = \boldsymbol{f}_s(k) \circ \boldsymbol{w}_{\boldsymbol{f}_s}(k) \tag{5.1.7}$$

$$\boldsymbol{w}_{\boldsymbol{f}_s}(k) = \frac{\boldsymbol{f}_s(k)}{\sum_{d=1}^{\varsigma} \boldsymbol{f}_d(k)} \tag{5.1.8}$$

where $\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{row_i}$ is the ith row of the matrix $\boldsymbol{W}_{\boldsymbol{m}_s} \in \mathbb{R}^{n_h \times q_s}$; $\|\cdot\|_1$ is the 1-norm; $\mathbf{1}_{q_s \times 1}$ is a vector of ones of dimension $q_s \times 1$. The logic behind the choice of the 1-norm in the constraint for the group weights $\boldsymbol{W}_{\boldsymbol{m}_s}$, instead of the 2-norm, will be explained in detail in the next section. However, the following is worth noting: there is no constraint on the sign of the weights and the 1-norm constraint is embedded in the network through a nonunique parametrization with respect to learnable parameters $\boldsymbol{\Theta}_s$, as shown in (5.1.9).

$$\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{i,j} = \frac{[\boldsymbol{\Theta}_s]_{i,j}}{\sum_{row_i} |\boldsymbol{\Theta}_s|} \tag{5.1.9}$$

where $\sum_{row_i} |\boldsymbol{\Theta}_s| = \sum_{j=1}^{q_s} \left|[\boldsymbol{\Theta}_s]_{i,j}\right|$.

In the definition of higher-order memory groups, the design choice of creating lag values that are multiplicative, $q_s q_{s-1} \dots q_1$, instead of additive is intended to avoid information overlapping, as shown in Fig. 5.1.1, that could create potentially unnecessary redundancy.

There are three major advantages in the proposed memory-group approach, when compared to the E-LSTM approach. (i) It theoretically allows for reaching very long-term dependencies due to the multiplicative lags in the higher-order memory groups, i.e., $k - \left(q_1 + q_2 q_1 \dots + q_\varsigma q_{\varsigma-1} \dots q_1\right)$, without the aid of the DC measure. (ii) Higher-order memory groups compress the information in lower-order memory groups, functioning as filters for past information. (iii) The architecture allows for a dynamic balance between short-term, long-term and very long-term dependencies due to the weighted gates $\hat{\boldsymbol{f}}_s(k)$ associated to each memory group. Additionally, and following a similar strategy to the E-LSTM architecture, only one forget gate is added per memory group which, when considering the multiplicative reaching effect, maintains the number of parameters low in practice (we will see this in Section 5.3).

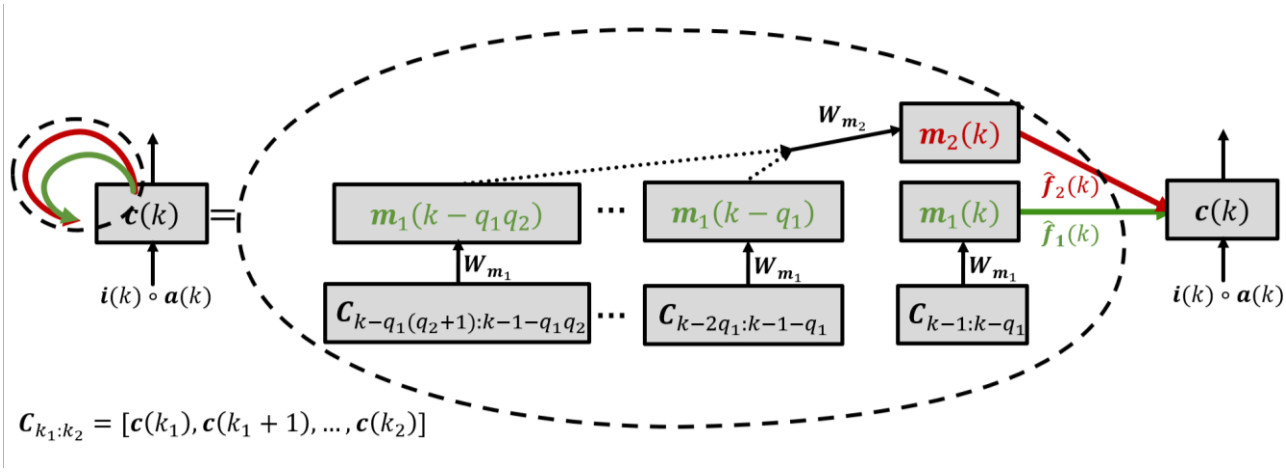


Fig. 5.1.1. Simplified graphical representation of the memory-group mechanism in the GI-LSTM, with $\varsigma = 2$.

5.1.3 Backward equations and analysis

The BP equations are derived following a similar approach to that described in Sections 3.1 and 3.2. Hence, the resulting backward equations linked to the GI-LSTM are as follows:

$$\boldsymbol{\delta}_{\boldsymbol{m}_\varsigma}(k) = \hat{\boldsymbol{f}}_\varsigma(k) \circ \boldsymbol{\delta}_{\boldsymbol{c}}(k) \tag{5.1.10}$$

$$\boldsymbol{\delta}_{\boldsymbol{m}_s}(k) = \hat{\boldsymbol{f}}_s(k) \circ \boldsymbol{\delta}_{\boldsymbol{c}}(k) + \sum_{r=1}^{q_s} \left(\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{col_r} \circ \boldsymbol{\delta}_{\boldsymbol{m}_{s+1}}(k + rq_{s-1} \dots q_1) \right), s < \varsigma \tag{5.1.11}$$

$$\boldsymbol{\delta}_{\boldsymbol{c}}(k) = \boldsymbol{\delta}_{\boldsymbol{h}}(k) \circ \boldsymbol{o}(k) \circ \dot{\sigma}_{th}\big(\boldsymbol{c}(k)\big) + \sum_{r=1}^{q_1} \left(\left[\boldsymbol{W}_{\boldsymbol{m}_1}\right]_{col_r} \circ \boldsymbol{\delta}_{\boldsymbol{m}_1}(k + r) \right) \tag{5.1.12}$$

$$\boldsymbol{\delta}_{\boldsymbol{W}_{m_s}}(k) = diag(\boldsymbol{\delta}_{\boldsymbol{m}_s}(k))\boldsymbol{N}_s(k) \tag{5.1.13}$$

$$D_{[\boldsymbol{\Theta}_s]_{row_i}} \left(\left[\boldsymbol{W}_{m_s}\right]^T_{row_i} \right) = \left(\frac{-[\boldsymbol{W}_{m_s}]^T_{row_i} sign\left([\boldsymbol{\Theta}_s]_{row_i}\right)}{\sum_{row_i} |\boldsymbol{\Theta}_s|} + \frac{\boldsymbol{I}_{q_s x q_s}}{\sum_{row_i} |\boldsymbol{\Theta}_s|} \right) \tag{5.1.14}$$

$$\boldsymbol{\delta}_{[\boldsymbol{\Theta}_s]_{row_i}}(k) = \left(\boldsymbol{\delta}_{[\boldsymbol{W}_{\boldsymbol{m}_s}]_{row_i}}(k) \right) D_{[\boldsymbol{\Theta}_s]_{row_i}} \left(\left[\boldsymbol{W}_{m_s}\right]^T_{row_i} \right) \tag{5.1.15}$$

$$\boldsymbol{\delta}_{\boldsymbol{\Theta}_s}(k) = nf(\boldsymbol{\Theta}_s) \left(-diag\left(\left(\boldsymbol{\delta}_{\boldsymbol{W}_{m_s}}(k) \circ \boldsymbol{W}_{\boldsymbol{m}_s} \right) \boldsymbol{1}_{q_s x 1} \right) sign(\boldsymbol{\Theta}_s) + \boldsymbol{\delta}_{\boldsymbol{W}_{\boldsymbol{m}_s}}(k) \right) \tag{5.1.16}$$

$$\boldsymbol{\delta}_{\boldsymbol{f}_s}(k) = 2\boldsymbol{\delta}_{\boldsymbol{m}_s}(k) \circ \hat{\boldsymbol{f}}_s(k) - \sum_{d=1}^{s_{max}} \boldsymbol{\delta}_{\boldsymbol{m}_d}(k) \circ \hat{\boldsymbol{f}}_d(k)^2 \tag{5.1.17}$$

where ς is the index of the last memory group; $\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{col_i}$ is the ith column of the matrix $\boldsymbol{W}_{\boldsymbol{m}_s}$; $nf(\boldsymbol{\Theta}_s)$ is a normalizing matrix, $diag\left(\left[\sum_{row_1} |\boldsymbol{\Theta}_s|, \dots, \sum_{row_{n_h}} |\boldsymbol{\Theta}_s|\right]\right)^{-1}$.

When analyzing the GI-LSTM backward equations it can be observed that in (5.1.14) the individual elements of the gradient $\boldsymbol{\delta}_{[\boldsymbol{\Theta}_s]_{row_i}}(k)$ do not vanish solely by the fact that the elements might have values close to or equal to 0, specifically due to the non-zero term $\frac{\boldsymbol{I}_{q_s x q_s}}{\sum_{row_i} |\boldsymbol{\Theta}_s|}$ resulting

from the 1-norm, something that would occur if the 2-norm were used instead. Also, it can be noticed that when (5.1.15) is expanded the resulting term $-\boldsymbol{\delta}_{[\boldsymbol{W}_{\boldsymbol{m}_s}]_{row_i}}(k)\left[\boldsymbol{W}_{m_s}\right]_{row_i}^{T} sign\left([\boldsymbol{\Theta}_s]_{row_i}\right)$ is a projection of the gradient $\boldsymbol{\delta}_{[\boldsymbol{W}_{\boldsymbol{m}_s}]_{row_i}}(k)$ over the current values of $\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{row_i}$. This projection, in the context of (5.1.9), can be interpreted as an opposition to the change in the learnable parameters of the memory groups, $[\boldsymbol{\Theta}_s]_{row_i}$. The opposition occurs when the change in the learnable parameters produces similar values for $\left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{row_i}$, due to a near-to-uniform scaling in $[\boldsymbol{\Theta}_s]_{row_i}$. Consequently, in (5.1.15) all elements in $[\boldsymbol{\Theta}_s]_{row_i}$ influence the gradient of each of its individual elements. This influence partly produces the desired effect of the rescaling transformation (5.1.9), which embeds the lack of need for different values of $\boldsymbol{\Theta}_s$ that result in the same memory-group weights $\boldsymbol{W}_{\boldsymbol{m}_s}$.

Upon further inspection of (5.1.14) and (5.1.16) two things can be noticed. First, the magnitude of $\boldsymbol{\delta}_{\boldsymbol{\Theta}_s}(k)$ is dependent on $nf(\boldsymbol{\Theta}_s)$ but not its direction; second, $\boldsymbol{W}_{\boldsymbol{m}_s}$ does not depend on the normalizing factor $nf(\boldsymbol{\Theta}_s)$ as seen in (5.1.9). Hence, the gradient $\boldsymbol{\delta}_{\boldsymbol{\Theta}_s}(k)$ could become unnecessarily affected by $nf(\boldsymbol{\Theta}_s)$ whenever any of its elements becomes significantly smaller than 1, i.e., $\left(\sum_{row_i}|\boldsymbol{\Theta}_s|\right)^{-1} < 1$. Consequently, $nf(\boldsymbol{\Theta}_s)$ is removed from (5.1.16), resulting in (5.1.18), and instead a 1-norm row normalization is performed on $\boldsymbol{\Theta}^{(s)}$ during each batch (mini-batch) iteration in the training process to ensure $\sum_{row_i}|\boldsymbol{\Theta}_s| = 1$, for each row i.

$$\boldsymbol{\delta}_{\boldsymbol{\Theta}_s}(k) = -diag\left(\left(\boldsymbol{\delta}_{\boldsymbol{W}_{\boldsymbol{m}_s}}(k) \circ \boldsymbol{W}_{\boldsymbol{m}_s}\right)\boldsymbol{1}_{q_s x 1}\right) sign(\boldsymbol{\Theta}_s) + \boldsymbol{\delta}_{\boldsymbol{W}_{\boldsymbol{m}_s}}(k) \tag{5.1.18}$$

It is important to notice that despite the removal of $nf(\boldsymbol{\Theta}_s)$ the parametrization of $\boldsymbol{W}_{\boldsymbol{m}_s}$ in terms of $\boldsymbol{\Theta}^{(s)}$ remains useful, due to the previously discussed projection effect in (5.1.15).

5.1.4 GI-LSTM Interpretability

The row-normalized memory-group matrices, $\| \left[\boldsymbol{W}_{\boldsymbol{m}_s}\right]_{row_i} \|_1 = 1$, in the GI-LSTM architecture represent an easy-to-analyze option to partly access what the network has learned, specifically the relevance that each individual unit, $[\boldsymbol{h}(k)]_i$, assigns to its memory groups, $[\boldsymbol{m}_s]_{row_i}$, which implies temporal relevance in the time series. This interpretability can be achieved by observing, through ς different plots, the absolute values of the ith-row memory-group matrices, i.e., $abs\left(\left[\boldsymbol{W}_{\boldsymbol{m}_1}\right]_{row_i}\right), \ldots, abs\left(\left[\boldsymbol{W}_{\boldsymbol{m}_\varsigma}\right]_{row_i}\right)$.

This approach, although offering a substantial degree of interpretability, would not express how the temporal relevance is distributed across the memory groups when more than one is used, since the relevance is dependent on the dynamic behaviour of the normalized forget gates, $\left[\hat{\boldsymbol{f}}_s(k)\right]_{row_i}$. On the other hand, the dynamic behavior of the forget gates increases the difficulty of interpreting the temporal relevance, since it tends to change from one iteration to the next. Therefore, a middle ground between obtaining a more accurate insight into the distributed relevance and handling the dynamism of the forget gates can be achieved by using the time-averaged values of the forget gates (5.1.19), $\overline{\hat{\boldsymbol{f}}}_s$, since they represent the overall effect the forget gates have across the forward pass. The process of incorporating the averaged values into the effect of the memory-group matrices, as described in (5.1.20), results in integrated memory-group matrices, $\boldsymbol{\omega}_s$, through which a more accurate interpretation of temporal relevance can be obtained. Plotting the rows of the integrated memory-group matrices, $[\boldsymbol{\omega}_s]_{row_i}$, produces the desired interpretability for an individual unit i.

$$\overline{\hat{\boldsymbol{f}}}_{s'} = \frac{1}{\mathcal{S}} \sum_{j=1}^{\mathcal{S}} \hat{\boldsymbol{f}}_s(j) \tag{5.1.19}$$

$$\boldsymbol{\omega}_s = [(\overline{\hat{\boldsymbol{f}}}_{s-1}, diag\left(\overline{\hat{\boldsymbol{f}}}_s\right) abs(\boldsymbol{W}_{\boldsymbol{m}_s})] \tag{5.1.20}$$

where as previously mentioned, $\mathcal{S}$ is the sequence length.

By looking at the individual rows of the integrated matrices it is possible to access the temporal instances considered by individual units. Nevertheless, when interpreting one unit at a time, it might not be straightforward to identify what the whole network assigns relevance to; this is due to the units' interdependence in the recurrence equations, affecting each other through the network gates. Therefore, if holistic interpretability is desired the mean value of the integrated row-normalized matrices (5.1.21), can be used instead.

$$\overline{\widehat{\boldsymbol{\omega}}_s} = \frac{1}{n_h}\mathbf{1}_{1\times n_h}\widehat{\boldsymbol{\omega}}_s \tag{5.1.21}$$

where $\widehat{\boldsymbol{\omega}}_s$ is the result of row-normalizing the matrix $\boldsymbol{\omega}_s$, i.e., $[\boldsymbol{\omega}_s]_{i,j}/\sum_j[\boldsymbol{\omega}_s]_{i,j}$.

5.1.5 Overhead analysis and training implementation

Considering the GI-LSTM structure defined by (5.1.1)-(5.1.13) and (5.1.17)-(5.1.18), the number of learnable parameters in a single layer is:

$$n_{\boldsymbol{\theta}}^{(GI-LSTM)} = (4+\varsigma-1)\big(n_h + n_{input} + 1\big)n_h + \big(q_1 + \cdots + q_\varsigma\big)n_h \tag{5.1.22}$$

As observed in the first term of the RHS of (5.1.22), for a fixed number of hidden units the number of parameters increases proportionally with the number of forget gates, which is equal to the number of memory groups being used, s_{max}, while the second term represents the increase due to the memory groups' learnable parameters, $\boldsymbol{\Theta}_s$. For a large number of hidden units if all the lag values are such that $q_s < n_h$, the increase in parameters per hidden unit with respect to an LSTM remains reasonable, while the long-term reach remains large, $q_1 + q_2q_1 + \cdots + q_sq_{s-1}\ldots q_1$.

Furthermore, for small values of n_h the temporal reach can be easily extended while still using fewer parameters.

The previous two relations are part of the core of the GI-LSTM capabilities as they substantially mitigate the need to add more hidden units to oppose the EWC effect and allow for adding units mostly to increase the expressive power, when the nonlinear complexity of the time series requires it. Also, it should be highlighted that when $\varsigma = 1$ and $q_1 = 1$ the number of parameters in (5.1.22) is equal to the standard LSTM, since such a configuration would result in $\boldsymbol{\Theta}_1 = \mathbf{1}_{n_h \times 1}$ and therefore there would be no need to store such parameters.

In more practical terms and as seen in (5.1.1)-(5.1.4), due to a memory group's need for accessing lagged values of the lower-order memory group, the number of transitory internal states in the GI-LSTM for the forward pass (5.1.23), $n_{\overrightarrow{\Phi}}^{(GI-LSTM)}$, is larger than in the E-LSTM and LSTM architectures. This number is achieved at the expense of frequently shifting values in auxiliary matrices, $\boldsymbol{A}_s \in \mathbb{R}^{n_h \times q_s \dots q_1}$, containing the internal states as expressed in (5.1.23).

$$n_{\overrightarrow{\Phi}}^{(GI-LSTM)} = \left(q_1 + q_2 q_1 \dots + q_\varsigma q_{\varsigma-1} \dots q_1\right) n_h \tag{5.1.23}$$

In the case of memory not being the main constraint, a reduction in computation can be achieved by using larger matrices, $\boldsymbol{A}_s \in \mathbb{R}^{n_h \times \mathcal{S}}$, storing the values across the forward pass and producing an internal overhead in the number of variables as seen in (5.1.24).

$$n_{\overrightarrow{\Phi}}^{(GI-LSTM)} = \mathcal{S}\varsigma n_h \tag{5.1.24}$$

For the backward pass of the training process the overhead analysis is similar to the forward pass, an overhead caused specifically by (5.1.11)-(5.1.13). Hence, when the analysis is carried out,

the resulting memory usage due to dynamic internal states, $n_{\overleftarrow{\Phi}}^{(GI-LSTM)}$, is as described in (5.1.25) for the memory prioritization approach.

$$n_{\overleftarrow{\Phi}}^{(GI-LSTM)} = \left(q_1 + q_2 q_1 \dots + q_\varsigma q_{\varsigma-1} \dots q_1\right) n_h \tag{5.1.25}$$

For the computation prioritization approach, the resulting number of dynamic internal states is given by (5.1.26).

$$n_{\overleftarrow{\Phi}}^{(GI-LSTM)} = \mathcal{S}\varsigma n_h \tag{5.1.26}$$

It is important to clarify that the multiplicative temporal reach, $q_\varsigma q_{\varsigma-1} \dots q_1$, in (5.1.25) is upper bounded by the sequence length, $\mathcal{S}$, itself, due to the standard training process in the RNN architecture. Also sequence lengths do not go beyond the tens of thousands in practice [117]-[123], which is a practical constraint for the number of transitory parameters per unit in the GI-LSTM architecture.

Algorithm 5.1: GI-LSTM training

Input: $\{(\boldsymbol{x}(1), \boldsymbol{y}(1)), \dots, (\boldsymbol{x}(n), \boldsymbol{y}(n))\}$ *//assumed to be normalized*
Set values:
  n_h *//number of hidden neurons*
  n_{ss} *//number of subsequences*
  ss_{length} *//training subsequence length*
  I_{max} *//maximum number of iterations before stopping*
  n_{fails} *//number of consecutive fails*
  $n_{fails-max}$ *//max number of consecutive fails before stopping*
  $E_{train-min} \leftarrow inf$ *//minimum training MSE*
  $E_{val-min} \leftarrow inf$ *//minimum validation MSE*
Initialize: $\boldsymbol{\theta}$ //random initialization of weights
Divide dataset: D_{train}, D_{val}, D_{test} //division keeping temporal order
for j = 1 to I_{max}
  for l = 1 to n_{ss}

//----------Forward pass----------//

Extract: $D_{train}^{(l)}$ *//Extract lth training subsequence from* D_{train}

for k = 1 to ss_{length}

Compute: $\boldsymbol{a}(k), \boldsymbol{i}(k), \boldsymbol{w}_{\boldsymbol{f}_s}, \hat{\boldsymbol{f}}_s, \boldsymbol{o}(k), \boldsymbol{N}_s(k), \boldsymbol{c}(k), \boldsymbol{h}(k)$

Compute: E_{train} // using $\mathcal{L}^{(train)}$ for the lth subsequence

//----------Backward pass----------//

for k = 1 to ss_{length}

Compute: $\boldsymbol{\delta_h}(k)$, $\boldsymbol{\delta_o}(k)$, $\boldsymbol{\delta_c}(k)$, $\boldsymbol{\delta_i}(k)$, $\boldsymbol{\delta_{f_s}}(k)$, $\boldsymbol{\delta_a}(k)$, $\boldsymbol{\delta_z}(k)$, $\boldsymbol{\delta_\Theta}(k)$, $\boldsymbol{\delta_\theta}(k)$

$\boldsymbol{\delta_\theta} \leftarrow \sum_{j=1}^{ss_{length}} \boldsymbol{\delta_\theta}(j)$

Update: $\boldsymbol{\theta}$ *//using any optimizer designed for this purpose*

//----------Validation Set Performance----------//

Compute: E_{val} *//using* $D_{validation}$ *and* $\mathcal{L}^{(val)}$ *while keeping temporal order*

//----------Stopping criteria----------//

if $E_{val-min} > E_{val}$ **and** $E_{train-min} > E_{train}$**:** //Storing best performance and optimal parameters//

$\boldsymbol{\theta}_{opt} \leftarrow \boldsymbol{\theta}$, $E_{val-min} \leftarrow E_{val}$, $n_{fails} \leftarrow 0$

else:

$n_{fails} \leftarrow n_{fails} + 1$

if $n_{fails} > n_{fails-max}$: *//maximum consecutive fails for reducing* $E_{val-min}$ *or* $E_{train-min}$

break *//ending main for loop*

return: $\boldsymbol{\theta}_{opt}$ //Maximum number of iterations reached//

5.2 Experimental Setup

The GI-LSTM network performance is assessed by using the datasets described in Chapter 4 in addition to two popular benchmarks in the RNN area, the Copy memory dataset [28] described in the following paragraphs. This new dataset belongs to the category of multiclass classification, $\boldsymbol{y}(k) \in \mathbb{R}^{n_{class}}$, a category of problems that often uses the cross-entropy (CE) loss function (5.2.1) for the training process, due to its capability to embed categorical information through a probabilistic-like function.

$$\mathcal{L}_{CE} = \sum_{j=1}^{k} \sum_{i=1}^{n_{class}} [\boldsymbol{y}(j)]_i \, log([\hat{\boldsymbol{y}}(j)]_i) \tag{5.2.1}$$

The experiments with the Copy memory dataset are also performed with the standard LSTM to maintain a baseline to compare with; however, the E-LSTM, the SARIMA and the CW-RNN algorithms are not considered, due to model incompatibility in the first two cases and low performance results across experiments in the last case (Chapter 4).

5.2.1 Copy memory dataset

The copy memory is a synthetic dataset initially proposed in [28] and has been extensively used to evaluate the capability of a network to remember patterns across long-term delays [118], [119], [121]-[124]. At a high level, the task consists of introducing a pattern to the model at hand, followed by a large delay after which a trigger indicates to the network to output the pattern initially presented.

In more detail, the copy memory dataset is composed of an input sequence of length $T_{delay} + 2T_{pattern}$, where the first $T_{pattern}$ elements in the sequence are chosen uniformly at random from the set of n_{sym} symbols $\{a_1, \dots, a_{n_{sym}}\}$, creating the pattern to be remembered. The following $T_{delay} - 1$ elements consist of a 'dummy' symbol, $a_{n_{sym}+1}$, causing the desired delay. Next, a trigger symbol, $a_{n_{sym}+2}$, is presented to signal the network to output the pattern. Finally, the dummy symbol is used for the last $T_{pattern}$ elements in the input sequence. The output sequence, of the same length as the input sequence, is composed of $n_{sym} + 1$ classes $\{c_0, c_1, \dots, c_{n_{sym}}\}$ whose $T_{delay} + 2T_{pattern}$ elements are the 'zero' class, c_0, everywhere except for the last $T_{pattern}$ elements after the trigger symbol, $a_{n_{sym}+1}$, which correspond to the classes of each of the $T_{pattern}$ symbols encountered in the initial input pattern.

5.2.2 Implementation details

For the GI-LSTM implementation the MATLAB 2020b environment was utilized for the simulations. Also, the Adam optimization algorithm [41], [42] was again used across all experiments, including any additional LSTM simulations presented in this chapter. Adam's hyperparameters were set to standard values, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The validation loss, $\mathcal{L}^{(val)}$, was used as one of the stopping criteria across experiments for the training process, as indicated in Section 5.1.5, with a threshold value of $m_{fails} = 4000$.

Experiments were carried out using different sizes of the GI-LSTM in terms of the number of hidden units, sizes from the set $\{2^0, 2^1, \ldots, 2^8\}$ were utilized over the validation set, with 20 repetitions for each size; the loss function value producing the best average validation performance was selected. Simulations were performed in the Beluga, Graham and Narval server clusters, operated by the Digital Research Alliance of Canada, using 2.4GHz CPUs. For each of the selected sizes the resulting performance in the testing set is reported.

5.3. Experimental Results and Analysis

In this section the GI-LSTM results for each introduced dataset, from the current and previous chapters, are presented together with some of the results from Section 4.3 to facilitate comparison across networks. The average (μ) and standard deviation (σ) of the RMSE value (or cross entropy for classification problems) are provided as performance indicators. Also, the following metrics are provided: number of hidden units and number of parameters (n_h, $n_{\boldsymbol{\theta}}$).; average training time per iteration ($\bar{t}_{iter}$); average time and number of iterations to achieve the optimal loss function value ($\bar{t}^{(opt)}$, $\bar{I}_{iter}^{(opt)}$). The top two GI-LSTM models, from a small set of possible configurations and based on the validation-set performance (minimum $\mu_{val} + \sigma_{val}$), are

provided. Also, bold text is used in the tables to represent the network with best testing performance (minimum $\mu_{test} + \sigma_{test}$).

The hyperparameter values for the GI-LSTM configurations are chosen based on easy-to-access information in the datasets: seasonality, maximum length of potential patterns, human-driven seasonalities (hours, days, weeks, months, etc.).

GI-LSTM interpretability plots are also provided, i.e., $\overline{\widehat{\boldsymbol{\omega}}_s}$ (5.1.21), where it should be noted that a temporal dependence in the plots at d implies a self-lag dependence in the time series at $d + 1$, since the input is the output with a lag of 1. Also, the qualitative description 'parameter-efficient' will be used across this section to describe when the GI-LSTM achieves a similar performance to the LSTM using significantly less parameters. Furthermore, validation-set performance results for different GI-LSTM sizes are presented, in the form of plots, as additional information to make performances comparisons under similar numbers of parameters.

5.3.1 Synthetic datasets

Results for the Switching datasets are presented in Table 5.3.1-Table 5.3.2 and Fig. 5.3.1-Fig. 5.3.4. In the case of the Switching-100 dataset, the best GI-LSTM variant outperforms the LSTM and its augmented-input variation, showing a similar performance to the proposed E-LSTM but using an order of magnitude less of parameters. Also, the interpretability plot shows that the GI-LSTM can detect the nonlinear lagged dependence at 22 without the aid of the DC correlation, as opposed to the E-LSTM. Furthermore, in terms of parameter efficiency the GI-LSTM prevails over the standard LSTM across different networks sizes and remains competitive with the E-LSTM.

Table 5.3.1. Results for the Switching-100 dataset ($k_i = 22$). Lower bound RMSE = 0.4227, GI-LSTM forward-backward pass length is set to 1000.

Switching-100 switching ($p_l = 22$)											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	0.6232	0.0980	0.6810	0.0085	0.6771	0.0182	128	66689	1.51	381.37	575.87
E-LSTM$_{22}$	**0.5894**	**0.0080**	**0.6198**	**0.0047**	**0.6060**	**0.0044**	**128**	**83329**	**1.97**	**69.82**	**137.55**
LSTM-A	0.6014	0.0052	0.6204	0.0029	0.6156	0.0040	128	67201	1.51	69.32	104.67
GI-LSTM$_{100}$	0.5817	0.0084	0.6175	0.0118	0.6087	0.0108	32	7585	1.11	310.65	344.82
GI-LSTM$_{100\text{-}4}$	0.5885	0.0128	0.6307	0.0105	0.6155	0.0102	64	27841	1.52	346.45	526.60

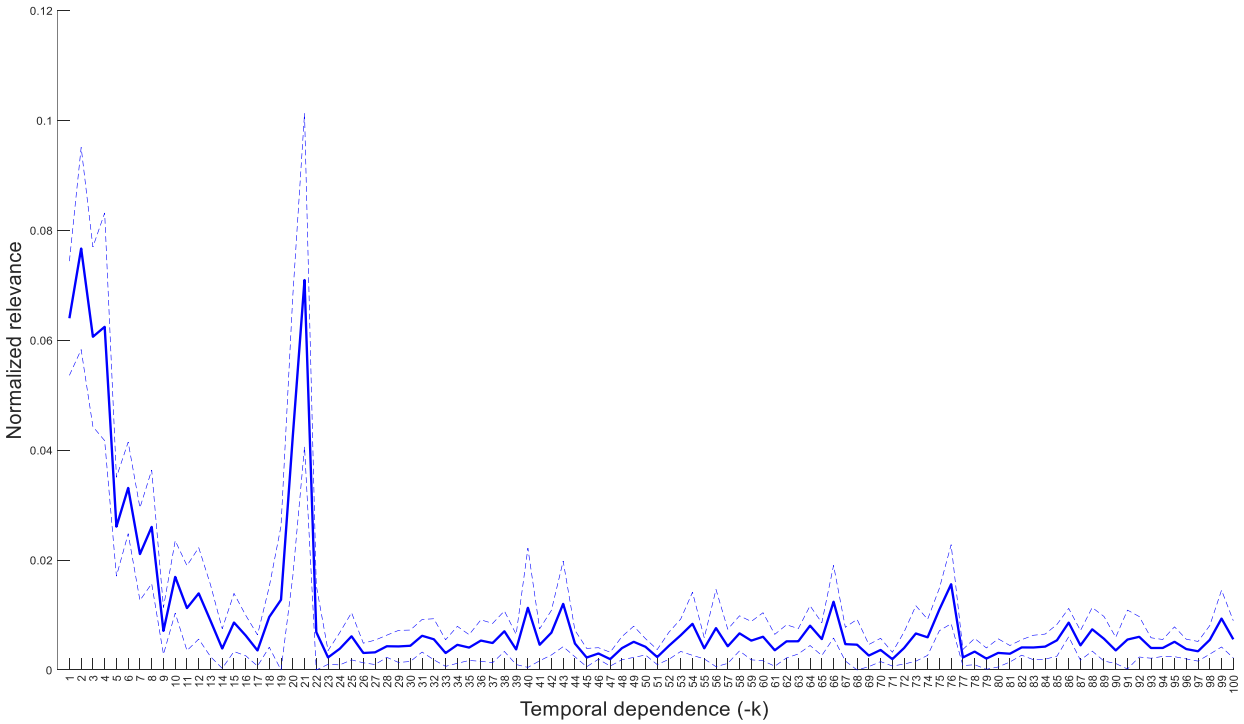


Fig. 5.3.1. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Switching-100 dataset.

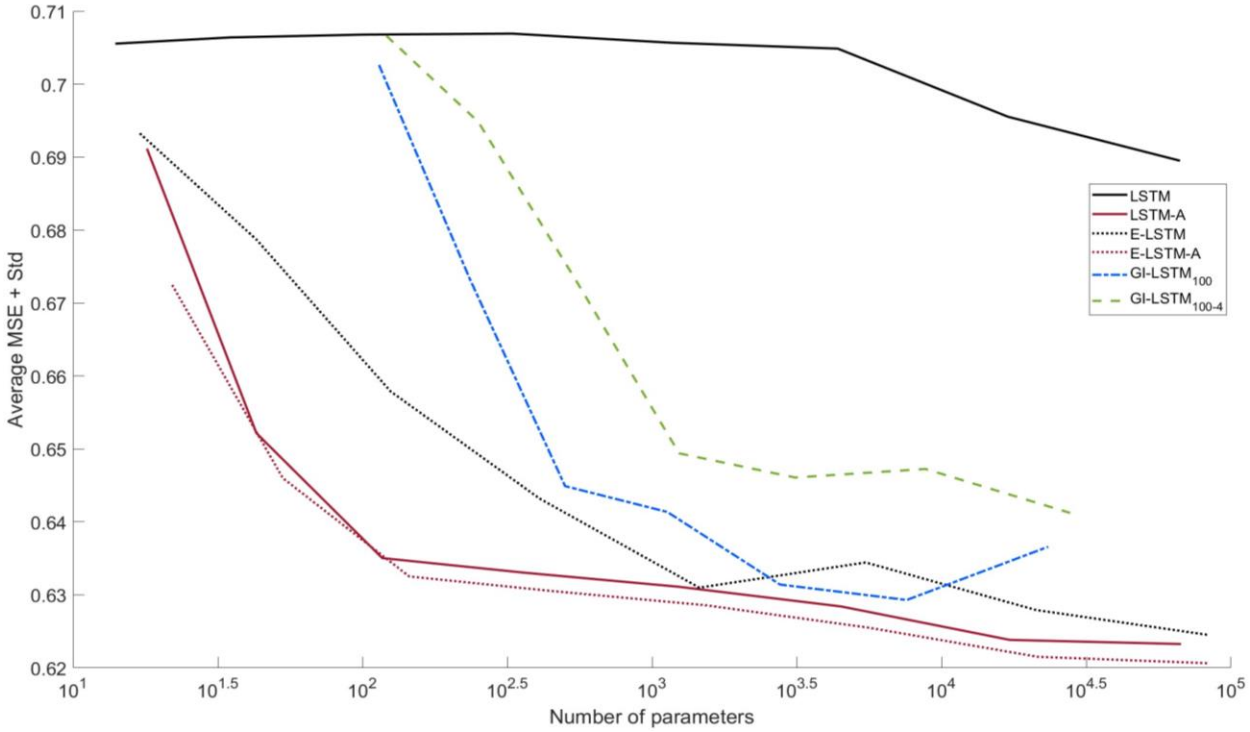


Fig. 5.3.2. Validation set performance across different sizes for the Switching-100 dataset.

For the Switching-01 dataset the best GI-LSTM variant outperforms the standard LSTM and E-LSTM, but the augmented-input variations of these networks remain on top. In this case, as in Section 4.3.1, the results can be attributed to the low-frequency switching. However, the GI-LSTM is able again to precisely detect the nonlinear lagged dependence at 50 and remains parameter-efficient with respect to the standard LSTM and E-LSTM networks.

Table 5.3.2. Results for the Switching-01 ($k_i = 50$), with a lower bound RMSE = 0.4095 and a GI-LSTM forward-backward pass length of 1000.

Switching-01 ($p_l = 50$)											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	0.6988	0.0041	0.7675	0.0012	0.7258	0.0116	32	4385	0.53	510.15	270.38
E-LSTM$_{50}$	0.6484	0.0110	0.7252	0.0082	0.7207	0.0204	64	21185	1.19	459.35	546.63
LSTM-A	0.6396	0.0111	0.6858	0.0041	0.6777	0.0181	8	361	0.41	413.72	169.63
E-LSTM$_{50}$-A	**0.6490**	**0.0064**	**0.6858**	**0.0052**	**0.6784**	**0.0096**	**4**	**145**	**0.67**	**2666.25**	**1786.39**
GI-LSTM$_{100}$	0.6066	0.0141	0.7074	0.0106	0.7270	0.0239	8	1129	0.76	2339.9	1778.324
GI-LSTM$_{100\text{-}4}$	0.6396	0.0114	0.7240	0.0130	0.7163	0.0135	8	1241	0.83	2315.6	1921.948

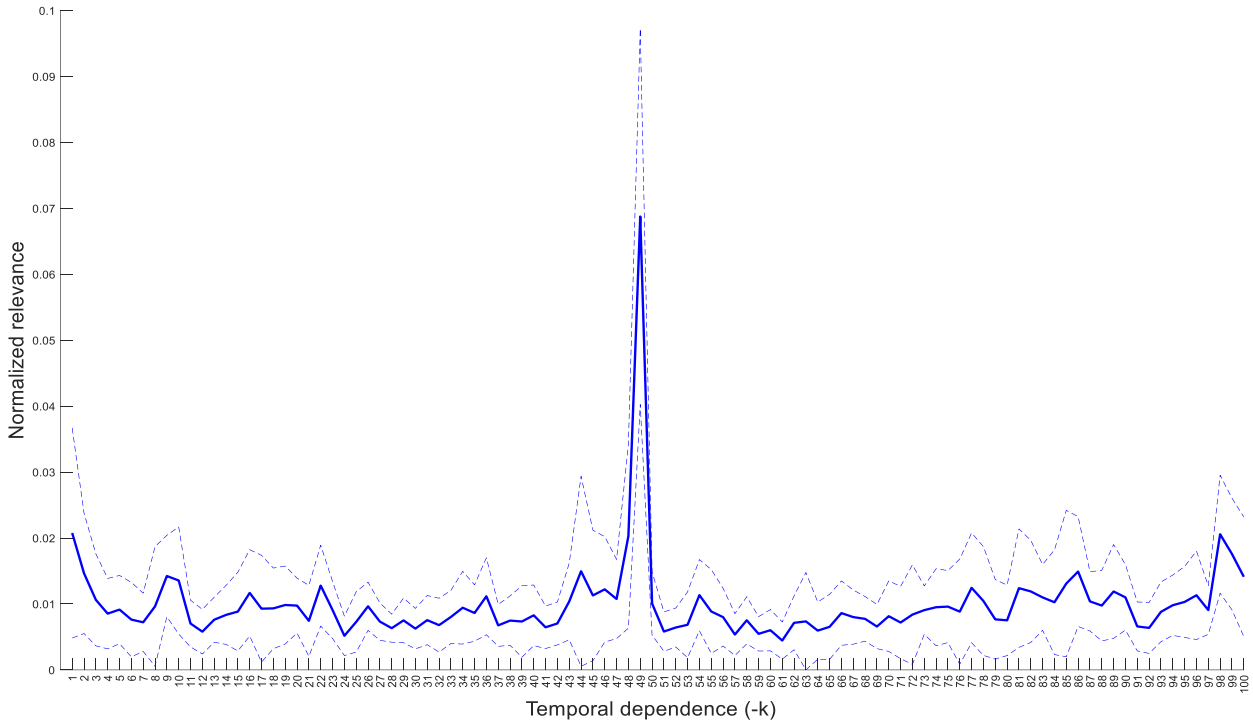


Fig. 5.3.3. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Switching-01 dataset.

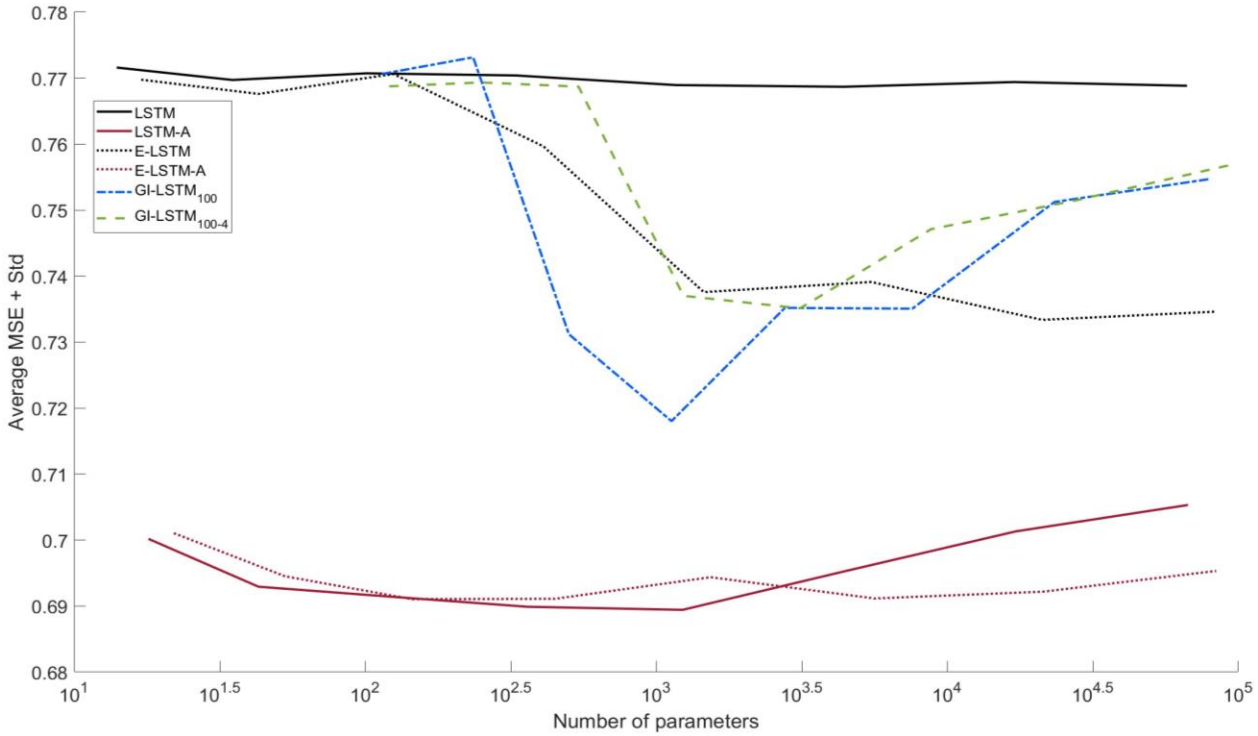


Fig. 5.3.4. Validation set performance across different sizes for the Switching-01 dataset.

For the Binary sequence dataset, results (Table 5.3.3) show that both configurations of the GI-LSTM, one using a lag-memory value equal to the embedded sequence's length ($q_1 = 112$), and the other using a larger lag-memory value ($q_1 = 200$) have better performance than the alternative networks, with performance close to the lower bound. Also, by analyzing the interpretability plot (Fig. 5.3.5) the relevance that the GI-LSTM gives to the long-term dependencies can be observed, potentially explaining the better performance. In addition, Fig. 5.3.6 indicates that the optimal validation performance obtained by the alternative networks can be achieved by the GI-LSTM using less parameters (again by an order of magnitude).

Table 5.3.3. Results for the Binary Sequence dataset ($k_i = 29$), with a lower bound RMSE = 0.4357 and a GI-LSTM forward pass length of 1120

Binary Sequence											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	0.4586	0.0037	0.4624	0.0039	0.4645	0.0041	8	329	1.488	1479.67	0.6116
E-LSTM$_{29}$	0.4314	0.0051	0.4520	0.0052	0.4582	0.0071	128	83329	7.242	39.02	0.0785
GI-LSTM$_{112}$	**0.4275**	**0.0020**	**0.4416**	**0.0014**	**0.4390**	**0.0013**	**128**	**81025**	**15.403**	**53.75**	**0.229975**
GI-LSTM$_{200}$	0.4278	0.0033	0.4434	0.0021	0.4419	0.0029	128	92289	18.527	566.15	2.913628

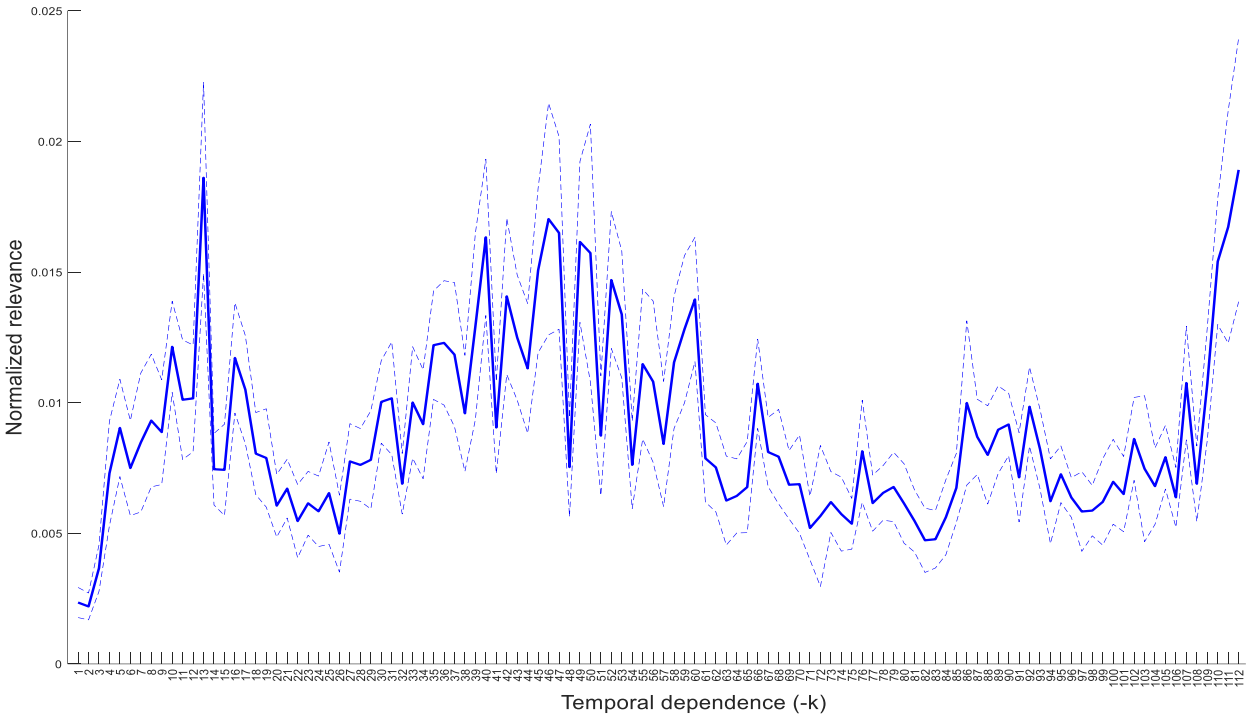


Fig. 5.3.5. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Binary sequence dataset.

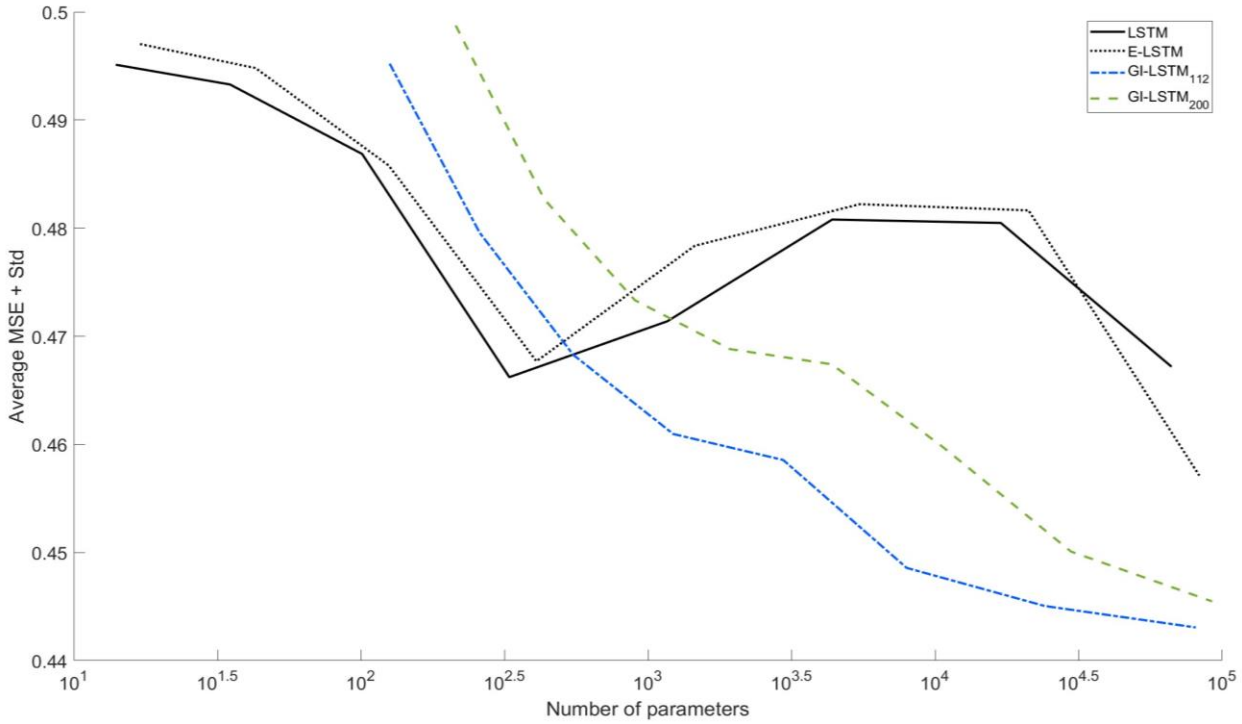


Fig. 5.3.6. Validation set performance across different sizes for the Binary sequence dataset.

5.3.2 Real-world datasets

In the Chickenpox dataset the GI-LSTM outperforms the standard LSTM and E-LSTM models, but the augmented E-LSTM remains on top (Table 5.3.3). It is worth noting that the GI-LSTM produces competitive performance with an acceptable number of parameters, with respect to the other networks. Furthermore, the dependence relevance (Fig. 5.3.7) is consistent with the highest-value nonlinear dependence identified by the DC measure and used for the E-LSTM ($p = 24$). In this dependence relevance, less value is assigned to the yearly data's seasonality (lag of 12 in the time series and equivalent to the dependence at 11 in the interpretability plot). In relation to the performance across different network sizes (Fig. 5.3.8), the GI-LSTM model remains consistently better in relation to the alternative networks.

Table 5.3.4. Results for the Chickenpox dataset ($k_i = 24$) a GI-LSTM forward pass length of 400.

Chickenpox											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	146.46	41.48	111.05	4.93	143.31	32.44	128	66689	0.168	8874.43	0.4141
E-LSTM$_{24}$	216.48	9.76	114.20	4.70	121.36	7.33	2	43	0.0601	4701.95	0.0785
E-LSTM$_{24}$-A	**201.42**	**0.84**	**100.72**	**1.75**	**108.99**	**1.53**	**1**	**22**	**0.17**	**8819.90**	**0.4165**
GI-LSTM$_{12}$	148.71	17.65	106.31	6.94	109.24	12.21	4	149	0.058	5736.3	0.0924
GI-LSTM$_{120}$	179.56	1.03	106.43	0.204	109.87	0.902	1	134	0.039	17535.95	0.1900

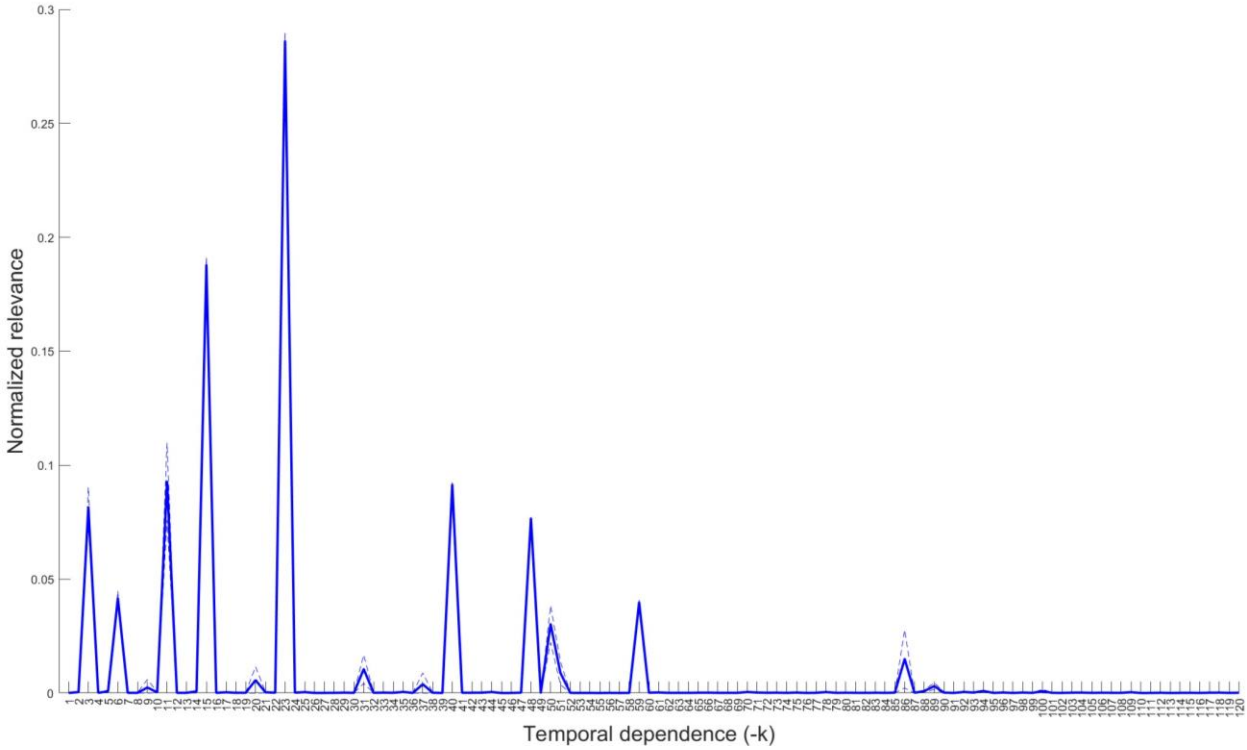


Fig. 5.3.7. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Chickenpox dataset.

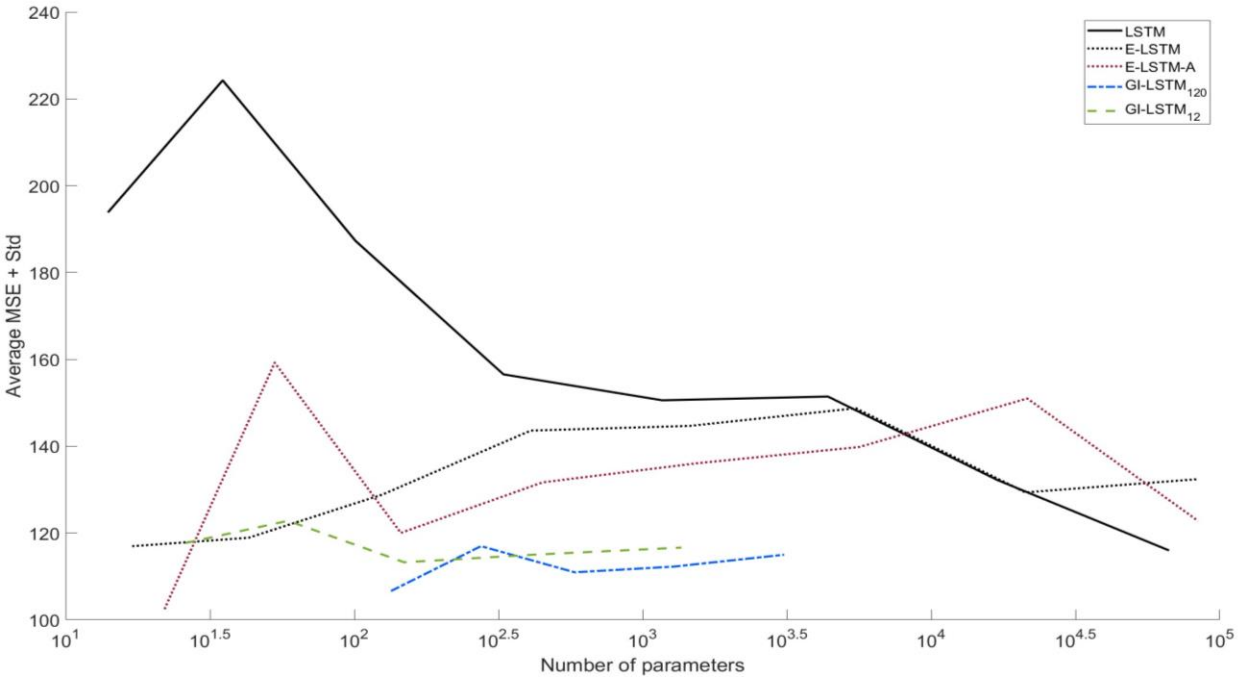


Fig. 5.3.8. Validation set performance across different sizes for the Chickenpox dataset.

For the Sunspots dataset, little improvement is obtained through the best GI-LSTM model in comparison to the other models. Nevertheless, when analyzing the interpretability plots of both GI-LSTM configurations, Fig. 5.3.9-Fig. 5.3.10, it can be observed that the 10-to-11-year seasonality in the dataset (120-132 lagged values in the timeseries) receives little relevance in the second-best configuration, while most of it is allocated for the last three years of information. This relevance distribution could potentially explain why little improvement is achieved in the model even when compared to the standard LSTM.

Table 5.3.5. Results for the Sunspots dataset ($k_i = 12$) and a GI-LSTM forward pass length of 1440.

Sunspots											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(s)
LSTM	24.22	0.19	24.89	0.21	22.71	0.27	4	101	0.2079	3386.75	704.11
E-LSTM$_{15}$	24.22	0.21	24.93	0.18	22.67	0.18	4	125	0.2910	3179.8	925.32
GI-LSTM$_{144}$	24.32	0.02	25.05	0.03	22.66	0.02	1	158	0.2771	4884.4	1353.47
GI-LSTM$_{12}$	**23.94**	**0.23**	**24.95**	**0.19**	**22.48**	**0.14**	**16**	**1361**	**0.4440**	**498.6**	**221.38**

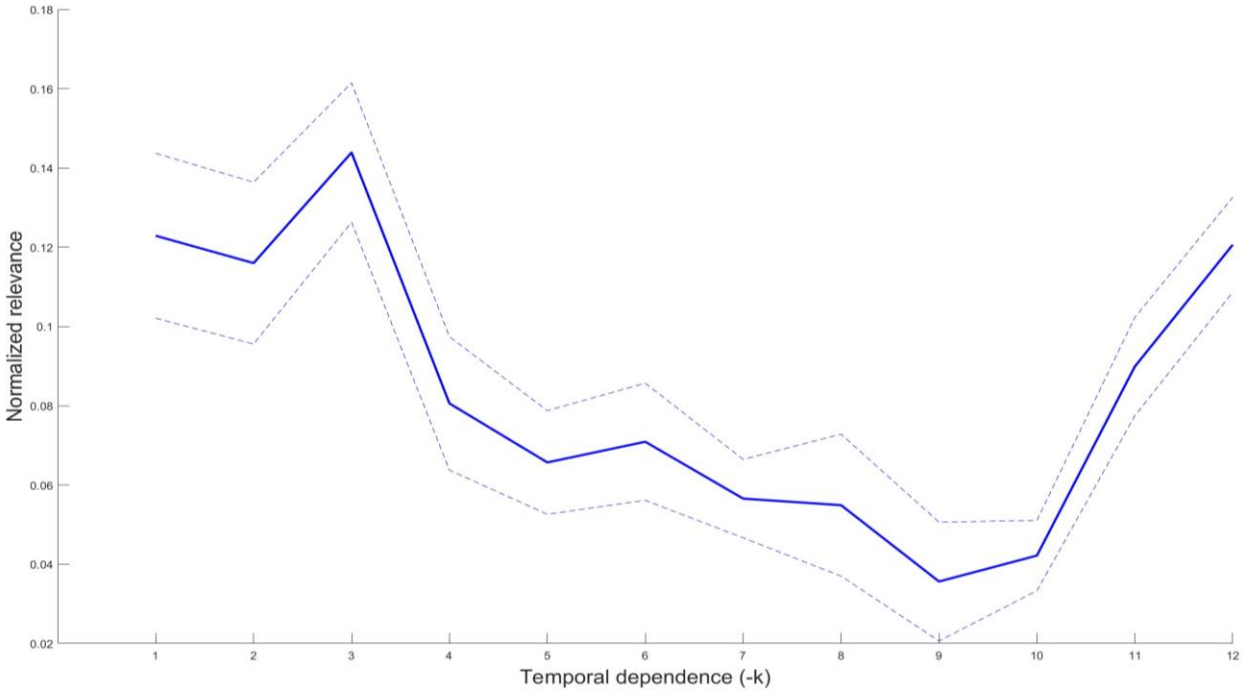


Fig. 5.3.9. Temporal-dependence relevance in the GI-LSTM memory-group 1, best configuration, for the Sunspots dataset,

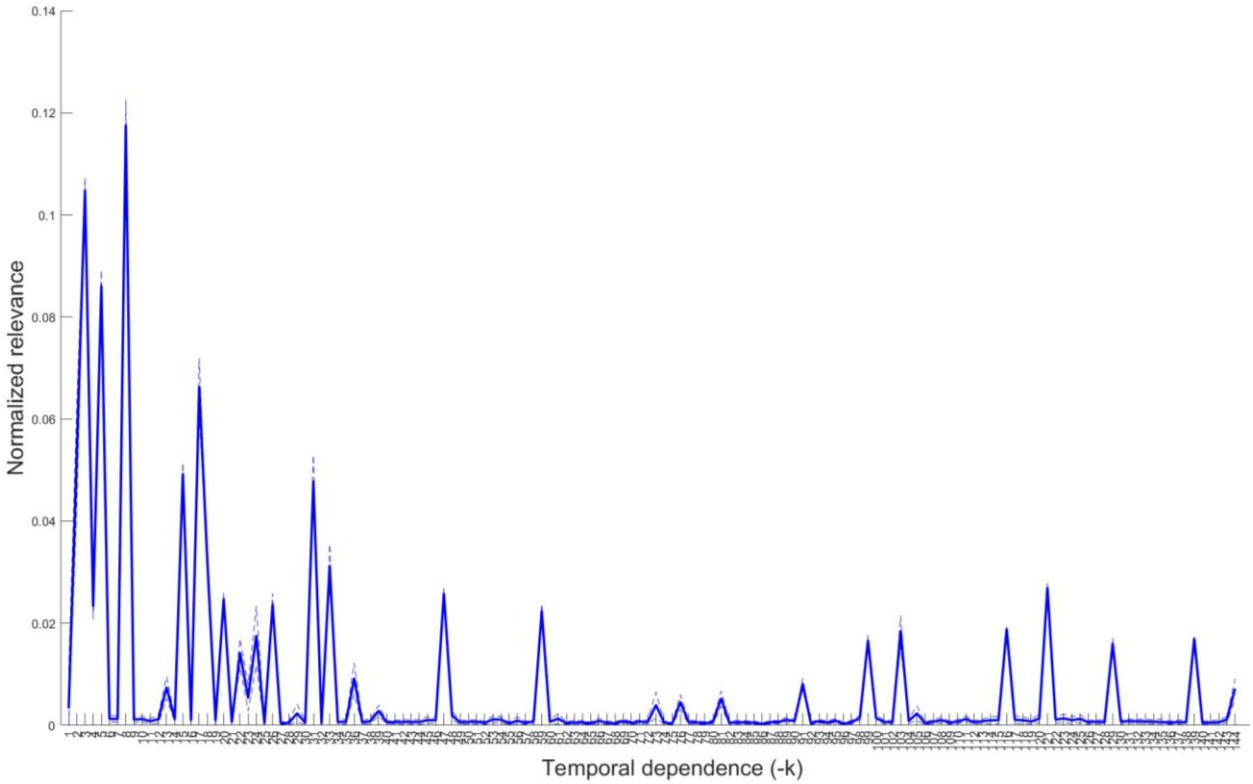


Fig. 5.3.10. Temporal-dependence relevance in the GI-LSTM memory-group 1, second-best configuration, for the Sunspots dataset.

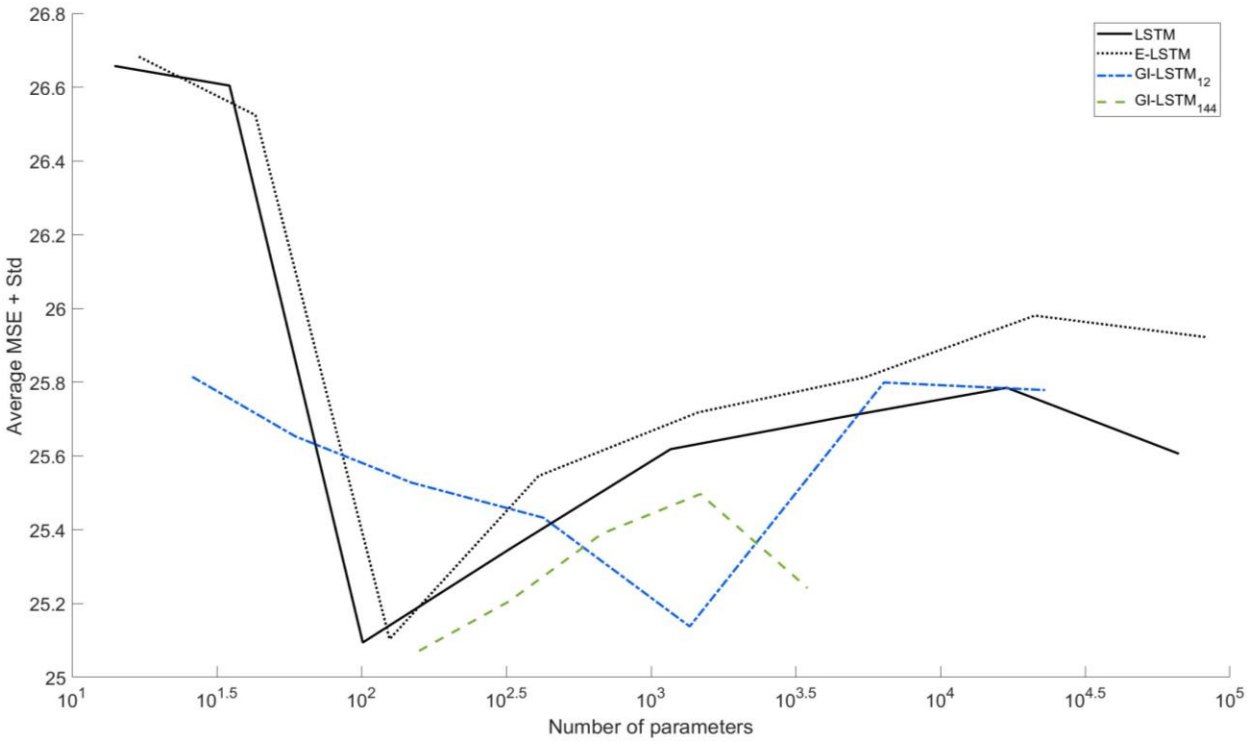


Fig. 5.3.11. Validation set performance across different sizes for the Sunspots dataset.

For the Power Consumption dataset, the GI-LSTM networks (Table 5.3.6) achieve the best performance; specifically, the configuration using two memory groups. The latter is of interest when considering that in theory both configurations possess the same temporal reach (derived from (5.1.1)-(5.1.4)), $168 = 24 * (6 + 1)$. One interpretation of the latter outcome is potential redundancy in the information across the previous 168 time units (hours). In more detail, by observing Fig. 5.3.12 - Fig. 5.3.13 it can be noticed that approximately 45% of the relevance is given to the information occurring in the last 24 hours, while the remaining relevance seems to be more uniformly distributed across the remainder of the previous information. Also, the interpretability plot in Fig. 5.3.14 shows a pattern in the relevance distribution, with pronounced local maxima occurring at lag values that are multiples of 24, after the first 48-time dependencies. This can indicate that information compression has no negative effect in the performance for this dataset, a trend that remains consistent across different sizes of the GI-LSTM network (Fig. 5.3.15).

Table 5.3.6. Results for the Power Consumption dataset ($k_i = 24$)) and a GI-LSTM forward pass length of 1344

Power Consumption											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	65.00	1.00	63.75	0.58	65.58	1.40	32	4385	6.18	1452.18	2.493
E-LSTM$_{24}$	65.16	1.54	63.08	0.53	63.71	0.63	128	83329	22.15	213.53	1.314
LSTM-A	65.35	0.92	63.61	0.42	64.74	0.57	32	4513	6.25	619.13	1.075
E-LSTM$_{24}$-A	65.08	1.20	62.98	0.52	63.47	0.63	64	21505	12.50	221.45	0.769
GI-LSTM$_{168}$	61.81	1.01	59.81	0.22	60.42	0.46	128	88193	29.59	270.05	2.22
GI-LSTM$_{24\text{-}6}$	**62.34**	**0.75**	**59.98**	**0.32**	**60.18**	**0.39**	**128**	**87169**	**37.12**	**186.05**	**1.918**

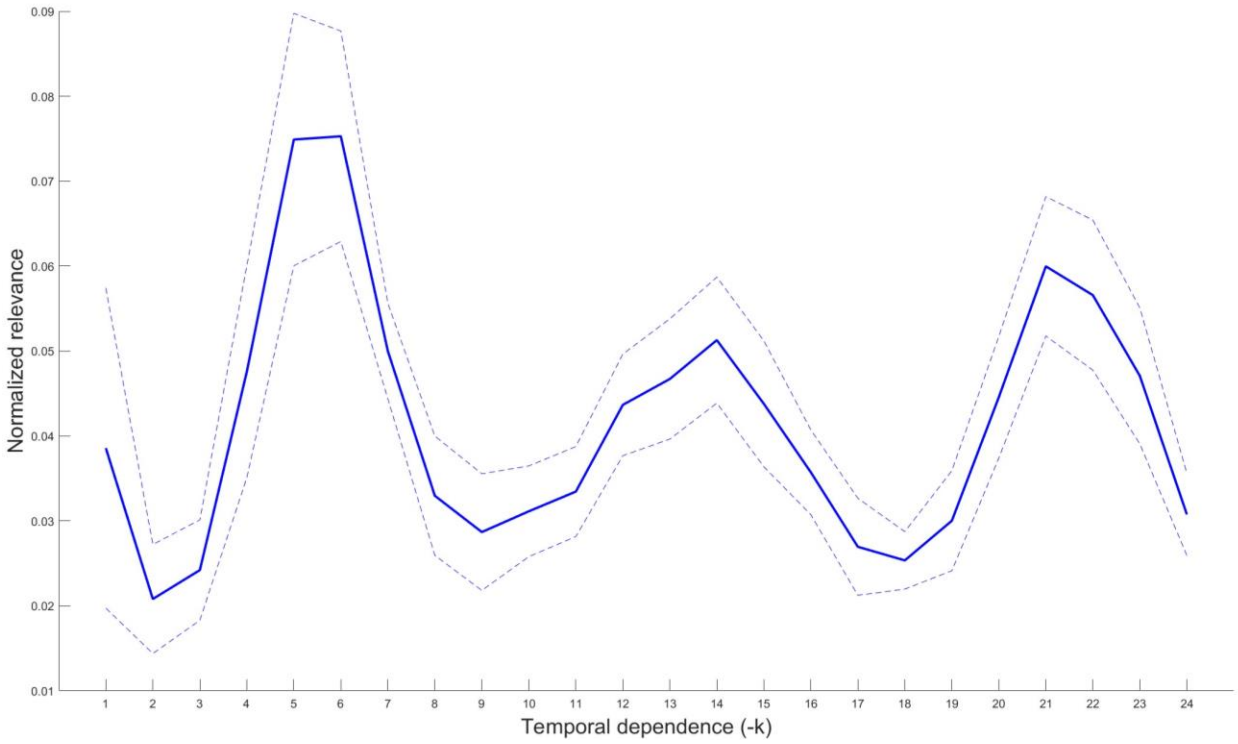


Fig. 5.3.12. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Power consumption dataset.

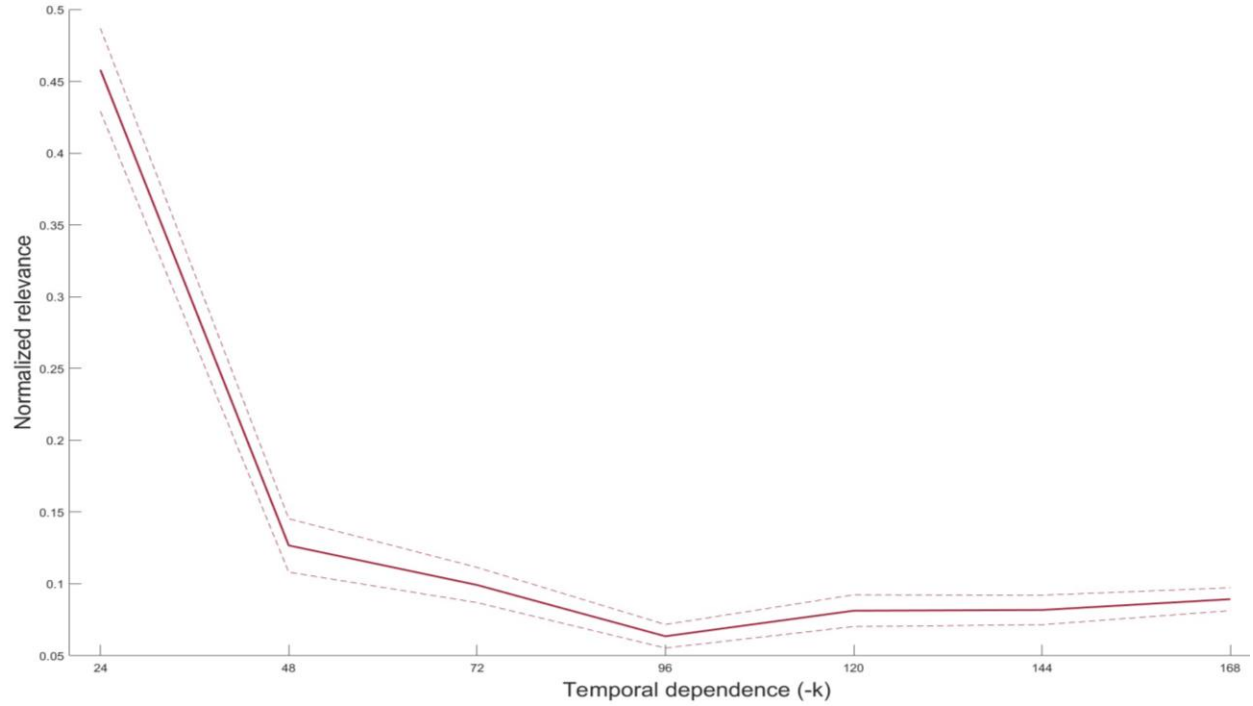


Fig. 5.3.13. Temporal-dependence relevance in the GI-LSTM memory-group 2, best configuration, for the Power consumption dataset.

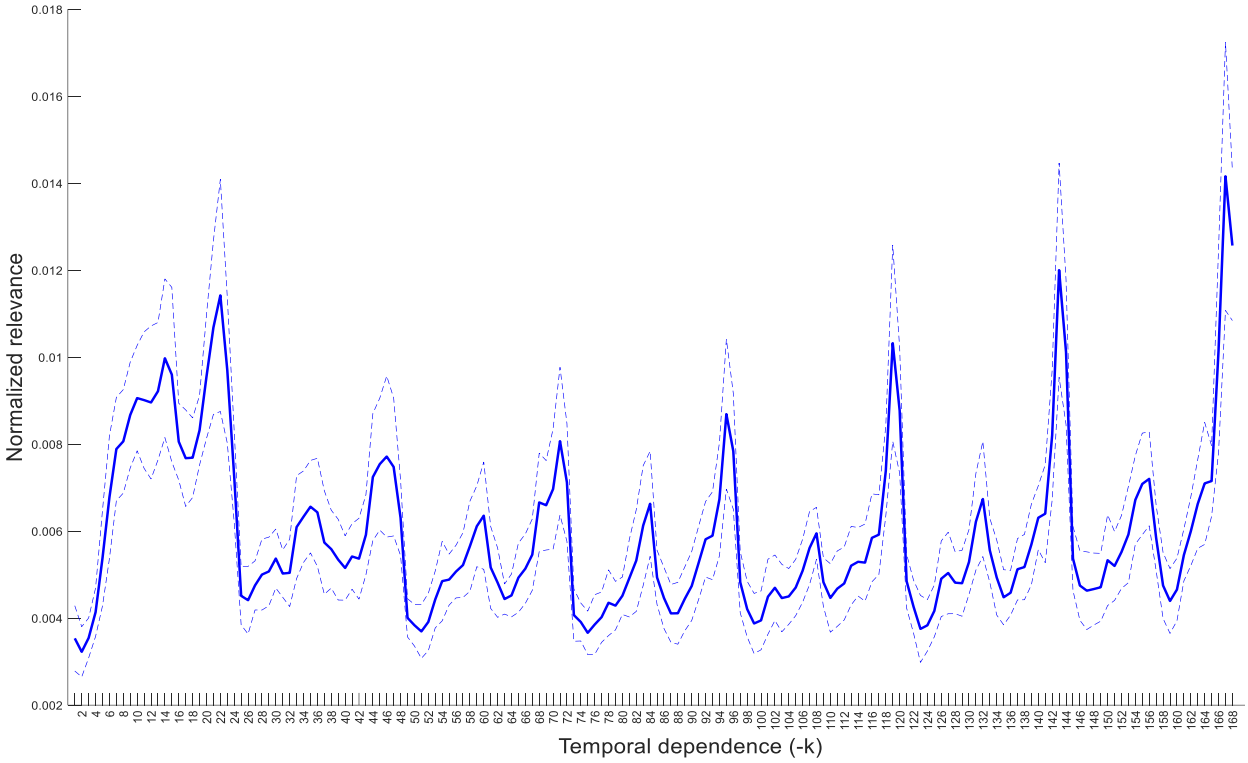


Fig. 5.3.14. Temporal-dependence relevance in the GI-LSTM memory-group 1, second-best configuration, for the Power consumption dataset.

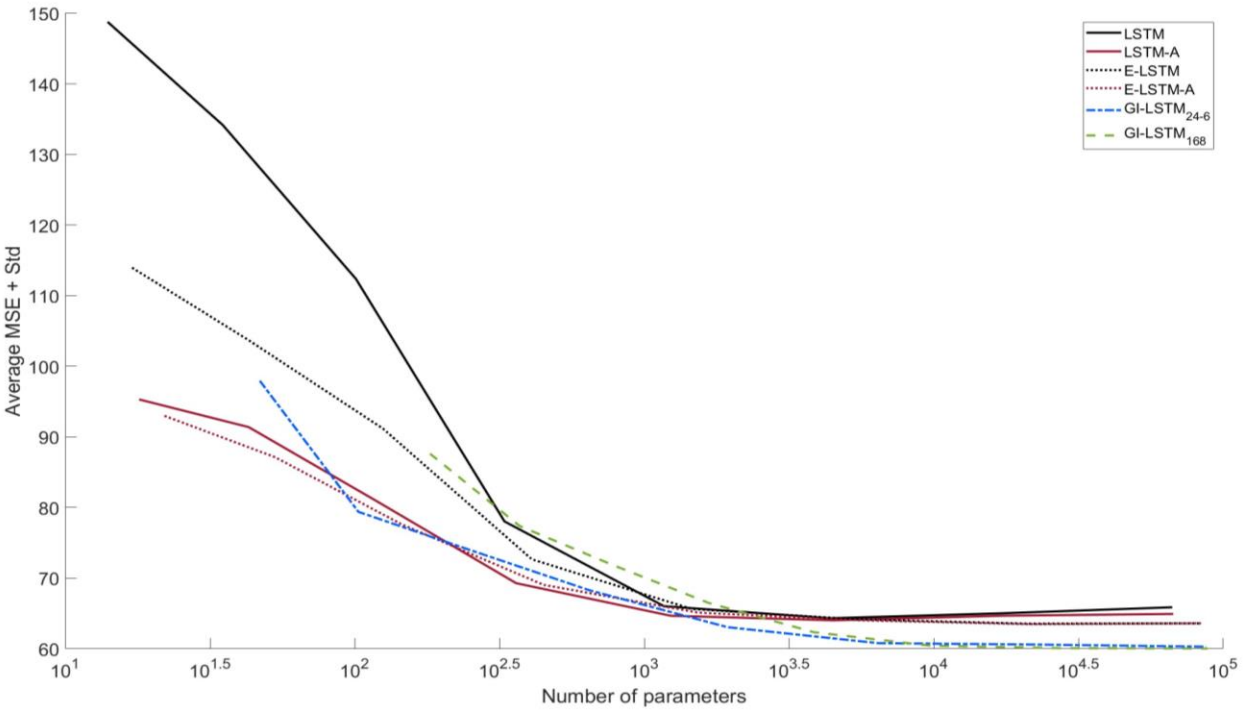


Fig. 5.3.15. Validation set performance across different sizes for the Power consumption dataset.

Similar to the previous case, in the Toronto temperature dataset the GI-LSTM configurations (Table 5.3.7) outperform the alternative models; both configurations are close in performance, despite the second-best using 15% less parameters due to the use of a second-order memory group. The latter effect can be further supported when observing the interpretability plots (Fig. 5.3.16-Fig. 5.3.18), where the relevance distribution shows a similar phenomenon to the Power Consumption dataset: a pattern of local maxima at multiples of 24 hours in the configuration using a single memory group (Fig. 5.3.16) and the configuration using two memory groups a relevance of approximately 45% for the previous 24 hours (Fig. 5.3.18). Additionally, the performance of the networks across different sizes (Fig. 5.3.19) shows that the GI-LSTM configurations remain parameter-efficient, and the performance trend is similar to that of the Power consumption dataset (Fig. 5.3.15). This performance trend resemblance is not further explored or analyzed in this work, but it could be hypothesized that the energy consumption of the US Eastern grid area, in the same time-zone as the city of Toronto, is heavily affected by the temperatures in the region which are in turn correlated with Toronto's temperature.

Table 5.3.7. Results for the Toronto temperature dataset and a GI-LSTM forward pass of 1344.

Toronto temperature											
	μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	0.6751	0.0029	0.6618	0.0011	0.6684	0.0015	16	1169	7.217	692.68	1.389
E-LSTM$_{24}$	0.6775	0.0023	0.6607	0.0020	0.6676	0.0014	8	409	9.961	1051.135	2.908
GI-LSTM$_{168}$	**0.6567**	**0.0027**	**0.6540**	**0.0008**	**0.6596**	**0.0008**	**64**	**27713**	**33.639**	**132**	**1.233**
GI-LSTM$_{24\text{-}13}$	0.6601	0.0021	0.6546	0.0011	0.6597	0.0008	64	23553	19.314	106.1	0.569

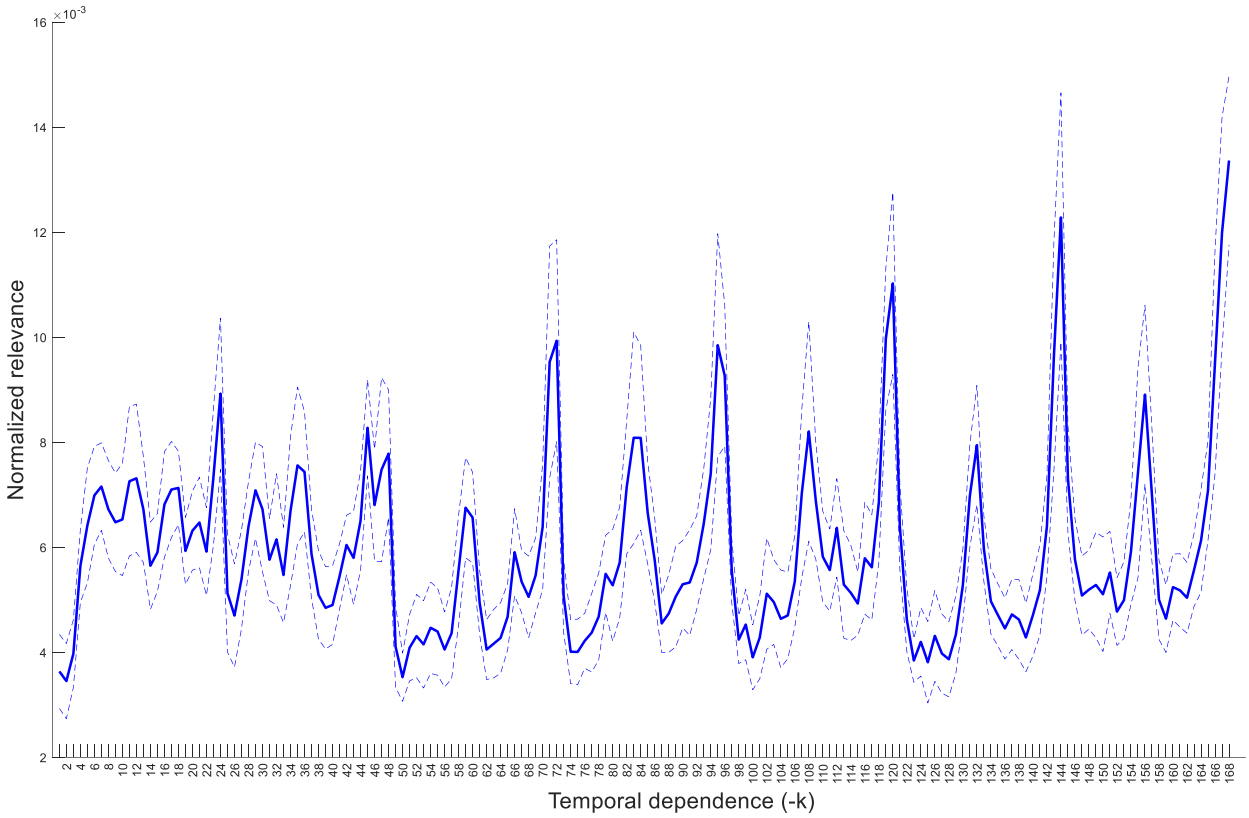


Fig. 5.3.16. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Toronto temperature dataset.

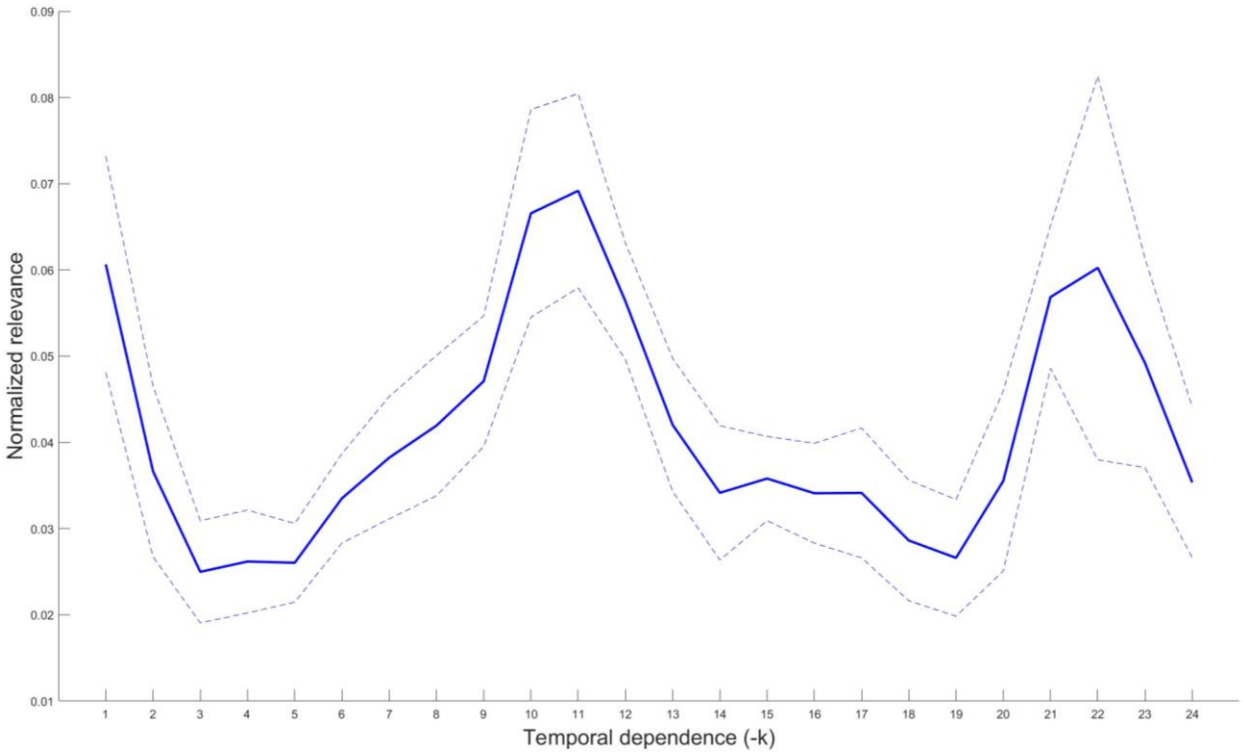


Fig. 5.3.17. Temporal-dependence relevance in the GI-LSTM memory-group 1, second-best configuration, for the Toronto temperature dataset.

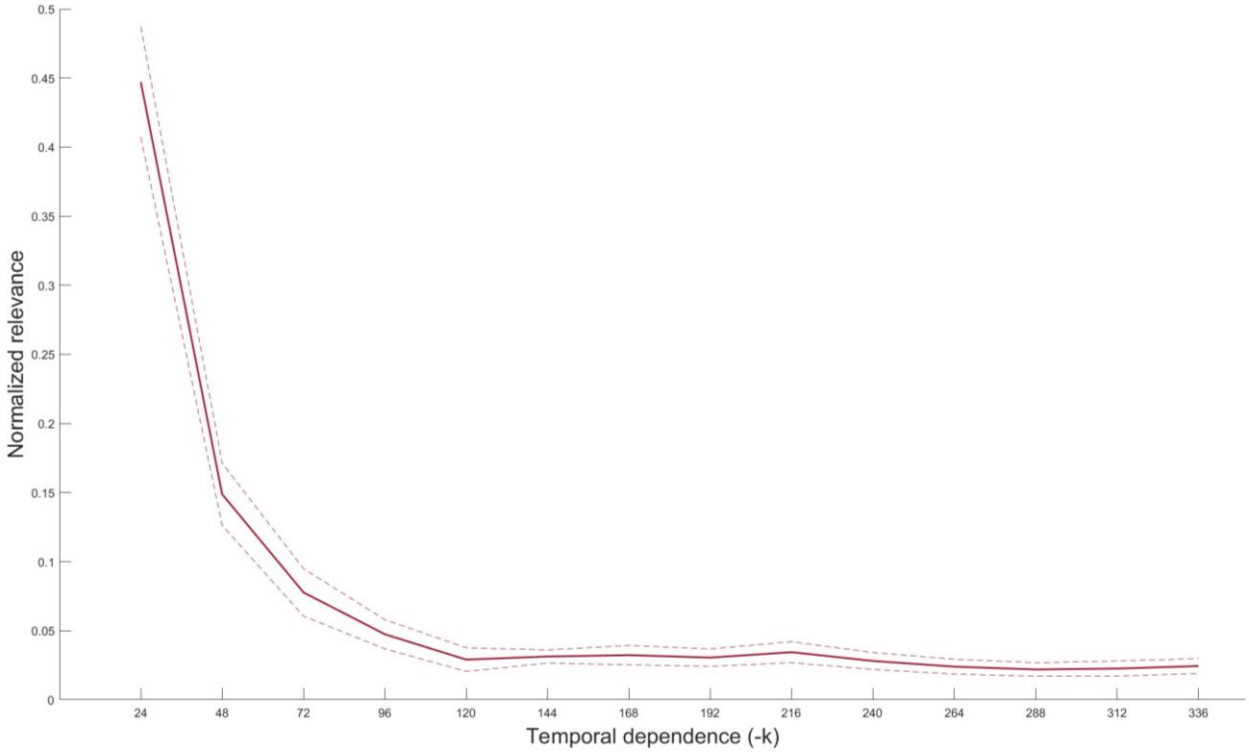


Fig. 5.3.18. Temporal-dependence relevance in the GI-LSTM memory-group 2, second-best configuration, for the Toronto temperature dataset.

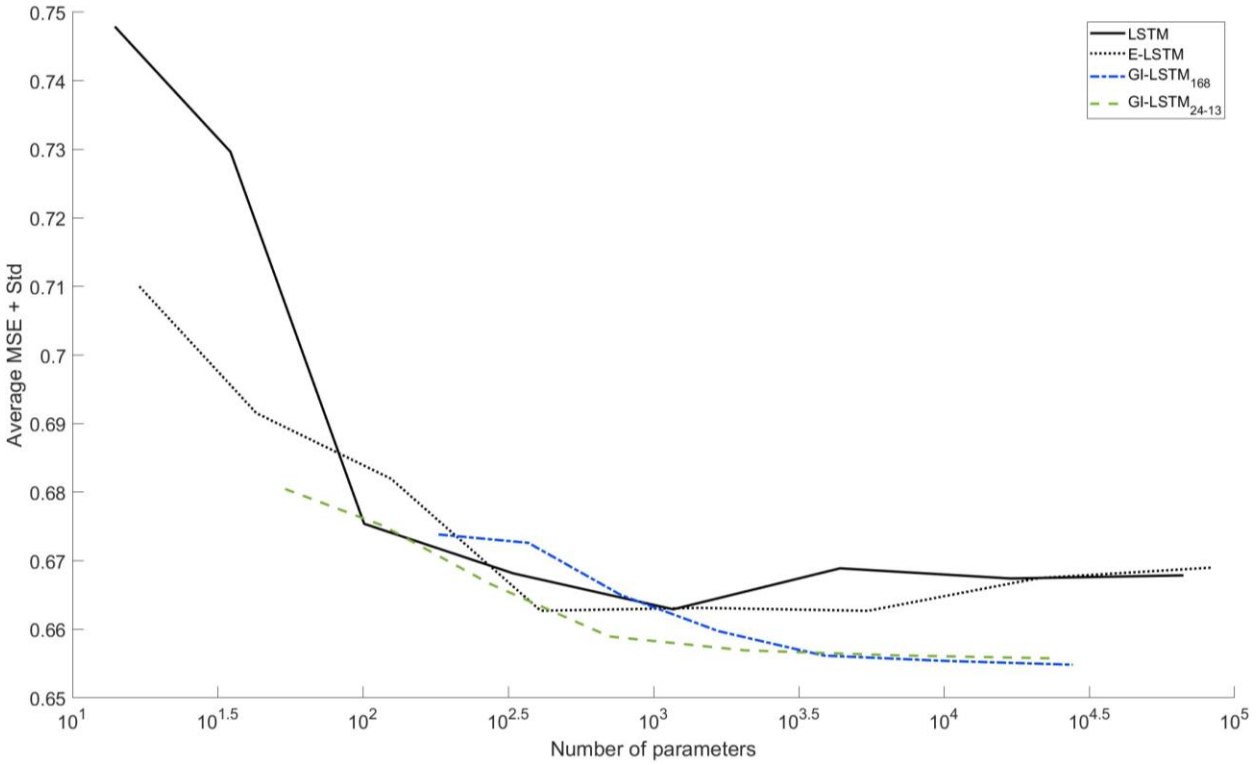


Fig. 5.3.19. Validation set performance across different sizes for the Toronto temperature dataset.

5.3.3 Copy memory dataset results and analysis

The copy memory dataset simulation results are studied separately due to the use of the CE loss function and the flexibility it allows for varying the delay, T_{delay}, between the introduction of the input patterns and the time to output their class. Experiments using delay values of 50, 200 and 400 were performed to test the long-term memory capability of the networks under the following conditions. First, $n_{sym} = 8$ and $T_{pattern} = 10$, as in the original copy memory task. Also, one-hot encoding is used for all the symbols, producing vectors of dimension 10. Since the encoding generates 0-1 values no input normalization is performed (which avoids the generation of large values in the input due to the scarcity of the symbols to be classified when T_{delay} is large). Additionally, batch processing is performed, and the Adam optimizer is used, with default hyperparameter values and learning rate $\alpha = 0.005$. Chrono initialization [122] is implemented for the standard LSTM, due to its proven efficacy to accelerate the learning process of long-term patterns. A maximum number of 100000 epochs for the LSTM and 40000-60000 epochs for the GI-LSTM are used during the training phase. Finally, a temporal reach equal to half the sequence length, $S/2$, in each experiment is used for all the GI-LSTM configurations.

Among the defined experimental conditions, it is important to highlight that avoiding normalization after the one-hot encoding also avoids the effect of unintentionally leaking information to the networks, since the mean-variance normalization would generate a very low variance for the features linked to the symbols to be reconstructed, greatly increasing the values of these features and decreasing the feature value associated with the dummy symbol, $a_{n_{sym}+1}$, giving away the relevance of the patterns.

For the result tables, in addition to the CE metrics, the total classification accuracy and the pattern reconstruction accuracy metrics are also reported; the pattern reconstruction accuracy reflects the true capability of the networks to learn and generalize the task, without the bias created by the dummy symbol that appears across most of the sequence length, i.e., in $T_{delay} + T_{pattern} - 1$ instances out of $\mathcal{S} = T_{delay} + 2T_{pattern}$.

In Table 5.3.8 the results for $T_{delay} = 50$, with the best performing sizes of the networks, is shown. As observed, the LSTM is essentially unable to perform the pattern reconstruction under the conditions of the experiment, despite the use of the chrono initialization, the use of a higher number of parameters (an order of magnitude higher than the GI-LSTM) and more training epochs. In contrast, the GI-LSTM is able to achieve near 100% accuracy in the testing set, showing its capability to generalize beyond the training and validation sets. Also, when analyzing the interpretability plot (Fig. 5.3.20) for this experiment, it can be observed that relevance is assigned to iterations beyond the trigger symbol, located at 10 in lag values. Consequently, information is propagated forward across the temporal connectivity generated in the memory groups.

The detailed evolution of the training process for this experiment can be observed in Fig. 5.3.21 - Fig. 5.3.22, where it can be seen that the average validation loss curves do not follow the trajectory of the training loss curves. This implies that the LSTM network is not able to generalize beyond the training set under these experimental conditions.

Table 5.3.8. Results for the Copy-memory-d50 dataset, 100 sequences and batch size of 100.

Copy-memory-d50												
		μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	Cross Entropy	0.2683	0.0158	0.2872	0.0065	0.2938	0.0130	64	17025	0.8482	4664.8	1.01
	Total accuracy(%)	89.78	0.65	88.93	0.44	88.75	0.45					
	Pattern accuracy(%)	28.51	0.54	22.62	3.08	21.67	3.07					
GI-LSTM35	Cross Entropy	**0.0019**	**0.0035**	**0.0028**	**0.0040**	**0.0033**	**0.0039**	**16**	**1729**	**0.2921**	**18876**	**1.53**
	Total accuracy(%)	**100**	**0**	**99.99**	**0.02**	**99.97**	**0.04**					
	Pattern accuracy(%)	**100**	**0**	**99.99**	**0.13**	**99.81**	**0.31**					

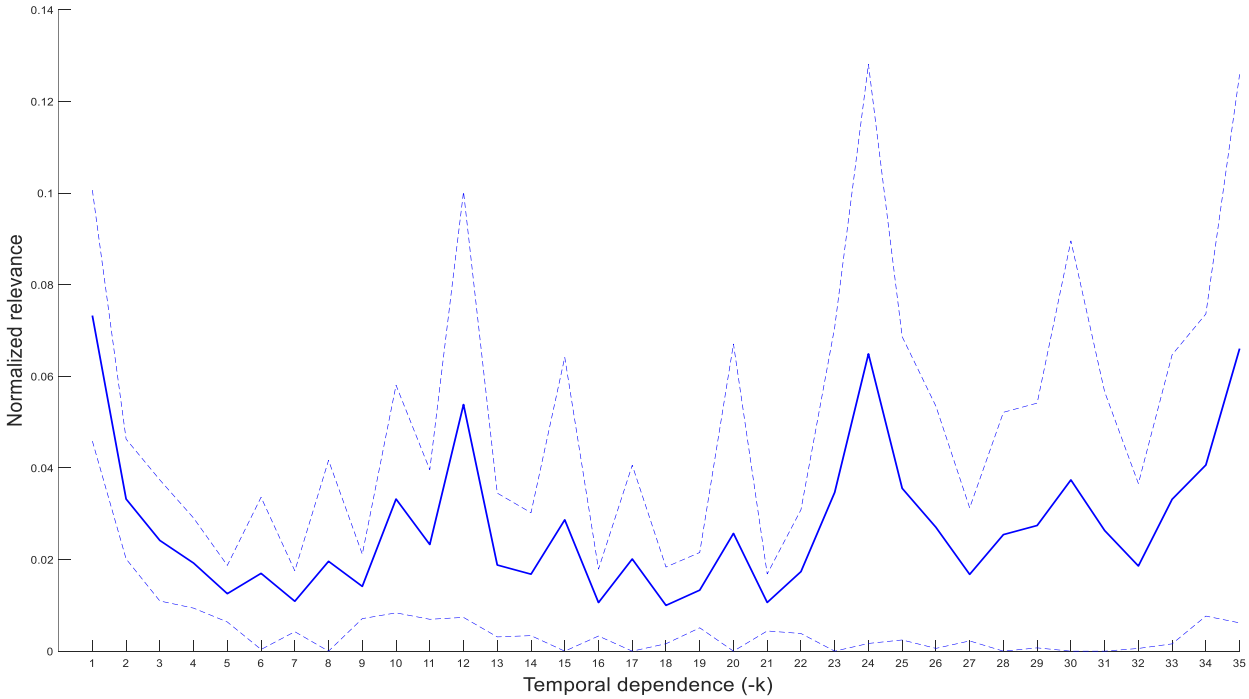


Fig. 5.3.20. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Copy-memory-d50 dataset.

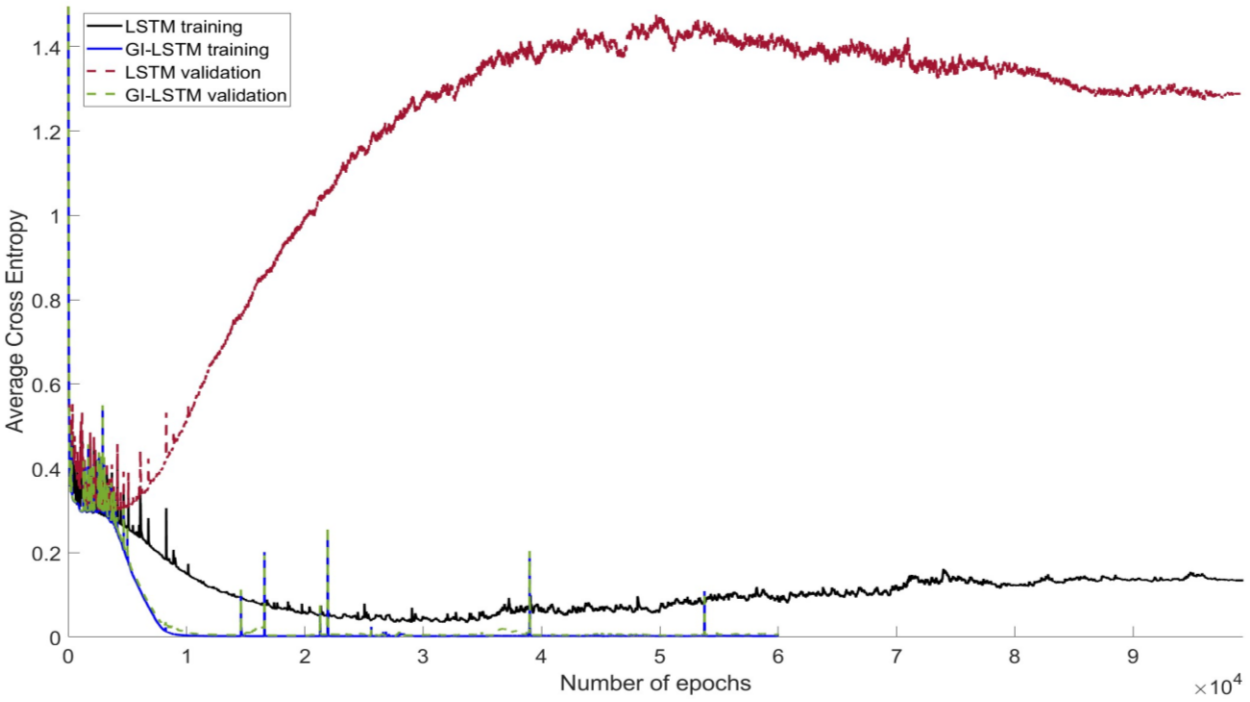


Fig. 5.3.21. Training and validation cross-entropy for the Copy-memory-d50 dataset.

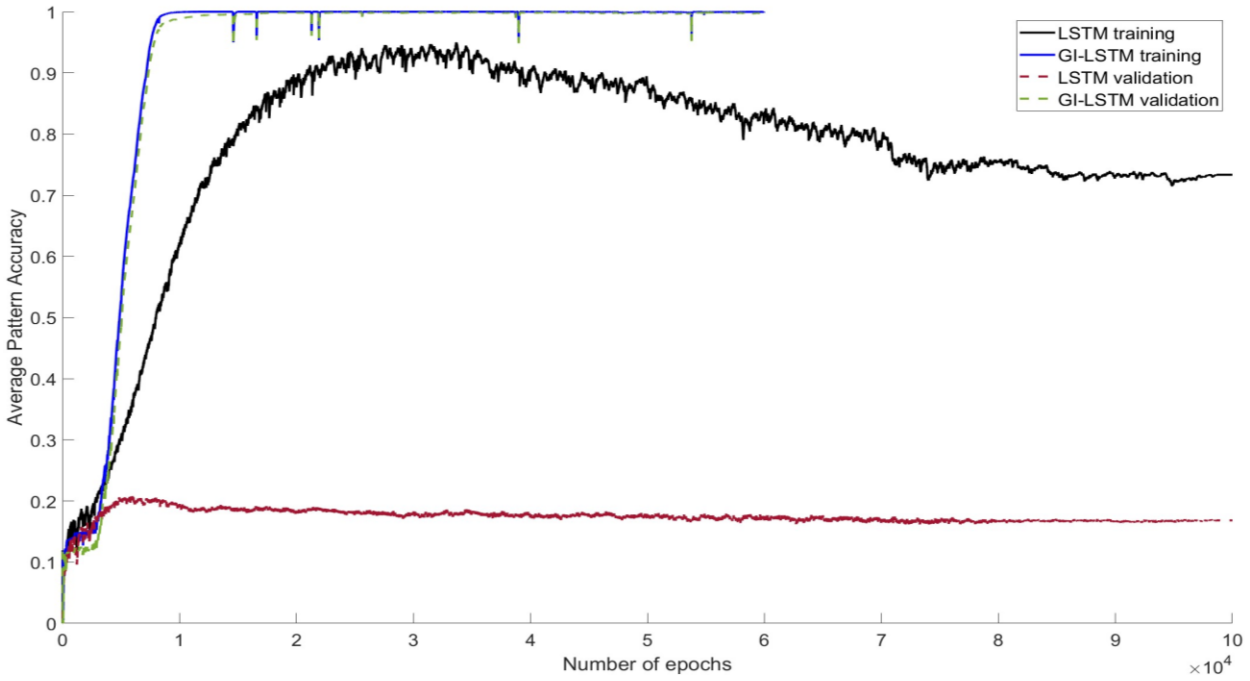


Fig. 5.3.22. Training and validation pattern accuracy for the Copy-memory-d50 dataset.

When the delay value is increased to 200 a more pronounced difference in relation to the generalization capability in the networks can be observed in Table 5.3.9; additional results for the GI-LSTM are provided to directly compare networks of similar sizes. In this case, the interpretability plot (Fig. 5.3.23) of the best performing GI-LSTM shows a high relevance associated to a long-term dependence at 105, revealing again that the network tries to propagate the pattern information occurring at the beginning of the sequence, located between 220 and 210 previous instances. Also, the training process in Fig. 5.3.24-Fig. 5.3.25 shows the LSTM is not able to learn the pattern in the training dataset, while the GI-LSTM starts to reduce its generalization performance in a noticeable but not critical fashion.

Table 5.3.9. Results for the Copy-memory-d200 dataset, 100 sequences and batch size of 100, with additional results for the GI-LSTM.

Copy-memory-d200												
		μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	n_{θ}	$\bar{t}_{iter}$(s)	$\bar{I}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
LSTM	Cross Entropy	0.0926	0.0020	0.0943	0.0011	0.0959	0.0040	16	1185	0.56	23167.15	3.60
	Total accuracy(%)	96.36	0.09	96.24	0.08	96.21	0.07					
	Pattern accuracy(%)	20.01	1.97	17.19	1.80	16.81	1.59					
GI-LSTM$_{110}$	Cross Entropy	0.0096	0.0137	0.0103	0.0140	0.0128	0.0184	8	1209	0.93	24138.7	6.24
	Total accuracy(%)	99.83	0.590	99.81	0.62	99.77	0.63					
	Pattern accuracy(%)	96.48	12.97	95.92	13.36	95.72	13.30					
	Cross Entropy	**0.0029**	**0.0066**	**0.0047**	**0.0093**	**0.0062**	**0.0116**	**16**	**2929**	**1.92**	**17467.9**	**9.3**
	Total accuracy(%)	**99.95**	**0.15**	**99.88**	**0.32**	**99.84**	**0.36**					
	Pattern accuracy(%)	**99.10**	**2.68**	**97.27**	**6.98**	**96.94**	**7.38**					

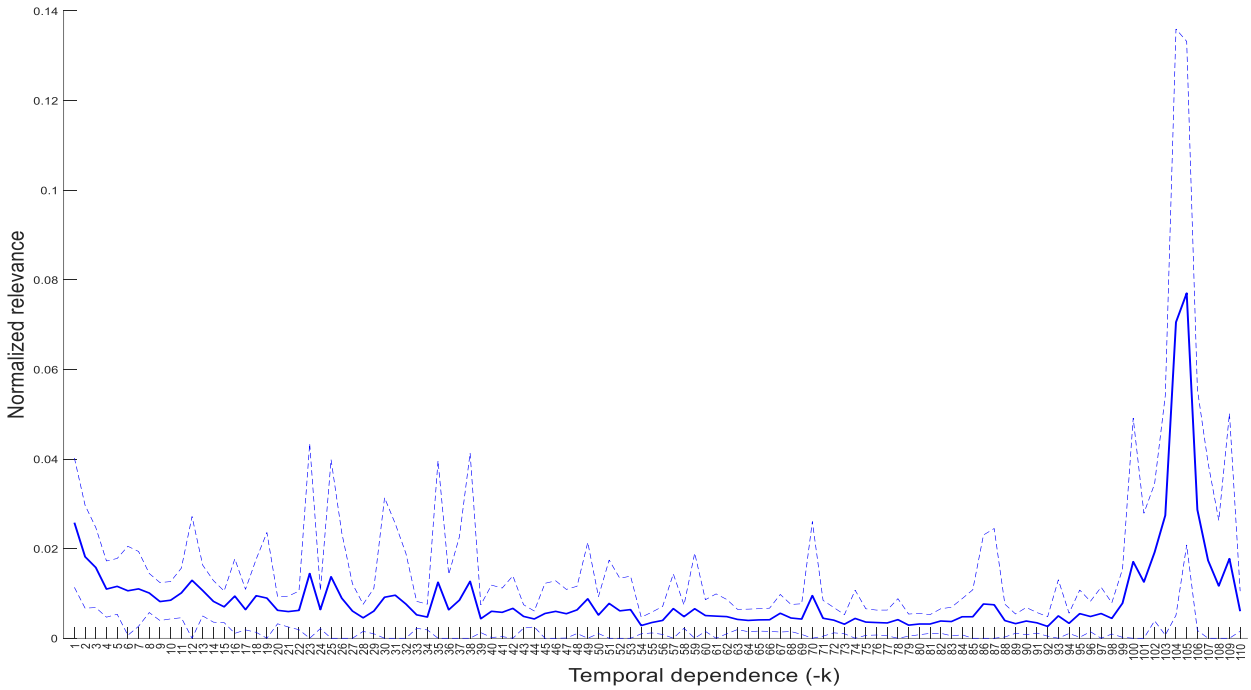


Fig. 5.3.23. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the copy-memory-d200 dataset.

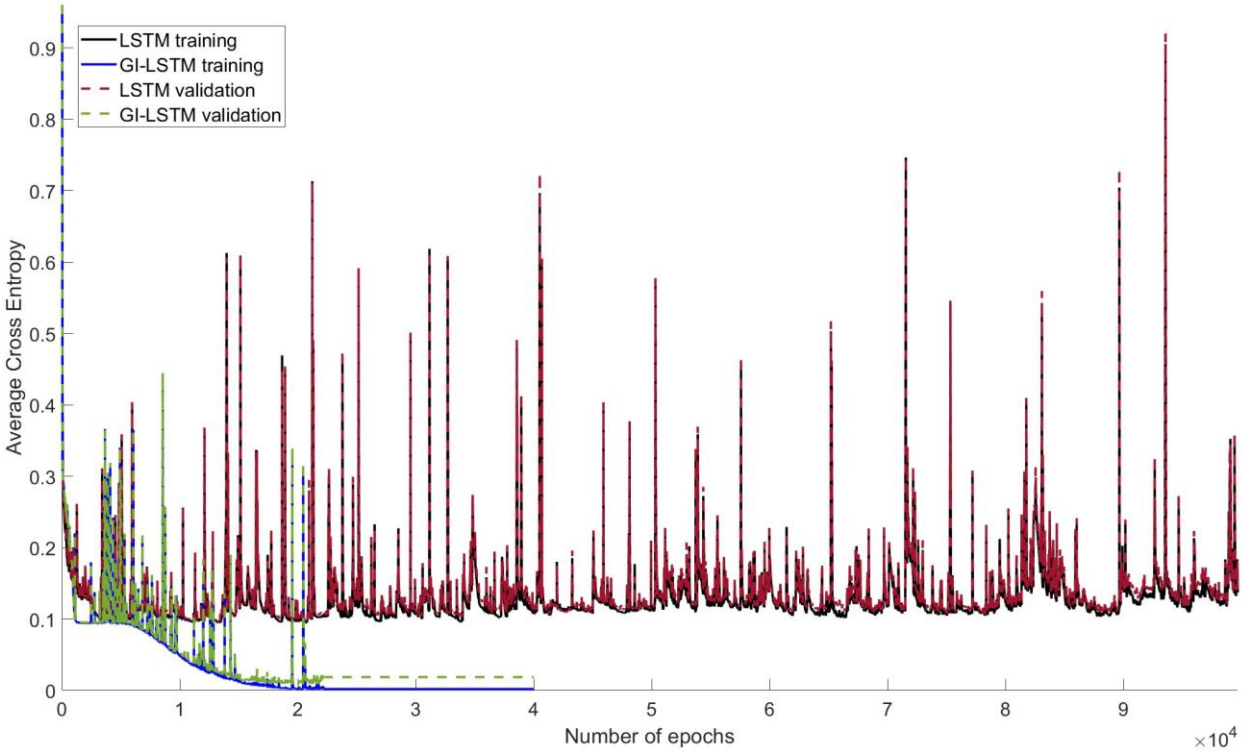


Fig. 5.3.24. Training and validation cross-entropy for the Copy-memory-d200 dataset.

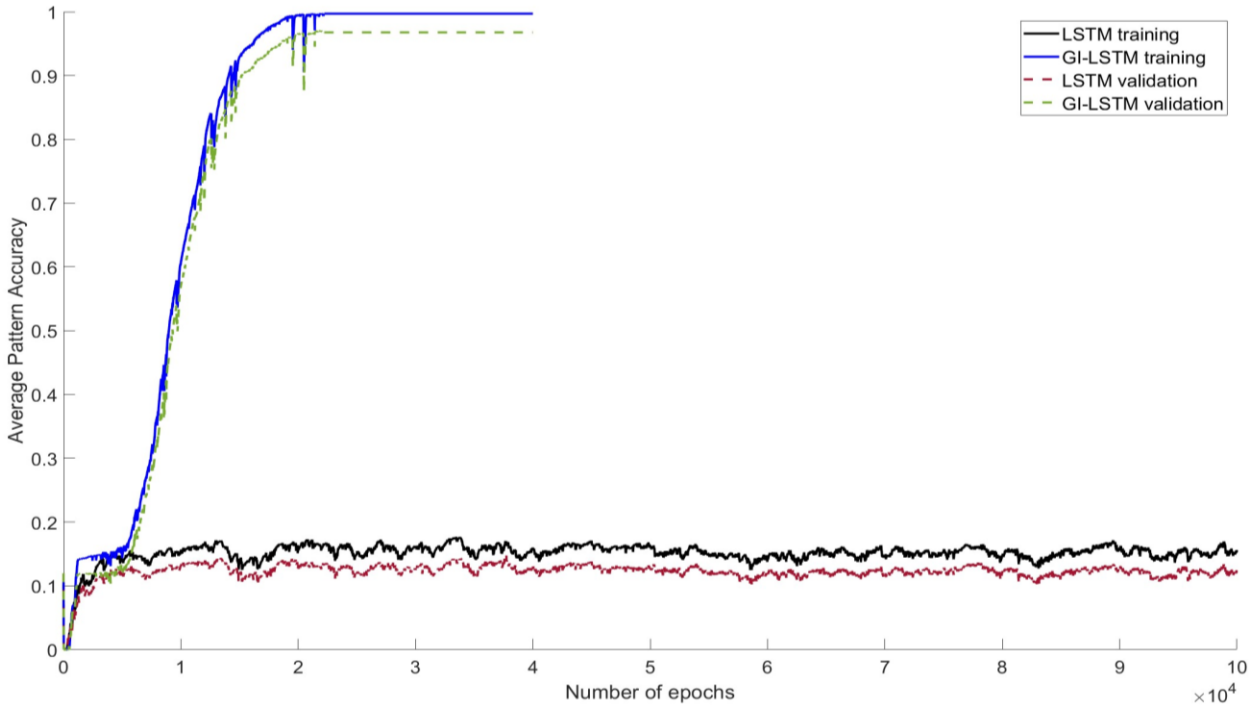


Fig. 5.3.25. Training and validation pattern accuracy for the Copy-memory-d200 dataset.

An experiment using delay of 400 is carried out (Table 5.3.10); however, the LSTM network is not used in this case, a decision made based on its low performance for easier tasks. For this scenario, the GI-LSTM generalization performance reduces, partly explained by the maximum simulation time used for this experiment. In addition, the associated interpretability plot (Fig. 5.3.26) for this experiment resembles that for a delay of 200, with a high relevance near the last dependencies. Furthermore, Fig. 5.3.27- Fig. 5.3.28 support the observation of the simulation time affecting the GI-LSTM performance, where the last figure shows an upward trend for the pattern accuracy.

Table 5.3.10. Results for the copy-memory-d400 dataset ($T_{delay} = 400$), 100 sequences and batch size of 100.

Copy-memory-d400												
		μ_{train}	σ_{train}	μ_{val}	σ_{val}	μ_{test}	σ_{test}	n_h	$n_{\boldsymbol{\theta}}$	$\bar{t}_{iter}$(s)	$\bar{l}_{iter}^{(opt)}$	$\bar{t}^{(opt)}$(h)
	Cross Entropy	0.0110	0.0167	0.0122	0.0175	0.0125	0.0173					
GI-LSTM$_{210}$	Total accuracy(%)	99.59	0.69	99.54	0.74	99.52	0.73	16	4529	3.37	21285.2	19.9
	Pattern accuracy(%)	82.82	29.97	80.53	31.11	80.28	30.97					

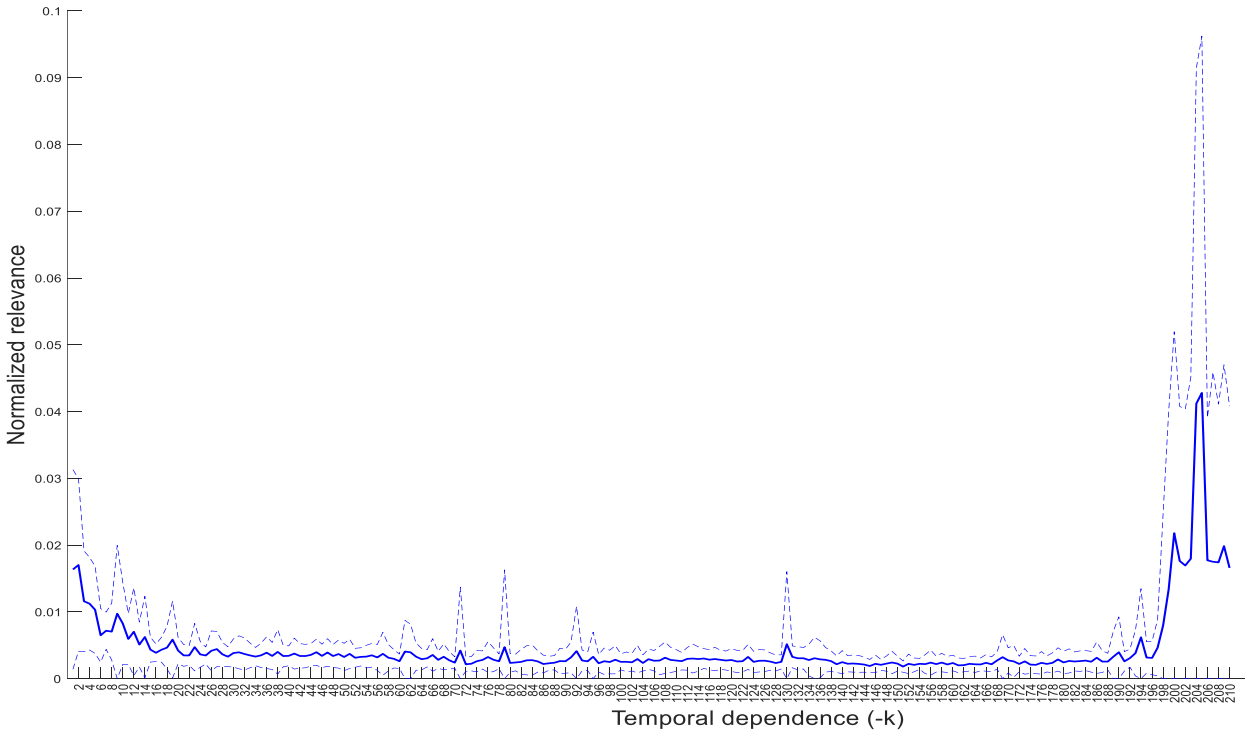


Fig. 5.3.26. Temporal-dependence relevance in the GI-LSTM memory-group 1 for the Copy-memory-d400 dataset.

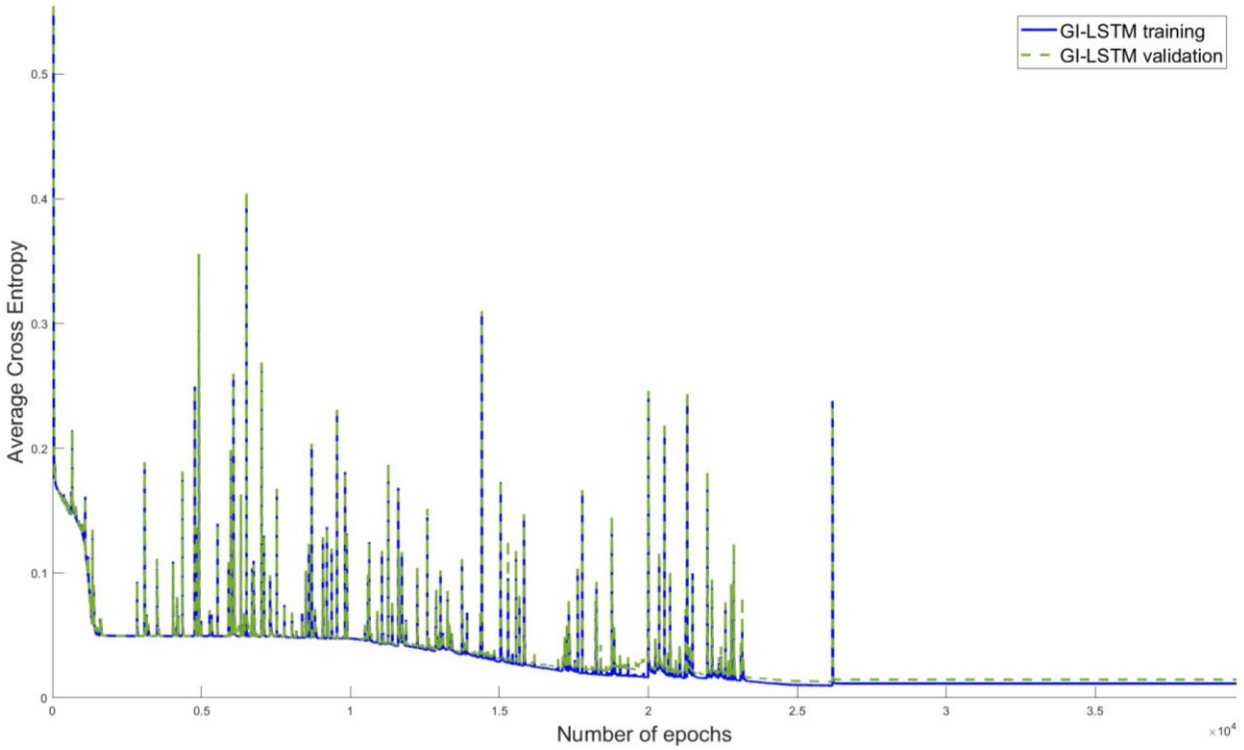


Fig. 5.3.27. Training and validation cross-entropy for the Copy-memory-d400 dataset.

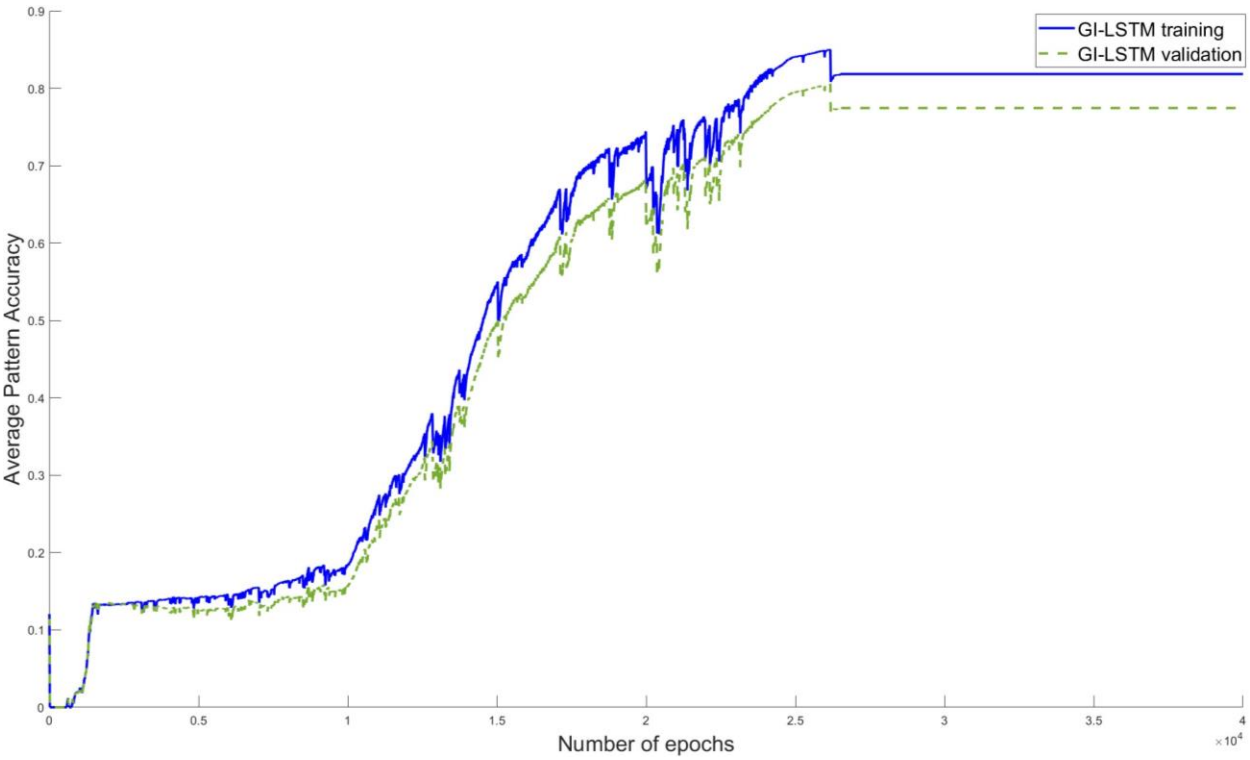


Fig. 5.3.28. Training and validation pattern accuracy for the Copy-memory-d400 dataset.

Chapter 6

Conclusion

In this work, machine learning models for time-series forecasting are proposed and experimentally tested. These models are intended to be interpretable, possess increasing long-term learning capability and maintain acceptable size. They exhibit promising performance across a variety of simulated and real-world experiments. These results were achieved by taking the following path:

- An inherently interpretable adaptive linear model for dynamic systems was proposed. The model was designed to incorporate information more actively through a time-varying forgetting factor that is constrained by physically interpretable and user-defined parameters.
- The adaptive linear model was used as the basis, together with a General Predictive Controller approach with a variable time horizon, to create an Adaptive Predictive Controller.
- The proposed Adaptive Predictive Controller was implemented on a real rack-mounted cooling system to control server temperatures in data centres, using a low-cost commercial microcontroller. The resulting model outperforms standard control algorithms (a standard GPC included), in both simulations and real-world tests.

- The proposed Adaptive Predictive Controller also showed capabilities to significantly reduce energy-consumption expenses by allowing for a monetary optimization algorithm.
- In order to consider nonlinear effects in time series, attention is redirected to the LSTM model. Initial steps to overcome the model's potential limitations on identifying long-term dependencies and using a large number of parameters are taken by increasing its internal temporal connectivity, resulting in the E-LSTM architecture.
- An approach based on the Distance Correlation intended to detect nonlinear effects that can be exploited by time-series nonlinear models is proposed. This approach is used to select the incremented temporal connectivity in the E-LSTM architecture.
- Experiments using the proposed E-LSTM, the LSTM and alternative time-series linear and nonlinear models showed that the E-LSTM achieved similar or better performance for a variety of synthetic and real-world time-series datasets, while in most cases maintaining or reducing the number of parameters.
- Further steps are taken to increase performance and interpretability by proposing a Generalized Interpretable LSTM (GI-LSTM) architecture, with even higher temporal connectivity than the E-LSTM, allowing for semi-global interpretation and removing the need for precisely locating the temporal connectivity.
- Experiments are carried out with the proposed GI-LSTM and alternative linear and nonlinear models, showing the proposed GI-LSTM provides better performance with respect to size while becoming more accessible for human interpretation.

From the previous milestones some insights can be extracted. First, the ability of an adaptive linear model to dynamically regulate how much relevance is given to new information,

based on interpretable physical constraints, not only promotes better performance and reliability, but also showcases the usefulness of simple and interpretable models for industrial applications.

Among the potential restrictions of the proposed adaptive linear model is the limited memory capability, which can force the model to forget useful information to adapt to the current dynamic system's state. In this regard, investigation in the direction of piece-wise linear models could be performed, from the perspective of having a set of learnable parameters (vectors) functioning as not necessarily disjoint 'memories' for different system states. In this way, relevant information is more likely to remain encoded and dynamic adaptation can be used mostly to handle time-varying conditions in the system and not to handle nonlinear effects.

In relation to the E-LSTM, the experimental results showed that, for time-series datasets in which nonlinear effects might be present, the standard LSTM architecture seems to rely on a larger number of hidden units and forget gates to identify long-term dependencies, consequently producing a large number of parameters in the model. In contrast, the extended connectivity in the E-LSTM alleviates this need while improving the performance in some cases.

The E-LSTM model has two limitations. It relies on an external approach to identify where the extended connectivity location should be created and it is not clear that such an approach can be used for more than a single-layer architecture while producing significantly better results. The first of these limitations is mostly addressed by the GI-LSTM, as shown in the last part of this thesis. In relation to the second limitation, investigation on incremental training of a multi-layer E-LSTM could be performed; specifically, selecting the temporal connectivity of a second E-LSTM layer based on the residual errors created by the first layer, and iteratively repeating this process for subsequent layers.

The proposed GI-LSTM network and the experimental results suggested that not only is it possible to further increase the internal connectivity in the E-LSTM, removing the need for preprocessing the data by means of an external algorithm, but also showed that a small network can still result in a competitive model. Also, the proposed architecture opens up the possibility of having the previous advantages while allowing for interpretability. Furthermore, the architecture is general enough to be used in deep layer architectures, an approach that would be worth testing for additional time-series showing more complex nonlinearities.

There are a number of research opportunities with respect to the GI-LSTM architecture, aimed to increase performance and interpretability, accelerate the training process and produce smaller network sizes. First, artificial stochastic variability could be produced in the memory-group parameters, since they represent the core mechanism in the identification of long-term dependencies. This could decrease the time spent in local optima and possibly reduce the number of units needed to identify relevant dependencies. Also, modifications to the input gate in the direction expressed by [119] could be performed if deep layers were to be investigated, since the memory-group strategy is compatible with such modifications. In relation to the temporal connectivity in the memory groups, it is worth noting that such connectivity could be further promoted by designing mechanisms that allow for information sharing across memory-group units, since the proposed temporal connectivity is limited to be unit-wise, i.e., the behavior of each element in a memory-group is not directly influenced by other elements' behavior. This lack of influence could lead to undesired, or at least not well-directed, redundancy; producing model sizes larger than necessary.

Bibliography

[1] Magnus, J. R., & Neudecker, H. (2019). *Matrix Differential Calculus with Applications in Statistics and Econometrics (3rd ed.)*. John Wiley & Sons. ISBN: 1119541204.

[2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

[3] Miotto, R., Wang, F., Wang, S., Jiang, X., & Dudley, J. T. (2018). Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, *19*(6), 1236-1246.

[4] Raghu, M., & Schmidt, E. (2020). A survey of deep learning for scientific discovery. *arXiv preprint arXiv:2003.11755*.

[5] Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181-1191.

[6] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[7] Marcus, G. (2020). The next decade in AI: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177.*

[8] Samek, W., & Müller, K. R. (2019). Towards explainable artificial intelligence. *Explainable AI: interpreting, explaining and visualizing deep learning*, 5-22.

[9] Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: visualising image classification models and saliency maps. *Proceedings of the International Conference on Learning Representations (ICLR)*, 1–8.

[10] Holzinger, A., Biemann, C., Pattichis, C. S., & Kell, D. B. (2017). What do we need to build explainable AI systems for the medical domain?. *arXiv preprint arXiv:1712.09923*.

[11] El-Sherief, H., & Sinha, N. (1979). Choice of models for the identification of linear multivariable discrete-time systems. *Proceedings of the Institution of Electrical Engineers*, *126*(12), 1326.

[12] Ding, F., & Chen, T. (2005). Hierarchical least squares identification methods for multivariable systems. *IEEE Transactions on Automatic Control*, *50*(3), 397-402.

[13] Li, J., Stoica, P., Xu, L., & Roberts, W. (2007). On parameter identifiability of MIMO radar. *IEEE signal processing letters*, *14*(12), 968-971.

[14] Ozsoy, C., Kural, A., Cetinkaya, M., & Ertug, S. (1999, October). Constrained MIMO self-tuning composition control in cement industry. In *1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA'99 (Cat. No. 99TH8467)* (Vol. 2, pp. 1021-1028). IEEE.

[15] Han, J., Cui, Q., Guo, L., Rehman, W. U., & Chen, Z. (2013, April). Application of orthogonal experimental design to MIMO detection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 4009-4014). IEEE.

[16] Phung, J., Young, C. L., & Zomaya, A. Y. (2017, May). Application-agnostic power monitoring in virtualized environments. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (pp. 335-344). IEEE

[17] Rodionov, R., De Martino, F., Laufs, H., Carmichael, D. W., Formisano, E., Walker, M., & Lemieux, L. (2007). Independent component analysis of interictal fMRI in focal epilepsy: comparison with general linear model-based EEG-correlated fMRI. *Neuroimage*, *38*(3), 488-500.

[18] Kollias, S., & Anastassiou, D. (1989). An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Transactions on Circuits and Systems*, *36*(8), 1092-1101.

[19] Dayal, B. S., & Macgregor, J. F. (1997). Recursive exponentially weighted PLS and its applications to adaptive control and prediction. *Journal of Process Control, 7*(3), 169-179.

[20] Chen, S., Billings, S. A., & Grant, P. M. (1992). Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *International Journal of Control, 55*(5), 1051-1070.

[21] Mao, Y., Ding, F., Xu, L., & Hayat, T. (2019). Highly efficient parameter estimation algorithms for Hammerstein non-linear systems. *IET Control Theory & Applications*, *13*(4), 477-485.

[22] Xu, L., & Ding, F. (2017). Recursive least squares and multi-innovation stochastic gradient parameter estimation methods for signal modeling. *Circuits, Systems, and Signal Processing*, *36*, 1735-1753.

[23] Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems, 13*.

[24] He, W., Chen, Y., & Yin, Z. (2015). Adaptive neural network control of an uncertain robot with full-state constraints. *IEEE transactions on cybernetics, 46*(3), 620-629.

[25] Liang, D., Krishnan, R. G., Hoffman, M. D., & Jebara, T. (2018, April). Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference* (pp. 689-698).

[26] Geva, M., Schuster, R., Berant, J., & Levy, O. (2021). Transformer Feed-Forward Layers Are Key-Value Memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[27] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[28] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation*, *12*(10), 2451-2471.

[29] Gers, F. (2001). Long short-term memory in recurrent neural networks. *PhD dissertation, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*.

[30] Athiwaratkun, B., & Stokes, J. W. (2017, March). Malware classification with LSTM and GRU language models and a character-level CNN. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 2482-2486). IEEE.

[31] Fu, R., Zhang, Z., & Li, L. (2016, November). Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth academic annual conference of Chinese association of automation (YAC)* (pp. 324-328). IEEE.

[32] Gensler, A., Henze, J., Sick, B., & Raabe, N. (2016, October). Deep Learning for solar power forecasting—An approach using AutoEncoder and LSTM Neural Networks. In *2016 IEEE international conference on systems, man, and cybernetics (SMC)* (pp. 2858-2865). IEEE.

[33] Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive business process monitoring with LSTM neural networks. In *Advanced Information Systems Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings 29* (pp. 477-492). Springer International Publishing.

[34] Wigington, C., Stewart, S., Davis, B., Barrett, B., Price, B., & Cohen, S. (2017, November). Data augmentation for recognition of handwritten words and lines using a CNN-LSTM network. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)* (Vol. 1, pp. 639-645). IEEE.

[35] Woo, W. W., Svoronos, S. A., & Crisalle, O. D. (1995, June). A directional forgetting factor for single-parameter variations. In *Proceedings of 1995 American Control Conference-ACC'95* (Vol. 2, pp. 1149-1151). IEEE.

[36] Rao, A. K., Huang, Y. F., & Dasgupta, S. (1990). ARMA parameter estimation using a novel recursive estimation algorithm with selective updating. *IEEE transactions on acoustics, speech, and signal processing*, *38*(3), 447-457.

[37] Bittanti, S., Bolzern, P., & Campi, M. (1990). Convergence and exponential convergence of identification algorithms with directional forgetting factor. *Automatica*, *26*(5), 929-932.

[38] Ydstie, B. E., Kershenbaum, L. S., & Sargent, R. W. H. (1985). Theory and application of an extended horizon self-tuning regulator. *AIChE, 31*(11), 1771-1780.

[39] Zhang, Y., Tiňo, P., Leonardis, A., & Tang, K. (2021). A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *5*(5), 726-742.

[40] Martinez-Garcia, F., Badawy, G., Kheradmandi, M., & Down, D. G. (2021). Adaptive Predictive Control of a data center cooling unit. *Control Engineering Practice*, *107*, 104674.

[41] Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming, 45*(1-3), 503-528.

[42] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics, 11*(2), 431-441.

[43] Kinga, D., & Adam, J. B. (2015, May). A method for stochastic optimization. In *International conference on learning representations (ICLR)* (Vol. 5, p. 6).

[44] Deng, B. C., Yun, Y. H., Liang, Y. Z., Cao, D. S., Xu, Q. S., Yi, L. Z., & Huang, X. (2015). A new strategy to prevent over-fitting in partial least squares models based on model population analysis. *Analytica Chimica Acta, 880*, 32-41.

[45] Mangan, N. M., Kutz, J. N., Brunton, S. L., & Proctor, J. L. (2017). Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473*(2204), 20170009.

[46] Clarke, D. W., Mohtadi, C., & Tuffs, P. S. (1987). Generalized predictive control—Part I. The basic algorithm. *Automatica, 23*(2), 137-148.

[47] Clarke, D. W., Mohtadi, C., & Tuffs, P. S. (1987). Generalized predictive control—Part II Extensions and interpretations. *Automatica, 23*(2), 149-160.

[48] Shekhar, R. C., & Maciejowski, J. M. (2012). Robust variable horizon MPC with move blocking. *Systems & Control Letters*, *61*(4), 587-594.

[49] Keerthi, S. S., & Gilbert, E. G. (1988). Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. *Journal of optimization theory and applications*, *57*, 265-293.

[50] Hedjar, R. (2013). Adaptive neural network model predictive control. *International Journal of Innovative Computing, Information and Control*, *9*(3), 1245-1257.

[51] Bobal, V., Kubalcik, M., Dostal, P., & Matejicek, J. (2013). Adaptive predictive control of time-delay systems. *Computers & Mathematics with Applications*, *66*(2), 165-176.

[52] Mizumoto, I., Fujimoto, Y., & Ikejiri, M. (2015). Adaptive output predictor based adaptive predictive control with ASPR constraint. *Automatica*, *57*, 152-163.

[53] Yoon, T. W., & Clarke, D. W. (1994). Adaptive predictive control of the benchmark plant. *Automatica*, *30*(4), 621-628.

[54] Ydstie, B. E., Kershenbaum, L. S., & Sargent, R. W. H. (1985). Theory and application of an extended horizon self-tuning regulator. *AIChE, 31*(11), 1771-1780.

[55] Lara, C., Flores, J. J., & Calderon, F. (2009). On the Hyperbox – Hyperplane Intersection Problem. *INFOCOMP, 8*(4), 21-27.

[56] Woodbury, M.A. & Princeton University. Department of Statistics. (1950). *Inverting Modified Matrices*. Department of Statistics, Princeton University.

[57] Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, *9*(2011), 161.

[58] Delforge, P., Whitney, J., & Anthesis. (2014). Data Center Efficiency Assessment: Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers (IP:14-08-a). NRDC.

[59] Patterson, M. K. (2008, May). The effect of data center temperature on energy efficiency. In *2008 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems* (pp. 1167-1174). IEEE.

[60] Dunlap, K., & Rasmussen, N. (2012). Choosing Between Room, Row, and Rack-based Cooling for Data Centers. Schneider Electr.

[61] El-Sayed, N., Stefanovici, I. A., Amvrosiadis, G., Hwang, A. A., & Schroeder, B. (2012, June). Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems* (pp. 163-174).

[62] Afram, A., & Janabi-Sharifi, F. (2014). Theory and applications of HVAC control systems–A review of model predictive control (MPC). *Building and Environment*, *72*, 343-355.

[63] Campi, M. (1994). Performance of RLS Identification Algorithms with Forgetting Factor: A ϕ-Mixing Approach. Journal of Mathematical Systems, Estimation, and Control, 4(3), 1-25.

[64] Lipták, B. G. (1985). Control and on-off valves (pp. 410-412). Instrument Engineers' Handbook, 2nd ed.

[65] Reyes-Lúa, A., Zotică, C., Forsman, K., & Skogestad, S. (2019). Systematic design of split range controllers. *IFAC-PapersOnLine*, *52*(1), 898-903.

[66] Ontario Energy Board. (n.d.). *Electricity Rates.* Retrieved July 10, 2018 from https://www.oeb.ca/rates-and-your-bill/electricity-rates.

[67] City of Toronto. (n.d.). *2018 Water Rates & Fees.* Retrieved July 10, 2018 from https://www.toronto.ca/services-payments/property-taxes-utilities/utility-bill/water-rates-and-fees/.

[68] Teran, R., Draye, J. P., Pavisic, D., Calderon, G., & Libert, G. (1996, January). Predicting a chaotic time series using a dynamical recurrent neural network. In *Proceedings IWISP'96* (pp. 115-118). Elsevier Science Ltd.

[69] Seidl, D. R., & Lorenz, R. D. (1991, July). A structure by which a recurrent neural network can approximate a nonlinear dynamic system. In *IJCNN-91-Seattle International Joint Conference on Neural Networks* (Vol. 2, pp. 709-714). IEEE.

[70] Schuster, M. (1996, September). Learning out of time series with an extended recurrent neural network. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop* (pp. 170-179). IEEE.

[71] Robinson, A., and F. Fallside. "Static and dynamic error propagation networks with application to speech coding." *Neural information processing systems*. 1987.

[72] Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE transactions on Neural Networks*, *5*(2), 298-305.

[73] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550-1560.

[74] Rao, D. H., & Gupta, M. M. (1994). Neuro-fuzzy controller for control and robotics applications. *Engineering Applications of Artificial Intelligence*, *7*(5), 479-491.

[75] Tsoi, A. C., Shrimpton, D., Watson, B., & Back, A. (1994). Application of artificial neural network techniques to speaker verification. In *Automatic Speaker Recognition, Identification and Verification*.

[76] Alvarez-Cercadillo, J., Ortega-Garcia, J., & Hernández-Gómez, L. A. (1993, April). Context modeling using RNN for keyword detection. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing* (Vol. 1, pp. 569-572). IEEE.

[77] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen [M.S. thesis]. *Technische Universitt München, München, Germany*.

[78] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.

[79] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.

[80] Williams, R. J., & Zipser, D. (1990). *Gradient-based learning algorithms for recurrent connectionist networks* (pp. 433-486). Boston, MA: College of Computer Science, Northeastern University.

[81] Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*.

[82] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, *18*(5-6), 602-610.

[83] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*

[84] Li, C., Yang, Y., Feng, M., Chakradhar, S., & Zhou, H. (2016, November). Optimizing memory efficiency for deep convolutional neural networks on GPUs. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 633-644). IEEE.

[85 Hwang, K., & Sung, W. (2015, April). Single stream parallelization of generalized LSTM-like RNNs on a GPU. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1047-1051). IEEE.

[86] Kuchaiev, O., & Ginsburg, B. (2017). Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*.

[87] Sussmann, H. J. (1992). Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural networks*, *5*(4), 589-593.

[88] Chen, A. M., Lu, H. M., & Hecht-Nielsen, R. (1993). On the geometry of feedforward neural network error surfaces. *Neural computation*, *5*(6), 910-927

[89] Martinez-Garcia, F., & Down, D. (2022, July). E-LSTM: An extension to the LSTM architecture for incorporating long lag dependencies. In *2022 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.

[90] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017, June). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture* (pp. 1-12)Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017, June). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture* (pp. 1-12)

[91] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222-2232.

[92] Ganesh, P., & Rakheja, P. (2018). Vlstm: Very long short-term memory networks for high-frequency trading. *arXiv preprint arXiv:1809.01506*.

[93] Xie, J., Yan, R., Xiao, S., Peng, L., Johnson, M. T., & Zhang, W. Q. (2020, May). Dynamic temporal residual learning for speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 7709-7713). IEEE.

[94] Shen, L., Yu, Z., Ma, Q., & Kwok, J. T. (2021, May). Time series anomaly detection with multiresolution ensemble decoding. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 11, pp. 9567-9575).

[95] Kieu, T., Yang, B., Guo, C., & Jensen, C. S. (2019, August). Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *IJCAI* (pp. 2725-2732).

[96] Pigou, L., Van Herreweghe, M., & Dambre, J. (2017). Gesture and sign language recognition with temporal residual networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 3086-3093).

[97] Zhang, J., Zheng, Y., & Qi, D. (2017, February). Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 31, No. 1).

[98] Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014, June). A clockwork rnn. In *International conference on machine learning* (pp. 1863-1871). PMLR.

[99] Achanta, S., Godambe, T., & Gangashetty, S. V. (2015). An investigation of recurrent neural network architectures for statistical parametric speech synthesis. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[100] Feng, Xiong, et al. "State-of-charge estimation of lithium-ion battery based on clockwork recurrent neural network." *Energy* 236 (2021): 121360.

[101] Musbah, H., & El-Hawary, M. (2019, May). SARIMA model forecasting of short-term electrical load data augmented by fast fourier transform seasonality detection. In *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)* (pp. 1-4). IEEE.

[102] Samal, K. K. R., Babu, K. S., Das, S. K., & Acharaya, A. (2019, August). Time series based air pollution forecasting using SARIMA and prophet model. In *proceedings of the 2019 international conference on information technology and computer communications* (pp. 80-85).

[103] Hsu, H. H., Hsieh, C. W., & Lu, M. D. (2011). Hybrid feature selection by combining filters and wrappers. *Expert Systems with Applications*, *38*(7), 8144-8150.

[104] Mafarja, M., & Mirjalili, S. (2018). Whale optimization approaches for wrapper feature selection. *Applied Soft Computing*, *62*, 441-453.

[105] Porkodi, R. (2014). Comparison of filter based feature selection algorithms: An overview. *International journal of Innovative Research in Technology & Science*, *2*(2), 108-113.

[106] Dueck, J., Edelmann, D., Gneiting, T., & Richards, D. (2014). The affinely invariant distance correlation. *Bernoulli*, *20*(4), 2305-2330.

[107] Székely, G. J., Rizzo, M. L., & Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances.

[108] Székely, G. J., & Rizzo, M. L. (2013). The distance correlation t-test of independence in high dimension. *Journal of Multivariate Analysis*, *117*, 193-213.

[109] Silva, E. S., Hassani, H., Heravi, S., & Huang, X. (2019). Forecasting tourism demand with denoised neural networks. *Annals of Tourism Research*, *74*, 134-154.

[110] Yakovyna, V., Uhrynovskyi, B., & Bachkay, O. (2019, September). Software failures forecasting by Holt-Winters, ARIMA and NNAR methods. In *2019 IEEE 14th International*

Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 2, pp. 151-155). IEEE.

[111] London, W. P., & Yorke, J. A. (1973). Recurrent outbreaks of measles, chickenpox and mumps: I. Seasonal variation in contact rates. *American journal of epidemiology*, *98*(6), 453-468.

[112] Royal Observatory of Belgium. (n.d). *World Data Center SILSO*. Retrieved November 5, 2020 from http://www.sidc.be/silso/datafiles.

[113] Government of Canada. (n.d). *The official website of the Government of Canada*. Retrieved December 20, 2020 from https://climate.weather.gc.ca/historical-data/search-historic-data-e.html/.

[114] PJM Interconnection LLC. (n.d). *kaggle*. Retrieved Aug. 10, 2020 from https://www.kaggle.com/robikscube/hourly-energy-consumption/.

[115] Reddi, S. J., Kale, S., & Kumar, S. (2018, February). On the Convergence of Adam and beyond. In *International Conference on Learning Representations*.

[116] Yazan, E., & Talu, M. F. (2017, September). Comparison of the stochastic gradient descent based optimization techniques. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)* (pp. 1-5). IEEE.

[117] Kanuparthi, B., Arpit, D., Kerg, G., Ke, N. R., Mitliagkas, I., & Bengio, Y. (2018, September). h-detach: Modifying the LSTM gradient towards better optimization. In *International Conference on Learning Representations*.

[118] Rusch, T. K., & Mishra, S. (2021, July). Unicornn: A recurrent model for learning very long time dependencies. In *International Conference on Machine Learning* (pp. 9168-9178). PMLR.

[119] Turkoglu, M. O., D'Aronco, S., Wegner, J. D., & Schindler, K. (2021). Gating revisited: Deep multi-layer RNNs that can be trained. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(8), 4081-4092.

[120] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., & Ré, C. (2021). Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, *34*, 572-585.

[121] Arjovsky, M., Shah, A., & Bengio, Y. (2016, June). Unitary evolution recurrent neural networks. In *International conference on machine learning* (pp. 1120-1128). PMLR.

[122] Tallec, C., & Ollivier, Y. (2018, April). Can recurrent neural networks warp time?. In *International Conference on Learning Representation 2018*.

[123] Henaff, M., Szlam, A., & LeCun, Y. (2016, June). Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning* (pp. 2034-2042). PMLR.

[124] Landi, F., Baraldi, L., Cornia, M., & Cucchiara, R. (2021). Working memory connections for LSTM. *Neural Networks*, *144*, 334-341.