PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

DEEP LEARNING METHODS FOR INTELLIGENT CYBER-PHYSICAL SYSTEMS

SAÚL ALBERTO LANGARICA CHAVIRA

Thesis submitted to the Office of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor of Science in Engineering

Advisor:
FELIPE NÚÑEZ

Santiago de Chile, March 2023

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

DEEP LEARNING METHODS FOR INTELLIGENT CYBER-PHYSICAL SYSTEMS

SAÚL ALBERTO LANGARICA CHAVIRA

Members of the Committee:

FELIPE NÚÑEZ

ALDO CIPRIANO

HANS LÖBEL

MARCOS ORCHARD

ELEONORA AIELLO

IGNACIO LIRA

Thesis submitted to the Office of Graduate Studies in partial fulfillment of the requirements for the Degree Doctor in Engineering Sciences

Santiago de Chile, March, 2023

Gratefully to God

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God because He is the one who has made all this possible. He has been the one who has blessed me with the family I was born into, the one who has guided and supported me in every decision I have made, as the decision to continue doing what I love and continue my postgraduate studies. And He is the one who has always provided me with all the means and health to be able to accomplish this important goal of my life.

To my beautiful wife who has given me unconditional support in every decision, has been with me in everything and motivates me to always move forward to continue building our family.

To my parents for all the support given in this long process, for never letting me give up and always encouraging me to give my best and continue in spite of everything that might be happening around us.

To my supervisor Felipe Núñez for all his support, teachings, advice, patience and guidance throughout this arduous research process, and especially for encouraging me to always strive for excellence.

To Frank J. Doyle III for receiving me at Harvard University for my international internship and letting me work with him and other amazing researchers in his lab, from whom I learned a lot. This incredible experience opened my mind to new future research directions and helped me to conclude this thesis with the paper we wrote together.

Finally, I would like to thank to the Agencia Nacional de Investigación y Desarrollo (ANID) that through their many grants have made this work possible. Thanks to the grant ANID/PFCHA/Doctorado Nacional/2020-21200565 I was able to pay for my costs of living during this research, and thanks to the project ANID PIA ACT192013, I had everything I needed to continue with my research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cyber-physical systems (CPSs) have emerged in recent years as a new paradigm that merges several technologies to allow the interface between the physical and the cybernetic world. This has opened the door to the use of artificial intelligence (AI) techniques to interact with the physical world in real time. However, interaction with real physical systems imposes a number of challenges that were absent in the application domains for which these data-driven techniques were originally designed, and the creation of a new set of models and methods specifically designed to address these difficulties becomes necessary. This has limited a wider adoption of AI methods in CPSs and instead, many have opted for classical filtering, modeling and control methods for the development of new CPSs.

In this thesis it is shown how CPSs can be highly benefited at different levels of their architecture by the incorporation of intelligent data-driven methods, in particular deep learning methods, when they are carefully designed to deal with the inherent difficulties imposed by the interaction with real physical systems. In particular, it is shown that deep learning methods can endow CPSs with new capabilities that cannot be achieved with classical techniques, and that these AI methods have better performance than their classical counterparts in different domains, when using several application-specific metrics. Thus, showing how deep learning techniques contribute to unleash the full potential of CPSs, transforming them into better performing, more efficient and intelligent systems, namely "Intelligent" CPSs or iCPSs.

To this end, three methods for iCPSs, materialized in three different applications, namely, a general purpose learning-based data-cleaning method, an adaptative deep-learning model based on meta-learning, and a neuroevolutive controller with learning capabilities and great flexibility during the optimization process, are proposed.

Results show superiority of the proposed methods when compared to other data-driven and classical methods across the different domains our techniques are applied. It is expected that the results of this research will encourage the development of more suitable intelligent data-driven methods for iCPSs.

Keywords: Cyber-Physical Systems, Deep Learning, Contrastive Learning, Meta-Learning, Recurrent Kalman Networks, Neuroevolution, Reinforcement learning, Glucose prediction, Artificial Pancreas, Model Predictive Control

RESUMEN

Los sistemas ciberfísicos (CPSs) han surgido en los últimos años como un nuevo paradigma que fusiona varias tecnologías para permitir la interfaz entre el mundo físico y el cibernético. Esto ha abierto la puerta al uso de técnicas de inteligencia artificial (AI) para interactuar con el mundo físico en tiempo real. Sin embargo, la interacción con sistemas físicos reales impone una serie de desafíos que estaban ausentes en los dominios de aplicación para los que se diseñaron originalmente estas técnicas basadas en datos, y se hace necesario la creación de un nuevo conjunto de modelos y métodos específicamente diseñados para hacer frente a estas dificultades. Esto ha limitado una adopción más amplia de los métodos de AI en los CPSs y en cambio, muchos han optado por métodos clásicos de filtrado, modelado y control para el desarrollo de este tipo de sistemas.

En esta tesis se muestra cómo los CPSs pueden verse altamente beneficiados en diferentes niveles de su arquitectura al incorporar métodos inteligentes basados en datos, en particular, métodos de aprendizaje profundo cuidadosamente diseñados para lidiar con las dificultades inherentes que impone la interacción con sistemas físicos reales. En concreto, se muestra cómo los métodos de aprendizaje profundo pueden dotar a los CPSs de nuevas capacidades que no se pueden lograr con técnicas clásicas, y que estos métodos de AI logran un mejor rendimiento que sus homólogos clásicos en diferentes dominios cuando se utilizan diversas métricas específicas de la aplicación. Por lo tanto, mostrando cómo las técnicas de aprendizaje profundo contribuyen a liberar todo el potencial de los CPSs, transformándolos en sistemas de mejor rendimiento, más eficaces e inteligentes, a saber, en CPS “inteligentes” o iCPS.

Para ello, se proponen tres métodos para iCPSs, materializados en tres aplicaciones diferentes, a saber, un método de limpieza de datos basado en aprendizaje por contraste, un modelo de aprendizaje profundo adaptativo basado en meta-aprendizaje, y un controlador

neuroevolutivo con capacidad de aprendizaje y gran flexibilidad durante el proceso de optimización.

Los resultados muestran la superioridad de los métodos propuestos cuando se comparan con otros métodos clásicos y otros métodos basados en datos en los diferentes dominios en los que se aplican nuestros métodos. Se espera que los resultados de esta investigación fomenten el desarrollo de más métodos inteligentes adecuados para los iCPS.

Palabras Claves: Sistemas Ciberfísicos, Aprendizaje profundo, Inteligencia Artificial, Aprendizaje por Constraste, Meta-Aprendizaje, Redes Recurrentes de Kalman, Aprendizaje reforzado, Predicción de Glucosa, Pancreas Artificial, Neuroevolución, Control Predictivo por Modelos.

1. INTRODUCTION

1.1. Motivation

Recent technological advances in terms of sensors, instrumentation, networking, embedded computing, and artificial intelligence, have enabled the development of systems and applications that have changed virtually every aspect of our lives and will continue to do so. Automated insulin delivery pumps, home automation systems, self-driving cars, and security drones, are just few examples of these modern systems that effectively integrate the physical world and the cyber space, in the form of the so-called cyber-physical systems (CPSs). A number of developed countries such as Germany, the US and Japan have already pushed forward initiatives to advance in this line of research (Serpanos, 2018), and it is expected that several application domains, originally coming either from the cyber or physical world, join to this new combined framework.

Formally, CPSs are feedback control systems that focus on the interconnection of physical components and the use of complex software entities to establish new network and systems capabilities (Oks et al., 2017). In terms of implementation, a CPS is a distributed, highly scalable, interoperable, and efficient system that relies on information and communication technologies (ICT) principles, particularly, pervasive sensing, information integration, and data-driven decision making, to supervise and control its surrounding physical world (Oks et al., 2017). Even though the CPS concept is closely related to other popular terms, both in industry and in academia, such as the Internet of things (IoT), Industry 4.0., or machine to machine (M2M), the term CPS is more foundational and durable (Lee, 2015a) since it does not reference either implementation approaches (like "Internet" in IoT) nor particular applications (like the "Industry" in Industry 4.0), but rather it focuses on the fundamental problem of integrating the engineering principles that govern both cyber and physical systems.

From an architectural point of view, a CPS can be abstracted as a system formed by three levels (Cheng et al., 2018; Colombo et al., 2017), each of them areas of ongoing

research: i) field or physical, ii) connectivity or fog, and iii) cloud. Both the physical and connectivity levels of a CPS are closely related to the IoT, which focuses more on interconnectivity, interoperability, and integration of physical components to a network, with the objective of gathering information from different sources for further processing (Lee, 2015a). On the other hand, the cloud level is more related to the upper layers of classical control architectures (supervisory control, planning and management) and computer science technologies, and focuses on data-driven decision making (Munir et al., 2017) .

The integration of control principles with computer science technologies in this context represents an attractive opportunity to develop data-driven algorithms and applications for supervision, modeling, and control (Yin et al., 2014; Y. Jiang et al., 2018). In particular, in the computer science domain, data-driven techniques capable of processing large amounts of data of different nature have experienced a major breakthrough in recent years. Among them, deep learning (DL), a set of machine learning techniques that use multiple layers to progressively extract higher-level features from raw inputs (Munir et al., 2017), has emerged as a disruptive paradigm, which has led to substantial improvements in many fields when compared to traditional techniques (Singh et al., 2017; G. Guo & Zhang, 2019; Lundervold & Lundervold, 2019).

One of the major advantages of DL over traditional mathematical models or typical machine learning methods is that these techniques are able to find meaningful features from raw data without the necessity of designing first-principles models or hand-crafted features, that may require the establishment of strong assumptions, simplifications, prior knowledge of the task at hand, and a long trial and error process to decide which features may be useful, a methodology that may be prohibitive in complex applications. In fact, according to (Munir et al., 2017), one of the major contributions that academia can make to industry is the development of accurate models of physical processes to be used in control applications. This need arises from the fact that most real processes are extremely complex to be fully understood and methods involving first-principle-based mathematical

models or hand-crafted features may be limiting the performance of these systems (Munir et al., 2017).

However, although promising, the integration of CPSs with artificial intelligence (AI) techniques, in the form of the so-called Intelligent Cyber-Physical Systems (iCPSs) (King, 2021), is far from being trivial and requires overcoming several hurdles that are intrinsically attached to the problem of interacting with real physical systems in real time. Some of these challenges are not present in the original application domains of AI techniques and hence, represent a concrete limitation for unleashing the full potential of iCPSs. Indeed, several authors agree that combining CPSs with AI demands its own models and methods (NSF, 2022; Lee, 2015b; Pirani et al., 2021; Kravets, 2022), and to keep pace with the rapid advances in CPSs devices and to create more effective and better-performing systems, there is a need to accelerate the development of these new methods across application domains (NSF, 2022). This is especially critical in those domains where current automation practices are reaching their limit (Leitão et al., 2016) and using CPSs represents an opportunity for improvement.

Among the fundamental challenges that must be addressed in order to unleash the full potential of iCPSs are the following, (Lee, 2015b; Pirani et al., 2021; Kravets, 2022; M. Xu et al., 2021; Núñez et al., 2020; Yao et al., 2018; Mohammadi et al., 2018):

(i) Data quality: The poor quality of process data captured in real time, usually corrupted by noise, outliers or missing values, precludes using directly intelligent data-driven decision making algorithms, and makes it necessary to develop or include a real-time data-cleaning and reconciliation algorithm in the lower layers of the iCPS, so any data-driven application built on top can operate at its full potential.

(ii) Uncertainty estimation: Real data is full of uncertainties caused by unmeasured disturbances, partial observability of the environment, human interventions or human errors, and other unexpected phenomena. Therefore, any intelligent decision making method

included in an iCPS should consider the different sources of uncertainty and try to estimate their impact in order to make uncertainty-aware decisions.

(iii) Adaptability or personalization: Personalization of systems and adaptation to different circumstances is one of the most attractive and expected features of an iCPSs (Serpanos, 2018) and is probably the most significant difference from traditional control systems. However, gathering a large enough amount of data of a particular instance or operating condition to train DL models that achieve an acceptable performance, is usually unfeasible, time-consuming, or expensive. Hence, models to be used in an iCPS should be equipped with a mechanism to rapidly adapt or personalize with a small amount of data and with low risk of overfitting.

(iv) Constraints satisfaction: Since iCPSs are intended to interact with the real world, there are hard constraints that should be considered at the moment of designing an intelligent decision-making algorithm, especially in safety critical applications as healthcare or industrial processes. In this regard, trial and error-based techniques as vanilla reinforcement learning (RL) are not suitable for iCPSs, instead, techniques that ensure the satisfaction of these constraints should be considered.

(v) Real-time operation and hardware considerations: Finally, since the models and methods on iCPSs are hardware dependent and are subject to time constraints, a general solution should flexible enough to be able to scale or de-scale depending on the available hardware or the available computational resources at the moment, in order to always satisfy the real-time operation constraints.

Even though, some of these challenges could be addressed, at least partially, incorporating classical control, filtering and modeling techniques at different levels of a CPS, and some successful works have done so (Liu et al., 2020; Brown et al., 2019). In this thesis it is argued that intelligent methods, especially those based on DL, when carefully designed, can successfully address the aforementioned challenges, giving the CPS new capabilities, and achieving superior performance than classical methods across different

application domains. Therefore, contributing to unleash the full potential of iCPSs in the form of more effective and better-performing systems.

To validate this claim, in this thesis three different DL methods for iCPSs, materialized in three different applications, are proposed, their capabilities and novel features are analyzed, and they are compared with several classical and other intelligent approaches using multiple application-specific performance metrics. Results show the superiority of the proposed methods across different domains where the aforementioned challenges are present. It is expected that the results of this thesis will promote the development of new intelligent data-driven methods for iCPSs.

1.2. Hypothesis, Objectives and Contributions

The driving hypothesis of this work is that CPSs benefit from the incorporation of intelligent learning-based data-driven methods. In particular, from DL methods carefully designed to address three specific and relevant areas for CPSs: data-cleaning, adaptive and probabilistic modeling, and control of complex nonlinear systems. Concretely, DL methods can i) provide CPSs with new functional capabilities in these areas, e.g., personalization, learning, and automatic features extraction; and ii) achieve better performance than classical techniques, when using application-specific metrics, such as root-mean-squared error for time series denoising, mean absolute relative difference for deterministic modeling, calibration error for probabilistic modeling, and integral squared error or total energy for control performance.

Consequently, the main objective of this work is to design, implement and evaluate three intelligent DL general methods for data-cleaning, adaptive and probabilistic modeling, and nonlinear multivariate control, methods that could be used in different iCPSs applications and across different levels of the architecture. These methods will be designed to address the main challenges faced by this kind of systems that interact with the

real world and will be compared to other intelligent approaches and classical techniques that try to address the same problems.

Accordingly, the following specific objectives are derived from the foregoing:

- To develop an intelligent general purpose data-cleaning method to be used in the lower layers of a CPS, capable of improving data quality in real-time and thereby enabling other data-driven applications to operate on the upper layers of the CPS.
- To develop an adaptive learning-based model capable of adapting to different instances with minimal amount of additional data and with low risk of overfitting. Additionally, this model should be able to estimate the inherent uncertainties of the target application.
- To develop a learning-based data-driven controller capable of scaling or descaling on demand based on the available computational resources. Additionally, this controller should have a mechanism to enforce both soft and hard constraints during the learning process and during real-time operation.

In the achievement of the specific objectives, this thesis makes the following contributions:

- A novel learning-based data-cleaning method based on an autoencoder architecture and a noise contrastive regularization, which forces the network to exploit the temporal structure of the data and cross-correlations between input variables in order to produce a clean version of the input signals in real-time.
- An adaptative model that is used for the task of real-time glucose prediction for patients with type 1 diabetes (T1D). The model is able to adapt, by using meta-learning, to different patients with a minimal amount of patient-specific data and with low risk of overfitting. Additionally, an extension of this model is proposed to estimate the uncertainty produced by the user's misspecification of meals (time and size), a very common problem in T1D modeling and control.

- Finally, a novel model-based neuroevolutive controller with learning capabilities based on the map elites optimization algorithm is proposed. At each optimization step, the controller not only takes proper control actions, but also gradually learns how to control the system, so that, optimization at each step can eventually be avoided. Additionally, the controller is flexible enough to change its computational burden on demand and has an intuitive and simple way to enforce hard constraints during the optimization process.

1.3. Organization

The rest of this thesis is organized as follows[1]. In the remainder of this chapter notation and basic definitions used throughout the thesis will be established. In chapter 2 the proposed learning-based data-cleaning method to be used in lower layers of an iCPS is described in detail. Chapter 3 presents the implementation of the adaptative model for glucose prediction and its probabilistic extension. Then, chapter 4 presents the proposed neuroevolutive controller and its evaluation in different industrial processes. Finally, chapter 5 states concluding remarks and directions for future research.

1.4. Notation and Basic Definitions

In this thesis, the following notation will be adopted. $\mathbb{R}$ denotes the real numbers, $\mathbb{Z}_{\geq 0}$ denotes the nonnegative integers, $\mathbb{R}^N$ denotes the Euclidean space of dimension N, and $\mathbb{R}^{N\times M}$ denotes the set of $N \times M$ matrices with real coefficients. For $a, b \in \mathbb{Z}_{\geq 0}$ we use $[a : b]$ to denote their closed interval in $\mathbb{Z}_{\geq 0}$. Matrices and vectors are denoted in bold fonts, the former also capitalized. For a vector $\mathbf{v} \in \mathbb{R}^N$, $\mathbf{v}^i$ denotes its ith component, and $[\mathbf{v} \oplus \mathbf{u}]$ denotes its concatenation with another vector $\mathbf{u} \in \mathbb{R}^N$. For a matrix $\mathbf{A} \in \mathbb{R}^{N\times M}$, $[\mathbf{A}]_i$ denotes its ith column and $[\mathbf{A}]^i$ its ith row. For an N-dimensional real-valued sequence

[1]This thesis follows the "3-paper format" of the Pontificia Universidad Católica de Chile, in which the thesis is the compendium of three research articles written by the student. Although redundant material between chapters has been minimized, some redundancy is unavoidable since removing material affects the flow of the articles in the compendium.

$\alpha : \mathbb{Z}_{\geq 0} \rightarrow \mathbb{R}^N$, $\alpha(n) \in \mathbb{R}^N$ denotes its nth element, and $\alpha_{[a:b]}$ denotes its restriction to the interval $[a : b]$, i.e., a sub-sequence. Similarly, $\alpha_{\sim r[a:b]}$ denotes a sub-sequence of length r whose elements are randomly chosen from $\alpha_{[a:b]}$. For a sub-sequence $\alpha_{[a:b]}$, $\mathbf{M}(\alpha_{[a:b]}) \in \mathbb{R}^{N \times (b-a+1)}$ is a matrix whose ith column is equal to $\alpha(a + i - 1)$, with $i \in [1 : b - a + 1]$. Given an N-dimensional (sub-)sequence α, we define its T-depth window as a matrix-valued sequence $\beta : \mathbb{Z}_{\geq 0} \rightarrow \mathbb{R}^{N \times T}$, where $\beta(n) = \mathbf{M}(\alpha_{[n-T+1:n]})$. Given a finite set of random samples $\mathcal{X}$, and a continuous function $f : \mathcal{X} \rightarrow \mathbb{R}$, $\mathbb{E}_{\mathcal{X}}[f(\cdot)]$ denotes the expectation of f from $\mathcal{X}$.

2. CONTRASTIVE BLIND DENOISING AUTOENCODER FOR REAL TIME DENOISING OF (INDUSTRIAL) IOT SENSOR DATA

This Chapter deals with the development of an intelligent data-cleaning method to be used in the lower layers of an iCPS. Although this is a general method, in this chapter the focus is on an Industrial Internet of Things application [1].

2.1. Context

The incorporation of Industrial Internet of Things (IIoT) technologies to modern industrial facilities enables the real-time acquisition of an enormous amount of process data, typically in the form of time series, which represents an opportunity to improve performance by using data-driven algorithms for supervision, modeling and control (Langarica et al., 2020; Langarica & Núñez, 2021).

Data-driven techniques, as statistical or machine learning algorithms, are capable of dealing with the multivariate and intricate nature of industrial processes; however, they rely on the consistency and integrity of the data to work properly (Núñez et al., 2020). This imposes a limitation on the application of these algorithms in real facilities, since IIoT sensor data is often highly corrupted with outliers and noise caused by multiple factors, as environmental disturbances, human interventions, and faulty sensors (Núñez et al., 2020). As a result, the vast majority of successful data-driven case studies use offline preprocessed data, simulations or databases generated in a controlled environment, and applications in real industrial environments are scarce.

A common practice for dealing with noisy process data is the use of smoothing filters (S. Xu et al., 2015), like discrete low-pass filters, the Savitsky-Golay (SG) filter or exponential moving average (EMA) filters. The main drawback of these techniques is their univariate nature, which causes that redundancy and correlations among variables are not

[1] The material in this chapter has been published in (Langarica & Núñez, 2023) and a preliminary version of this method in (Langarica et al., 2020)

exploited for denoising. Multivariate denoisers, which can exploit cross-correlation between signals, are a natural improvement to univariate filters. Approaches like Kalman or particle filters are common techniques used in industry; however, they require the selection of a suitable model and an a-priori estimation of parameters, as the covariance matrices in the Kalman filter, which could be very hard to estimate accurately.

The use of transforms, like Wavelets (Donoho, 1995) or Gabor (Nezamoddini-Kachouie & Fieguth, 2005), is also a typical approach that exploit statistical properties of the noise, so the signal can be thresholded in the transformed domain to preserve only the high-valued coefficients, and then, by applying the inverse transform, obtain a cleaner signal. A limitation of this kind of techniques is the difficulty of knowing a-priori the best basis for representing the signals and without knowledge of the noise nature, is hard to determine where to threshold.

Classical learning-based denoising algorithms, like principal component analysis (PCA), Kernel PCA, or dictionary learning (Shao et al., 2014), solve some of the problems of fixed transforms by learning a suitable representation of the data in a transformed space. These approaches are also multivariate in nature, hence exploit spacial correlations between input signals; nevertheless, they were designed for static data and, consequently, important information from temporal correlations are not exploited at all.

Recently, denoising autoencoders (DAEs) (Yu & Zhao, 2020) have emerged as a learning-based denoising technique that is multivariate in nature and is capable of learning complex nonlinear structures and relationships between variables. Originally, DAEs emerged for image denoising, but with the use of recurrent neural networks, their application in denoising dynamical data has been successful (Chaitanya et al., 2017). Unlike PCA, dictionary learning or fixed transforms techniques, DAEs are not blind in the sense that for learning to denoise a signal, the clean version of the signal (the target) has to be known beforehand. In addition, information about the characteristics of the noise affecting the signal is required to create realistic training examples. This is an important drawback

for using DAEs in real-world applications, where the clean version of the signal and the noise characteristics are usually unknown.

In this chapter, inspired by: i) the results in (Lehtinen et al., 2018), which show that, under mild conditions, it is possible to recover clean observations by only looking at corrupted ones; and ii) the recent advances in self-supervised learning (Liu et al., 2020), we propose a novel blind denoising autoencoder for real-time denoising of IIoT sensor data, called Contrastive Blind Denoiser Autoencoder (CBDAE).

The proposed CBDAE uses noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2010) as a regularization technique over the latent space of a recurrent autoencoder to achieve blind denoising of multivariate time series. The CBDAE preserves all the advantages of DAEs, i.e., it is multivariate and purely data-driven, and additionally, it eliminates the need during training of the clean version of the signals and prior knowledge of the noise characteristics, i.e., it achieves blind denoising.

Consequently, the concrete contributions of this chapter are three-fold. First, as main contribution, we introduce a novel technique, the CBDAE, which uses NCE as a temporal regularization over the latent space. Second, we present a methodology for finding *hard negative* examples, which is an active field of research in contrastive learning, to train the CBDAE more efficiently. Finally, we show that using NCE regularization induces smooth, meaningful and compact representations of input sequences, which can be used for other downstream tasks such as fault detection or prediction in the latent space.

2.2. Related work

Numerous learning-based denoising techniques that use neural networks have been proposed recently, which have consistently shown a superior performance than classical techniques as Kalman filters or Wavelets. However, most of these efforts are not directly applicable in online real applications since they: i) require the clean version of the signal to compute the loss; or ii) include non-causal filters that require the entire signal at once

for denoising. In this context, the relevant state of the art can be divided into efforts that use neural networks to denoise time series and efforts that achieve blind denoising using neural networks but not necessarily targeting time series.

2.2.1. Time series denoising with neural networks

In the context of industrial applications, (Liu et al., 2019) proposes a 1-D convolutional autoencoder for denoising measured signals from rotating machines as a preprocessing step before using a classifier for intelligent fault diagnosis. The proposed denoising autoencoder requires the clean version of the signal for training; therefore, although an interesting proof of concept, it has no practical use. In (C. Jiang et al., 2019), a mixed neural network architecture is proposed consisting in two GRU layers followed by two LSTM layers for denoising data from a micro-electro-mechanical system inertial measurement unit (MEMS-IMU), which is the core component in navigation systems. In this case, only static data was used and no regularization nor constraint was imposed over the identity loss function. Even though this approach seems effective, it solely relies on the neural architecture and the summarizing capabilities of recurrent networks, and as it will be shown in section 4.1, if the size of the network is not carefully tuned for the specific application and data, this could lead to over-fitting the noise as well.

In a different setting where time series denoising is also of utmost importance, several works have targeted ECG data denoising. For example, (Arsene et al., 2019) compares a LSTM with a convolutional neural network (CNN) for this task. Even though both show similar performance, the CNN requires less parameters and trains faster. A related approach is presented in (Frusque & Fink, 2022), where the threshold for wavelet coefficients are learned using a neural network and then the transformation is applied to the signal for denoising. Although of interest, both of these approaches require the clean targets for training, and thus are not applicable in real environments where this information is not available.

2.2.2. Blind denoising with neural networks

Since the breakthrough of (Lehtinen et al., 2018), where it is shown that a neural network can be trained to denoise its inputs by only using the corrupted versions as a target, i.e., blind denoise, as long as the noise is zero-mean and some kind of regularization is applied, many works have targeted blind denoising using neural networks in image-related applications. Most of these approaches have taken advantage of image-specific assumptions, as spatial smoothness, similarity across patches of the same image, and sparsity (Batson & Royer, 2019). For example, based on the sparsity assumption, a naive but still effective approach is to use the bottleneck of an autoencoder to learn a compressed representation of an image, leaving out uncorrelated and unimportant information as noise (Laakom et al., 2022); however, this requires extensive experimentation to find the optimal size of the network for a particular dataset. In this same line, (Majumdar, 2019) proposed a more sophisticated approach based on an autoencoder network to learn a dictionary decomposition of an image for denoising. Even though this approach does not require a clean target nor extensive hyper-parameter tuning, it is an iterative procedure that requires optimization for every instance; therefore, is unsuitable for online applications. Other works have exploited the spatial smoothness assumption by randomly masking the input images by zeroing out random pixels or replacing them by a function of their neighboring pixels (Batson & Royer, 2019; Krull et al., 2018; Lecoq et al., 2021). Although these works have been successful, it is not clear what percentage of the input pixels should be masked out for optimal performance and what kind of mask should be used for a specific application; therefore, extensive trial and error experimentation is required (Lecoq et al., 2021).

Similar to image-related blind denoising methods, the proposed CBDAE is inspired by the results in (Lehtinen et al., 2018) to achieve blind denoising. However, in this case we rely on the temporal smoothness of time series measured from an underlying dynamical system, to regularize the latent space of an autoencoder using our proposed NCE loss. The CBDAE not only successfully achieves blind denoising with minimal hyper-parameter tuning, but also is applicable to online real industrial applications and

is especially suitable for other data-driven control applications built in top, where data quality is of utmost importance (Das Sharma et al., 2021; Zamfirache et al., 2022b, 2022a; Precup et al., 2022).

2.3. Theoretical background

2.3.1. Autoencoders and Denoising Autoencoders

Autoencoders (AEs) (Hinton & Salakhutdinov, 2006) are unsupervised neural networks trained to reconstruct their inputs at the output layer, passing through an intermediate layer usually of lower dimension than the inputs.

Formally, given an N-dimensional sequence $\mathbf{y}$, the AE maps an input vector $\mathbf{y}(n) \in \mathbb{R}^N$ to a latent representation $\mathbf{z} \in \mathbb{R}^M$, with $M < N$, using a function f_{θ_E}, which in the simplest case is a linear layer with σ as an arbitrary activation function, namely,

$$\mathbf{z} = f_{\theta_E} = \sigma(\mathbf{E}\mathbf{y}(n) + \mathbf{b}), \tag{2.1}$$

where $\mathbf{E} \in \mathbb{R}^{M \times N}$ and $\mathbf{b} \in \mathbb{R}^M$ are trainable parameters of the network. In more complex approaches, f_{θ_E} can be chosen to be any type of layer, like recurrent or convolutional layers, or even a stack of multiple layers.

After encoding, the latent vector is mapped back to the input space by a second function g_{θ_D} known as the decoder:

$$\hat{\mathbf{y}}(n) = g_{\theta_D} = \sigma(\mathbf{D}\mathbf{z} + \mathbf{e}), \tag{2.2}$$

where $\mathbf{D} \in \mathbb{R}^{N \times M}$ and $\mathbf{e} \in \mathbb{R}^N$.

During training, the parameters of the AE are found by solving the optimization problem in (2.3) using gradient descent and the back-propagation algorithm,

$$\theta^* = \arg\min_\theta ||\hat{\mathbf{y}}(n) - \mathbf{y}(n)||^2, \tag{2.3}$$

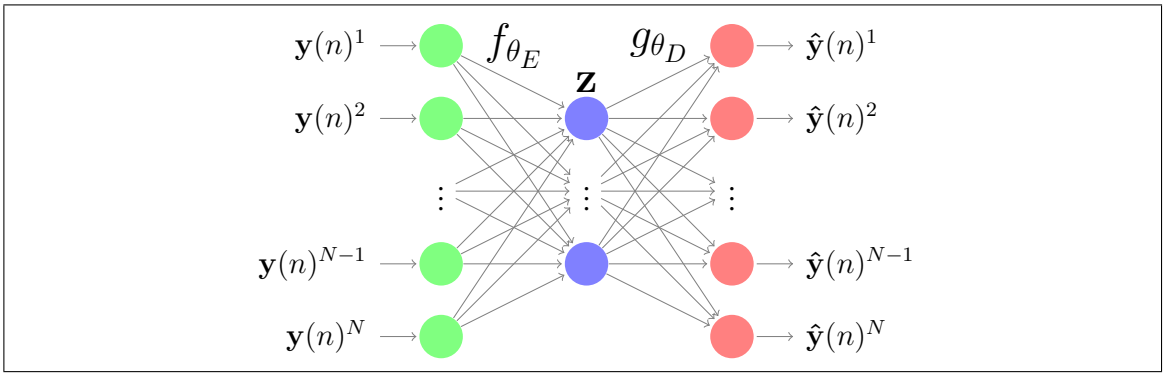


Figure 2.1. Simple AE architecture where f_{θ_E} encodes the input vector $\mathbf{y}(n)$ to a latent representation z and then, g_{θ_D} decodes $\mathbf{z}$ to reconstruct the input as $\hat{\mathbf{y}}(n)$.

where θ accounts for all the trainable parameters. Fig. 2.1 illustrates a simple AE network.

When the input of the AE is corrupted with noise and the target output is clean, the resulting latent space is more robust, contains richer features, and the AE learns how to denoise corrupted inputs (Vincent et al., 2008). This is the working principle of the so called Denoising Autoencoders (DAEs), and has led to many denoising applications in static data, as images (Creswell & Bharath, 2019), and with the use of recurrent AEs (Chaitanya et al., 2017) this technique has also been effectively applied to dynamic data. However, DAEs work under the assumption that the clean version of the input data is available (since is used as the target during training), as well as information about the noise. This fact limits the application of DAEs to IIoT sensor data where, in general, none of these requirements are fulfilled.

2.3.2. Noise contrastive estimation

Noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2010) is a learning method for fitting unnormalized distributions that has been adapted for different machine learning tasks, such as natural language processing (NLP) (A. Mnih & Kavukcuoglu, 2013), time series feature extraction (Hyvarinen & Morioka, 2016) and semi-supervised image and audio classification (Chen et al., 2020; van den Oord et al., 2018). The basic idea of NCE

is to estimate the parameters of the model by learning to discriminate between samples from the target distribution and samples from an arbitrary noise distribution, transforming complex density estimation tasks into probabilistic binary classification ones. NCE is based on "learning by comparison" and thus, falls under the family of contrastive learning methods (Liu et al., 2020).

Mathematically, let $\mathbf{y} \in \mathbb{R}^N$ be an input data vector belonging to one of L possible classes. Denote as $\mathbf{y}^+$ and $\mathbf{y}^-$ positive and negative examples of $\mathbf{y}$, respectively, meaning that $\mathbf{y}^+$ and $\mathbf{y}$ are of the same class, and $\mathbf{y}^-$ belongs to a different class. Consider a generic nonlinear learnable transformation with parameters θ, $f_\theta : \mathbb{R}^N \rightarrow \mathbb{R}^M$, which takes elements from the input data space and projects them to another space (for example, an encoder). The NCE loss can be formulated as:

$$L_{NCE} = \mathbb{E}_{\mathbf{y},\mathbf{y}^+,\mathbf{y}^-}\left[-\log\left(\frac{e^{f_\theta(\mathbf{y})^T f_\theta(\mathbf{y}^+)}}{e^{f_\theta(\mathbf{y})^T f_\theta(\mathbf{y}^+)} + e^{f_\theta(\mathbf{y})^T f_\theta(\mathbf{y}^-)}}\right)\right]. \tag{2.4}$$

The NCE loss has the property of maximizing a lower bound of the mutual information between $\mathbf{y}$ and $\mathbf{y}^+$ (van den Oord et al., 2018). However, as noted by (Sohn, 2016), the NCE loss with only one negative example per update yields to slow convergence and is prone to getting stuck in a local optima. Therefore, the NCE loss is modified by (Sohn, 2016) to use multiple negative examples, yielding a formulation known as the InfoNCE loss (van den Oord et al., 2018):

$$L_{NCE} = \mathbb{E}_{\mathcal{Y}}\left[-\log\left(\frac{e^{f_\theta(\mathbf{y})^T f_\theta(\mathbf{y}^+)}}{\sum_{\bar{\mathbf{y}}\in\mathcal{Y}} e^{f_\theta(\mathbf{y})^T f_\theta(\bar{\mathbf{y}})}}\right)\right], \tag{2.5}$$

where $\mathcal{Y}$ is a finite set of samples containing only one positive example, $\mathbf{y}^+$, and multiple negative ones. However, even if multiple negative examples are used, slow convergence might still be a problem, particularly when negative examples are far from positive ones, which produces a near-zero loss and very small gradients early in training. How to select the so-called *hard negative* examples (those very close to the target but still further than positive examples) is still an open problem in the literature.

Other proposals have been made to improve performance of the NCE loss, as using other similarity functions, e.g., cosine similarity, and introducing a temperature parameter (Chen et al., 2020).

2.4. Process data denoising using a Contrastive Blind Denoising Autoencoder

In this work, we aim to transfer the blind denoising results obtained for images in (Lehtinen et al., 2018) and (Majumdar, 2019) to time series by using a recurrent AE. The use of a recurrent AE seems natural given its ability to process sequential data, and its proven summarizing capabilities that can be useful for denoising (C. Jiang et al., 2019). In particular, in (Lehtinen et al., 2018) it was shown that a neural network can be trained to denoise its inputs by only using corrupted versions as a target, i.e., blind denoise, as long as the noise is zero-mean. Nonetheless, it was also shown that when an AE with enough capacity is used, at some point it not only learns high level features, which is fundamental for data-based denoising, but also overfits the data and reproduces noise as well. Fig. 2.2 shows this effect when training a vanilla recurrent AE to denoise time series.

To solve this issue, we propose to use NCE regularization to transfer temporal smoothness from the input time series to the latent space of the AE. Our application of the NCE regularization forces the latent space vectors to encode the underlying shared information between different parts of the input signals, i.e., dynamical characteristics of the process, leaving out low-level information, as noise (van den Oord et al., 2018). Additionally, the NCE regularization contributes to obtaining meaningful and compact representations of the input time series, which can be used for other downstream tasks, as it has been shown in prior works (Wang & Gupta, 2015).

2.4.1. General setup

Consider a multi-input multi-output dynamical system, which is sampled periodically by an IIoT system with period τ. From the IIoT system perspective, the process can be

modeled as a discrete-time system given by

$$\mathbf{x}(n+1) = F(\mathbf{x}(n), \mathbf{w}(n)), \tag{2.6}$$

where n denotes the time step, $\mathbf{x}(n) \in \mathbb{R}^W$, with W the order of the system, denotes the current value of the internal state, $\mathbf{x}(n+1)$ is the future value of the state, $\mathbf{w}(n) \in \mathbb{R}^Q$ denotes process inputs, and $F : \mathbb{R}^W \times \mathbb{R}^Q \to \mathbb{R}^W$ is a smooth nonlinear mapping governing the dynamics of the system.

The measurement (observation) process is given by

$$\tilde{\mathbf{y}}(n) = C(\mathbf{x}(n), \mathbf{v}(n)), \tag{2.7}$$

where $\tilde{\mathbf{y}}(n) \in \mathbb{R}^N$ denotes the measurements vector generated by the IIoT system, C is the output mapping, and $\mathbf{v}(n)$ represents measurement noise. For a noise-free condition, we denote the noise-free (unknown) measurements vector at time step n as:

$$\mathbf{y}(n) = C(\mathbf{x}(n), \mathbf{0}). \tag{2.8}$$

Under this setup, the measurement process generates a sequence $\tilde{\mathbf{y}} : \mathbb{Z}_{\geq 0} \to \mathbb{R}^N$, where, without loss of generality, $N \neq W$, therefore:

$$\tilde{\mathbf{y}}(n) = [\tilde{\mathbf{y}}^1(n), \tilde{\mathbf{y}}^2(n), \ldots, \tilde{\mathbf{y}}^N(n)]^T \in \mathbb{R}^N, \tag{2.9}$$

where each $\tilde{\mathbf{y}}^i(n)$ represents a real-valued measurement of the ith system output at time step n.

2.4.2. Problem formulation

Let $\tilde{\mathbf{Y}}(n)$ be the T-depth window of $\tilde{\mathbf{y}}$, i.e.,

$$\tilde{\mathbf{Y}}(n) = [\tilde{\mathbf{y}}(n-T+1), \tilde{\mathbf{y}}(n-T+2), \ldots, \tilde{\mathbf{y}}(n)] \in \mathbb{R}^{N \times T}, \tag{2.10}$$

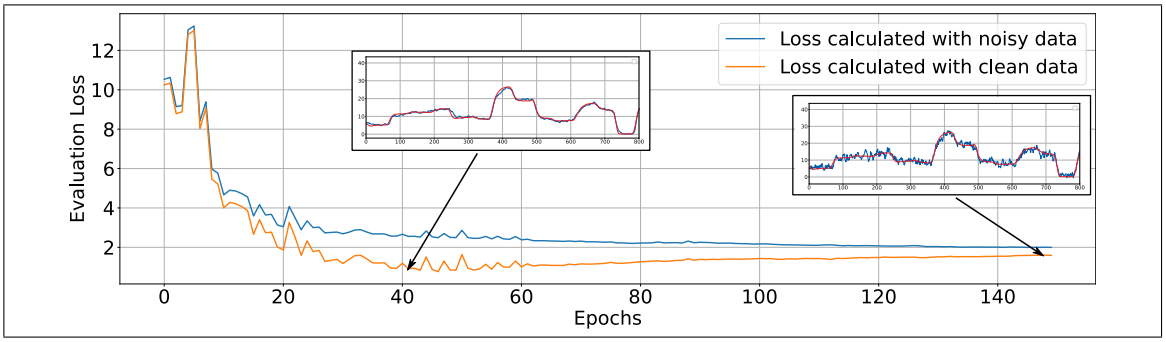


Figure 2.2. Blind denoising with a recurrent AE with no regularization. The light blue line represents the loss calculated using the output of the AE and the noisy signal, i.e., blind denoising, while the orange line represents the loss calculated using the output of the AE and the clean version of the signal (not available for training), which is the real objective we would like to minimize, but in practice is not possible to calculate. The network learns to denoise the signal at some point, as is shown in the left subplot in which the AE cleans the signal successfully, (red curve represents the clean version of the signal and the dark blue curve the autoencoder output), but as training continues it begins to overfit, reproducing noise as well, as shown in the right subplot.

hence $[\tilde{\mathbf{Y}}(n)]_k \in \mathbb{R}^N$ is a column vector containing the N measurements at time instant $n - T + k$, and $[\tilde{\mathbf{Y}}(n)]^k \in \mathbb{R}^T$ is a row vector containing the last T measurements of the k-th output.

At each time step n, the CBDAE uses $\tilde{\mathbf{Y}}(n)$ to generate an estimate of $\mathbf{y}(\mathbf{n})$, denoted as $\hat{\mathbf{y}}(n)$. Since the CBDAE is a dynamical system itself that processes data sequentially (due to the use of recurrent neural networks) to accomplish this task, two timescales should be introduced in the formulation. The *process time* refers to the time scale at which (2.6) evolves, which is ruled by τ and indexed by n, while the *CBDAE time* refers to the internal time of the CBDAE, which is ruled by a host computer's processor time step and is indexed by j for each fixed n.

Since the objective of the CBDAE is to operate in real-time, the *CBDAE timescale* has to be fast enough to process the data in $\tilde{\mathbf{Y}}(n)$ before another set of measurements, generated at the *process timescale*, becomes available. Hence, we have two dynamical

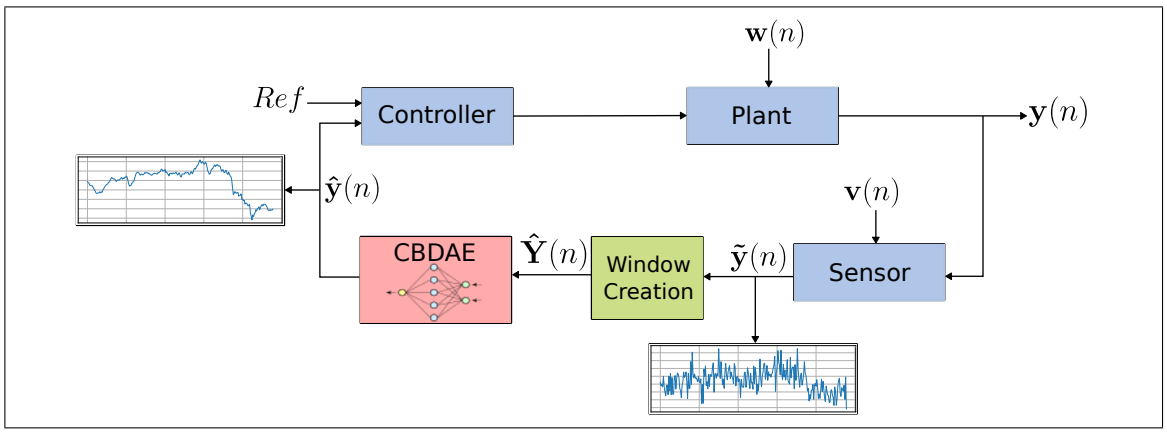


Figure 2.3. Control scheme that incorporates the proposed CBDAE to denoise corrupted signals measured by the sensor. $\mathbf{y}(\mathbf{n})$ denotes the real value of the output signals at time n; $\tilde{\mathbf{y}}(n)$ denotes the measured signal by the sensor; $\tilde{\mathbf{Y}}(n)$ represents the T-depth window of measurements, which is the CBDAE input; and $\hat{\mathbf{y}}(n)$ denotes the output of the CBDAE, which approximates $\mathbf{y}(n)$.

systems at work, the slow system (the process) and the fast system (the CBDAE), which at each process time step gets triggered and iterates itself a number of steps depending on the size of $\tilde{\mathbf{Y}}(n)$ and the CBDAE architecture.

Hence, summarizing, the problem to be solved is to generate in real time an estimate $\hat{\mathbf{y}}(n)$ of the noise-free measurement vector $\mathbf{y}(n)$ by processing $\tilde{\mathbf{Y}}(n)$. Unlike typical DAEs, in this case we do not have access to $\mathbf{y}(n)$; therefore, denoising must be done in a *blind* manner. To illustrate the potential of the proposed approach, Fig. 2.3 illustrates how the CBDAE can be incorporated in a data-driven control scheme.

2.4.3. Blind Denoising

The working principle of the proposed blind denoising technique will be divided in three steps, which are detailed in the following.

2.4.3.1. Autoencoder network

As mentioned, due to its recurrent nature the autoencoder network is a dynamical system itself; hence, in the following is formulated in dynamical terms, stressing the (local)-time dependence of its internal variables. First, at process time n, the encoder generates a latent representation $\mathbf{h_L}$ of $\tilde{\mathbf{Y}}(n)$, which is a Q_L-dimensional finite-length sequence of length $T+1$ (an initial condition plus one element for each column of $\tilde{\mathbf{Y}}(n)$), by feeding the network iteratively with the columns of $\tilde{\mathbf{Y}}(n)$ and then recursively feeding back the internal state of the network, namely:

$$\mathbf{h_L}(j) = f_{enc}([\tilde{\mathbf{Y}}(n)]_j, \mathbf{h_L}(j-1)),\ j \in [1:T], \tag{2.11}$$

where f_{enc} represents a recurrent network, with L layers and Q_l neurons in each layer $l \in [1:L]$, that in our case is composed by Gated Recurrent Units (GRU) (Cho et al., 2014). The (sequential) operations at each GRU layer are given by:

$$\begin{aligned}
\mathbf{z_l}(j) &= \sigma(\mathbf{W_{z_l}}\mathbf{p_l}(j) + \mathbf{U_{z_l}}\mathbf{h_l}(j-1) + \mathbf{b_{z_l}}) \\
\mathbf{r_l}(j) &= \sigma(\mathbf{W_{r_l}}\mathbf{p_l}(j) + \mathbf{U_{r_l}}\mathbf{h_l}(j-1) + \mathbf{b_{r_l}}) \\
\mathbf{n_l}(j) &= \tanh(\mathbf{W_{n_l}}\mathbf{p_l}(j) + \mathbf{r_l}(j) \circ (\mathbf{U_{n_l}}\mathbf{h_l}(j-1) + \mathbf{b_{n_l}})) \\
\mathbf{h_l}(j) &= (1 - \mathbf{z_l}(j)) \circ \mathbf{n_l}(j) + \mathbf{z_l}(j) \circ \mathbf{h_l}(j-1),
\end{aligned} \tag{2.12}$$

where $\mathbf{h_l}(j) \in \mathbb{R}^{Q_l}$ is the hidden state of layer l at CBDAE time j, $\mathbf{r_l}$, $\mathbf{z_l}$ and $\mathbf{n_l}$ are Q_l-dimensional finite-length sequences representing the value of reset, update and new gates of layer l respectively. σ and $\tanh$ are the sigmoid and the hyperbolic tangent activation functions, $\circ$ is the Hadamard product and $\mathbf{p_l}(j)$ is the input of layer l at CBDAE time j. For the first layer ($l = 1$) the input, $\mathbf{p_1}(j)$ is given by the input of the network $[\tilde{\mathbf{Y}}(n)]_j$, and for the subsequent layers the input is given by $\mathbf{h_{l-1}}(j)$, which is the current hidden state of the previous layer (this stresses the sequential nature of the CBDAE).

Finally, the latent representation of $\tilde{\mathbf{Y}}(n)$ is selected to be $\mathbf{h_L}(T) \in \mathbb{R}^{Q_L}$, which is the last hidden state of the last layer. This vector will be denoted as $\mathbf{h}(T)$ from now on by dropping L for the sake of clarity.

After the final latent representation is obtained, the decoder takes the final hidden state of each encoder layer $\mathbf{h}_i(T)$ as the initial hidden state of each of its recurrent layers, which implies an equal number of layers, and begins to decode the sequence as

$$\mathbf{d}_L(j) = f_{dec}(\mathbf{p_j}, \mathbf{d}_L(j-1)),\ j \in [1:T] \tag{2.13}$$

where $\mathbf{p_j} \in \mathbb{R}^N$ is the input for each decoding step of the network. The first input ($j = 1$) is given by the first element of the target sequence $[\tilde{\mathbf{Y}}(n)]_1$, while for subsequent steps the input is given either by the delayed target $[\tilde{\mathbf{Y}}(n)]_{j-1}$ or the previous network estimate $[\hat{\mathbf{Y}}(n)]_{j-1}$, based on a strategy that will be explained in the following. The network estimates are calculated by projecting $\mathbf{d_L}(j)$ using a linear transformation o_{dec} as

$$[\hat{\mathbf{Y}}(n)]_j = o_{dec}(\mathbf{d_L}(j)) = \mathbf{W_{out}}\mathbf{d_L}(j) + \mathbf{b_{out}}, \tag{2.14}$$

where $\mathbf{W_{out}} \in \mathbb{R}^{N \times Q_L}$ and $\mathbf{b_{out}} \in \mathbb{R}^N$ are trainable parameters. Then, the final output of the CBDAE corresponds to $\hat{\mathbf{y}}(n) = [\hat{\mathbf{Y}}(n)]_T$. Note that the decoder starts its processing once $\mathbf{h}(T)$ is available, which involves T previous iterations of the encoder at CBDAE time. Therefore, careful should be taken when interpreting the local time of the decoder.

Two modifications are proposed to this vanilla version of the recurrent AE to enhance its denoising capabilities. First, it was found that *scheduled sampling* (Bengio et al., 2015) is beneficial for denoising. This technique consists in using at early training stages the target at $j-1$ ($[\tilde{\mathbf{Y}}(n)]_{j-1}$) as the input of the decoder to predict the target at j, and as training progresses, increasingly using the network estimates ($[\hat{\mathbf{Y}}(n)]_{j-1}$) to estimate the target at j. This shift is controlled by a probability p_d, that in our case is increased linearly as

$$p_d = min(1, k_d + c_d e), \tag{2.15}$$

where k_d and c_d are parameters and e is the training epoch.

Shifting gradually from using the target as the input in each step to using the network's own estimates improves the stability of the training process, specially when the decoded sequence is long. Moreover, it forces $\mathbf{h}(T)$ to retain all the information of the

input sequence in a compressed manner, since the entire sequence has to be recovered from it. This helps to retain only important information about the sequence, leaving out non-relevant information as noise.

The second modification to the recurrent AE, is the incorporation of an additional transformation that maps $\mathbf{h}(T)$ to another space, where the contrastive loss is applied. This transformation is given by

$$\mathbf{z}(T) = g(\mathbf{h}(T)) = \mathbf{W}_{g2}\sigma(\mathbf{W}_{g1}(\mathbf{h}(T))) \in \mathbb{R}^G, \tag{2.16}$$

where $\mathbf{W}_{g1} \in \mathbb{R}^{G_1 \times Q_L}$ and $\mathbf{W}_{g2} \in \mathbb{R}^{G \times G_1}$ are trainable parameters and σ represents the Relu activation function. According to (Chen et al., 2020), using this additional transformation is beneficial since the contrastive loss induces invariance of the representation between positive examples, hence applying it directly in the latent space could be detrimental for downstream tasks, where it is desirable that positive examples have similar but not equal representations.

2.4.3.2. Selection of hard negative examples for NCE loss

As mentioned before, the incorrect selection of negative examples makes the network to exhibit sub-optimal performance and slow convergence (Wu et al., 2020). Therefore, is vital to develop an effective method for obtaining them, in the context of time series.

Although still an open problem, a typical approach for finding hard negatives is to randomly sample a batch of B negative examples, calculate the score of each element and select the K elements with higher loss values for the training step (Wang & Gupta, 2015).

Instead, we propose a hybrid scheme that takes advantage of the temporal consistency of the time series for selecting hard negatives. To this end, given a training database in the form of a finite length sequence $\tilde{\mathbf{Y}}$ of length $\bar{T}$, for a given index k, define a batch $\mathbf{B_c}$ as a sub-sequence of length C as $\tilde{\mathbf{Y}}_{[k-C+1:k]}$, i.e., a sub-sequence of consecutive data matrices chained in time. Similarly, define a batch $\mathbf{B_r}$ as a sub-sequence of R randomly chosen

matrices $\tilde{\mathbf{Y}}_{\sim R[0:\bar{T}-1]}$. In each training iteration a batch is formed by concatenating both batches (sub-sequences) as (see Fig. 2.4)

$$\mathbf{B} = \mathbf{B_c} \oplus \mathbf{B_r}, \tag{2.17}$$

where $\oplus$ denotes the concatenation operator. For each element of $\mathbf{B}$ the following operations are performed: i) a forward pass of the encoder; ii) transformation g to the latent vector; and iii) concatenation of the resulting projections to obtain the following matrix

$$\mathbf{Z} = [\mathbf{Z_c} \oplus \mathbf{Z_r}] \in \mathbb{R}^{G\times(C+R)} \tag{2.18}$$

Since consecutive columns of $\mathbf{Z_c}$ are obtained by processing consecutive elements from $\tilde{\mathbf{Y}}$, the rationale is that their representations should be similar and very close in the latent space of the CBDAE. However, because of the temporal structure of the time series, it is desirable that $[\mathbf{Z_c}]_{j\pm1}$ is closer to $[\mathbf{Z_c}]_j$ than $[\mathbf{Z_c}]_{j\pm2}$, and $[\mathbf{Z_c}]_{j\pm2}$ is closer than $[\mathbf{Z_c}]_{j\pm3}$, and so on. Therefore, when iterating through the batch, we can select $[\mathbf{Z_c}]_{j\pm1}$ as a positive example of $[\mathbf{Z_c}]_j$ to increase the mutual information of consecutive representations, thus helping to transfer the dynamic characteristics of the underlying system to the latent space of the CBDAE. On the other hand, all the other representations $[\mathbf{Z_c}]_{k\neq j\pm1}$ are selected as negative examples to push them further from $[\mathbf{Z_c}]_j$ than $[\mathbf{Z_c}]_{j\pm1}$. These representations are inherently hard negatives because they will remain close due to the dynamical structure the space is learning. Similarly, $\mathbf{Z_r}$ representations are always used as negative examples.

Note that using only $\mathbf{B_c}$ with subsequent and highly correlated examples for training could damage learning since the batch is also used to train the autoencoding part (Bengio, 2012). Also, as pointed out by (Wu et al., 2020), randomly selected samples could be beneficial in different stages of training when using contrastive losses, particularly in early stages. Therefore, we consider beneficial to use a combination of randomly selected and consecutive samples.

2.4.3.3. Loss calculation

After processing the batch and obtaining $\mathbf{Z}$, similar to (Chen et al., 2020), the NCE loss is calculated using

$$l(i,j,q) = -\log\left(\frac{\exp(sim([\mathbf{Z}]_i,[\mathbf{Z}]_j))}{\sum_{k=1}^{C+R} 1_{k\neq i,q}\exp(sim([\mathbf{Z}]_i,[\mathbf{Z}]_k))}\right), \tag{2.19}$$

where sim stands for the cosine similarity, although, any similarity measure can be used, and $1_{k\neq i,q}$ is an indicator function evaluating to 1 if $k \neq i$ and $k \neq q$ and zero otherwise. To obtain the final NCE loss L_{NCE}, $l(i,j,q)$ is computed across all pairs $(j, j+1)$ and $(j+1, j)$ (since the NCE loss is not symmetric), where $j \in [1, c-1]$ because only columns in $\mathbf{Z_c}$ are considered as positive pairs. Special attention should be given to index q, which is used to exclude examples from the loss calculation and is selected equal to $j-1$ when the loss is calculated for the pair $(j, j+1)$ and equal to $j+2$ when the pair is $(j+1, j)$. This is because when calculating the loss around $[\mathbf{Z}]_j$ with $[\mathbf{Z}]_{j+1}$ as positive example, all other examples in $\mathbf{Z}$ are considered as negatives. However, we want the representation of $[\mathbf{Z}]_{j-1}$ to remain as close to $[\mathbf{Z}]_j$ as $[\mathbf{Z}]_{j+1}$, therefore $[\mathbf{Z}]_{j-1}$ has to be excluded from the list of negatives. The same occurs when calculating the loss around $[\mathbf{Z}]_{j+1}$, where $[\mathbf{Z}]_{j+2}$ has to be excluded from the list of negatives.

Given this, the complete loss for each training batch consists in two elements, the autoencoding loss given by

$$L_{AE} = \frac{1}{(C+R)NT}\sum_{\tilde{\mathbf{Y}}(k)\in\mathbf{B}}\sum_{i=1}^{N}\sum_{j=2}^{T}|[\hat{\mathbf{Y}}]_j^i(k) - [\tilde{\mathbf{Y}}]_j^i(k)|, \tag{2.20}$$

and the NCE loss

$$L_{NCE} = \frac{1}{2(C-1)}\sum_{j=1}^{C-1}\left(l(j,j+1,j-1) + l(j+1,j,j+2)\right), \tag{2.21}$$

which are combined to form the batch loss,

$$L = L_{AE} + \beta L_{NCE}, \tag{2.22}$$

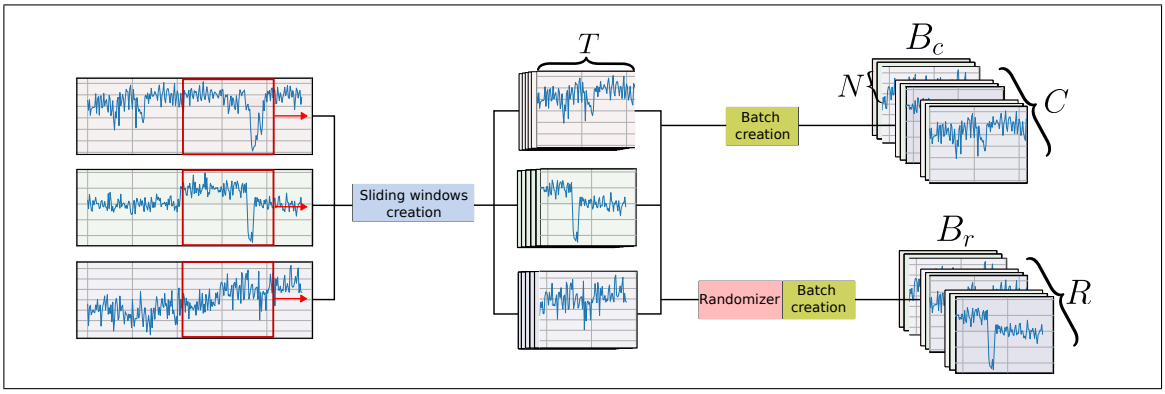


Figure 2.4. Data processing for training the CBDAE. First, each of the N signals to be denoised is split in a sliding window fashion to simulate a real-time processing. Then, each of these univariate windows of depth T are combined to form multiple multivariate windows $\tilde{\mathbf{Y}}(n) \in \mathbb{R}^{N \times T}$ in which the temporal dimension is synchronized. Finally, multiple $\tilde{\mathbf{Y}}$(n) are combined to form two different kind of batches; a batch B_c of consecutive $\tilde{\mathbf{Y}}(n)$ (e.g. $n = 1, 2, 3,$ etc) and a batch B_r of randomly selected $\tilde{\mathbf{Y}}(n)$. Both B_c and B_r are transformed by the CBDAE and their respective representations are used for the calculation of L_{NCE} (see section 4.3.2).

where β is a trade-off parameter between both terms. Finally, network parameters are iteratively optimized during training to minimize the complete loss L, using the back-propagation-through-time (BPTT) algorithm, which is suitable for training recurrent architectures. Then, during operation, parameters are fixed and the network only needs forward passes to estimate the clean version of the signals.

Figs. 2.4 and 2.5 illustrate the roles that all the elements presented in this section play in the training and architecture of the CBDAE. The complete training algorithm of the proposed technique is presented in Algorithm 1.

2.5. Experimental Evaluation

2.5.1. Simulated example

As a proof of concept, we use the CBDAE in a simulated industrial process, the well-known quadruple tank process (Johansson, 2000), which is a nonlinear multi-input

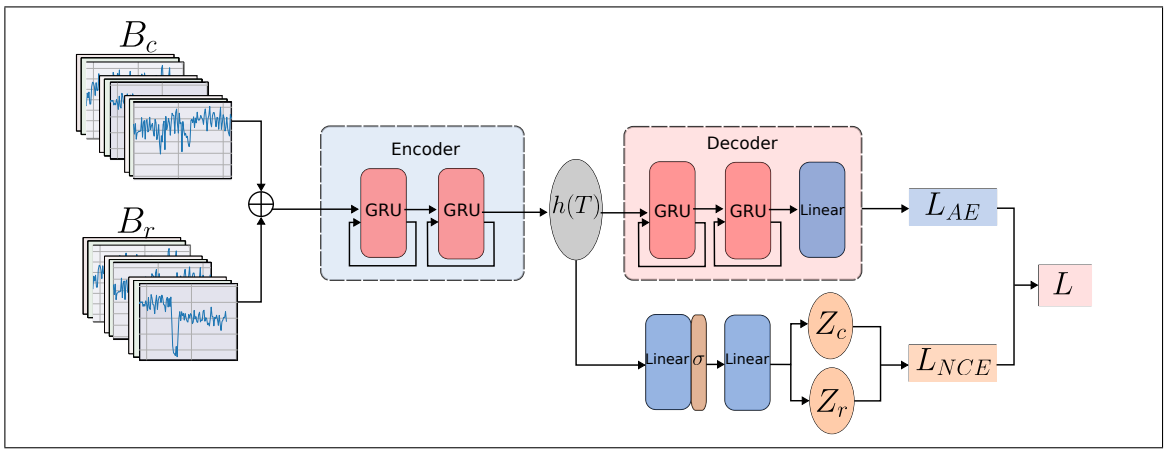


Figure 2.5. Training iteration and proposed network architecture of the CBDAE. Both the encoder and the decoder are composed by two GRU recurrent layers, with the decoder having a final linear projection to produce the outputs from which the reconstruction loss (L_{AE}) is calculated. The batch of latent space vectors $\mathbf{h}(T)$ is transformed by two linear layers to obtain $\mathbf{Z}(T)$, which is then used to calculate the regularization term of the loss L_{NCE}. Finally, both terms are combined to calculate the final loss of the network.

multi-output system that can switch between minimum phase and non-minimum phase behaviour. For all our experiments we used the non-minimum phase configuration to make the dynamics more challenging. Inputs are the voltages applied to the pumps, which vary in the range 0 to 1 volts. We assume that only the real value of the inputs (manipulated variables) are known, i.e., clean signals, which are normally available in any control system. The outputs are the four levels of the tanks that vary between 0 and 50 centimeters and are affected by noise. Both inputs and outputs are sampled to generate $\tilde{\mathbf{Y}}$ and are given to the CBDAE.

To gather data for training, the system is excited with multiple steps, then noise with different characteristics is added to the outputs. Finally, the CBDAE is trained to denoise the signals, following Algorithm 1.

All the experiments were conducted on a 2.4GHz Intel Core i5-9300H machine with 16GB of RAM and an NVIDIA GeForce GTX 1650 graphics card with 4GB of RAM. Also, for all the experiments the following parameters were used: a window depth of

Algorithm 1 Blind Denoising Autoencoder Training

Input: Batch size $C + R$, number of training epochs E, training iterations per epoch W, training database size $\bar{T}$, number of measurements N, window depth T, autoencoder transformations f_{enc}, g, f_{dec} and o_{dec}, scheduled sampling parameters p_d, k_d and c_d, and loss parameter β.
procedure CBDAE TRAINING
    Initialize f_{enc}, g, f_{dec} and o_{dec} weights randomly
    for e in E **do**
        $its = 0$
        while $its \leq W$ **do**
            Sample C consecutive data matrices from the database to form batch $\mathbf{B_c}$
            Randomly sample R data matrices to form $\mathbf{B_r}$
            $\mathbf{B} = \mathbf{B_c} \oplus \mathbf{B_r}$
            Initialize $\mathbf{Z}$ as empty matrix and $\hat{\mathbf{B}}$ as empty sequence, which will be used to hold the autoencoder's latent representations and estimates, respectively
            for each i in $[0 : C + R - 1]$ **do**
                Process $\mathbf{B}(i)$ iteratively with f_{enc} as in (2.11) to obtain $\mathbf{h}(T)$
                Use g to obtain $\mathbf{z}(T)$ from $\mathbf{h}(T)$ as in (2.16) and save this representation in $[\mathbf{Z}]_i$
                Initialize state of the decoder as $\mathbf{d}(1) = \mathbf{h}(T)$ and its first input as the first element of the sequence $[\mathbf{B}(i)]_1$
                for j in $[2 : T]$ **do**
                    Select random number $\epsilon_d \in [0, 1]$
                    SS = True If $\epsilon_d \leq p_d$ Else False
                    if SS = True **then**
                        $\mathbf{d}(j) = f_{dec}([\hat{\mathbf{B}}(i)]_{j-1}, \mathbf{d}(j-1))$ (use past estimates)
                    else
                        $\mathbf{d}(j) = f_{dec}([\mathbf{B}(i)]_{j-1}, \mathbf{d}(j-1))$ (use past target)
                    end if
                    $[\hat{\mathbf{B}}(i)]_j = o_{dec}(\mathbf{d}(j))$
                end for
            end for
            Calculate L_{AE} with all the elements of $\mathbf{B}$ and $\hat{\mathbf{B}}$ as in (2.20)
            Calculate L_{NCE} with elements of $\mathbf{Z}$ as shown in (2.19)
            $L = L_{AE} + \beta L_{NCE}$
            Update f_{enc}, g, f_{dec} and o_{dec} to minimize L using BPTT algorithm.
            $its = its + 1$
        end while
        $p_d = min(1, k_d + c_d e)$
    end for
    Return f_{enc}, g, f_{dec} and o_{dec} that minimize L
end procedure

$T = 60$, batch size $C = 32, R = 32$, two hidden layers in the encoder and the decoder, and a hidden size of 80 in each layer. The additional transformation dimension was selected as $dim(\mathbf{z}(T)) = 20$ and the trade off parameter in the loss $\beta = 1.5$. Adam

optimizer was used for optimization and Pytorch as the deep learning framework. Finally, the $RMSE$ between the original (clean) signal and the output of the different models is used as performance indicator for evaluation.

To evaluate blind denoising, the output signals are corrupted using a combination of white and impulsive noise, which is typically seen in real industrial processes. The noise power is varied from moderate ($\sigma = 0.5$) to very strong noise ($\sigma = 4$). To illustrate the advantages of the proposed CBDAE, we compare it against multiple classical data-driven and model-based baselines typically used in the industry, as well as against other BDAE networks trained with different variations of Algorithm 1, this to highlight the positive effects of using the NCE loss.

As for the classical data-driven baselines, a Savitsky-Golay filter with a polynomial order of 2, an EMA filter with $\alpha = 0.33$ and typical mean and median filters were tested with different window sizes. For the classical model-based techniques, a Kalman Filter, a Particle Filter, and an Extended Kalman filter, all using the real process matrices, were implemented as well. Finally, in the case of the different BDAE variations, we used a recurrent AE without any type of regularization (BDAE$_{NoReg}$) but trained as described in subsection 4.3.1. A recurrent AE with L_1 regularization instead of NCE regularization (BDAE$_{L1}$), which has been found to be useful for denoising (Thanh et al., 2020), with $\beta_{L1} = 1 \times 10^{-6}$ (the weight of the L1 regularization term) and a CBDAE without the transformation g (CBDAE$_h$), meaning that in this case NCE regularization is applied directly on $\mathbf{h}(T)$.

Table 2.1 shows the denoising results in terms of RMSE for the different methods. In the table, it can be seen that all BDAE methods clearly outperform both classical data-driven and model-based denoisers. In the case of data-driven denoisers, surprisingly the Savitsky Golay filter had the worst results. This is most likely due to the high frequency preservation property of this filter, which is specially harmful when the input signal is corrupted with "salt and pepper" noise. As for model-based denoisers, their poor performance may be due to the fact that these type of filters have major difficulties when faced

Table 2.1. Comparison of denoising results in terms of the average RMSE [cm] for all the tank levels. For classical data-driven techniques w_i represents the length of the window that delivered the best results. For each noise power, the best result is highlighted in bold.

Method / Noise Power	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 1.5$	$\sigma = 2$	$\sigma = 2.5$	$\sigma = 3$	$\sigma = 3.5$	$\sigma = 4$
Original Noisy input	2.165	2.406	2.572	2.850	3.121	3.515	3.929	4.277
Mean F. (w_5)	1.254	1.269	1.381	1.475	1.634	1.777	1.911	2.115
Median F. (w_5)	0.803	0.917	1.117	1.337	1.583	1.785	2.019	2.173
Savitsky Golay F. (w_{30})	1.303	1.313	1.473	1.560	1.757	1.907	2.04	2.207
EMA F.	1.244	1.264	1.375	1.469	1.630	1.773	1.913	2.089
Kalman F.	2.005	2.108	2.176	2.259	2.324	2.445	2.548	2.581
Particle F.	1.708	1.774	1.986	2.088	2.334	2.560	2.797	3.026
Extended Kalman F.	1.773	1.846	1.956	2.041	2.187	2.342	2.497	2.603
BDAE$_{NoReg}$	0.671	0.502	0.577	0.638	0.663	0.647	0.779	0.641
BDAE$_{L1}$	0.668	0.572	0.58	0.587	0.588	0.668	0.566	0.620
CBDAE$_h$	0.392	**0.313**	**0.326**	0.445	0.521	0.569	0.741	0.602
CBDAE	**0.276**	0.318	0.355	**0.394**	**0.407**	**0.518**	**0.489**	**0.542**

with a non-minimum phase system (Ansari & Bernstein, 2019). Regarding the different BDAE methods, it can be seen that the CBDAE$_h$ and the CBDAE, the two methods that use NCE regularization, achieve the best results in all cases. It is interesting to note that the CBDAE clearly outperforms the CBDAE$_h$ for high values of σ, which is probably due to transformation g which helps the CBDAE to better preserve the dynamical information of the input signals in another set of vectors different from those used for decoding.

Fig. 2.6 shows the denoising results of the CBDAE when applied to denoise one of the quadruple tank signals. It can be seen that indeed the CBDAE is able to successfully recover the clean version of the signal, as suggested by the numerical results presented in Table 2.1. Additionally, to explore the feasibility of implementing the CBDAE in real time, we analyze its computational complexity when denoising a multivariate sequence using the hardware detailed above. Table 2.2 shows the execution time of the CBDAE and other classical methods. It can be seen that classical filters have considerably lower execution times than the CBDAE, which is expected due to their simplicity; however, model-based nonlinear techniques present higher execution times, a difference that becomes more significant when using a GPU accelerator. The execution times of other neural-based denoisers are omitted since they are very similar to the ones of the CBDAE. The results in

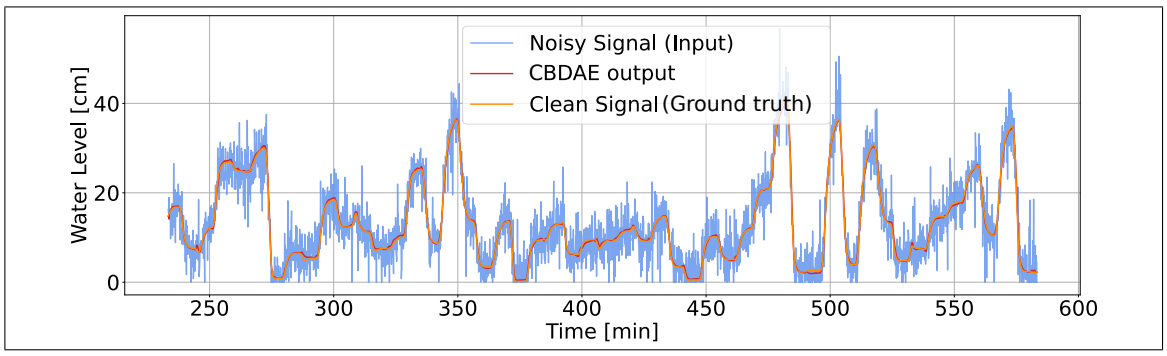


Figure 2.6. CBDAE results when blind denoising one of the quadruple tank sensors corrupted with white and salt and pepper noise with $\sigma = 3$.

Table 2.2. Execution time of different denoising methods on the specified hardware.

Method	Time $[ms]$
Mean F. (w_5)	0.36
Median F. (w_5)	0.42
Savitsky Golay F. (w_{30})	0.37
EMA F.	0.21
Kalman F.	5.42
Particle F.	167.58
Extended Kalman F.	281.64
CBDAE (CPU)	23.15
CBDAE (GPU)	10.82

Table 2.2 indicate that the CBDAE is fast enough for real time implementations and that it can be implemented in modern edge devices as part of a control system pipeline.

Finally, Fig. 2.7 shows the latent space trajectories obtained for the different BDAE networks when two different but similar trajectories are given for denoising, after using PCA as dimensional reduction technique to project the latent space to two dimensions. It can be seen in the figure that the CBDAE latent space is much more ordered, smooth and predictable than the latent space of the other BDAE networks. It is also interesting to note that the principal components of the CBDAE latent space present two lobes that match with the first-order information of the input signals. When the tanks are being filled, the latent space trajectory is on the right-side lobe, and when the tanks are being emptied, the latent space trajectory is on the left-side lobe. This dynamically smooth and ordered

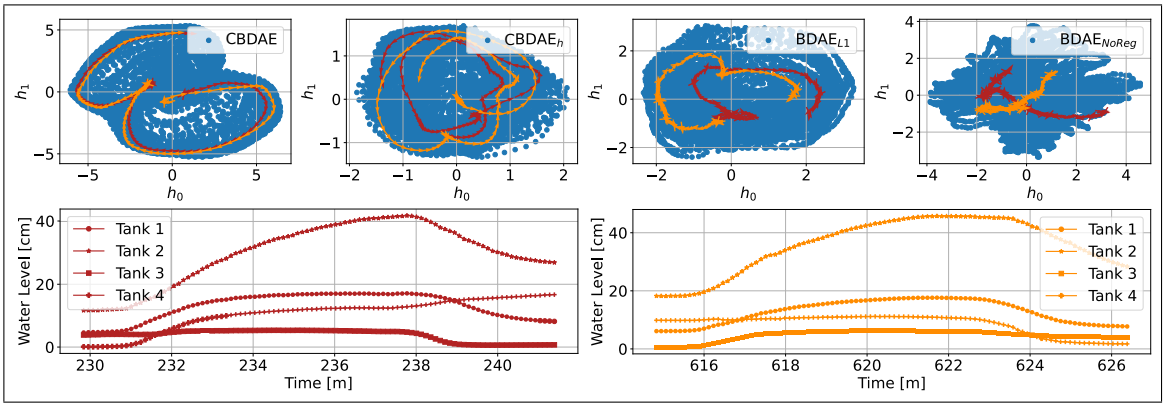


Figure 2.7. Two-dimensional latent space trajectories obtained with PCA for the different BDAE networks when the process follows two similar trajectories for $\sigma = 3$. In the upper graphs, the red line represents the latent trajectory when the tank levels follow the lower left-hand graph trajectories and the orange line in the upper graphs represents the same for the lower right-hand graph.

structure acquired by the latent space, as a result of using NCE regularization, could be further exploited for other downstream tasks as prediction and fault detection.

2.5.2. Application to an industrial paste thickener

As a second experiment, the CBDAE was applied to denoise the signals of a real industrial paste thickener. This process is considerably more challenging than the quadruple tank process, since the thickener is subject to many unmeasured disturbances and we only have access to a reduced number of measurements. Is worth mentioning that since in this experiment we are working with real process data, we do not have access to the clean version of the signals and therefore, we cannot evaluate the results of the CBDAE quantitatively. A detailed description of the IIoT infrastructure used to generate the data of the industrial paste thickener can be found in (Núñez et al., 2020).

2.5.2.1. Thickening process description

Thickening is the primary method in mineral processing for producing high density tailings slurries. Thickening generally involves a large tank (see Fig. 2.8) with a slow

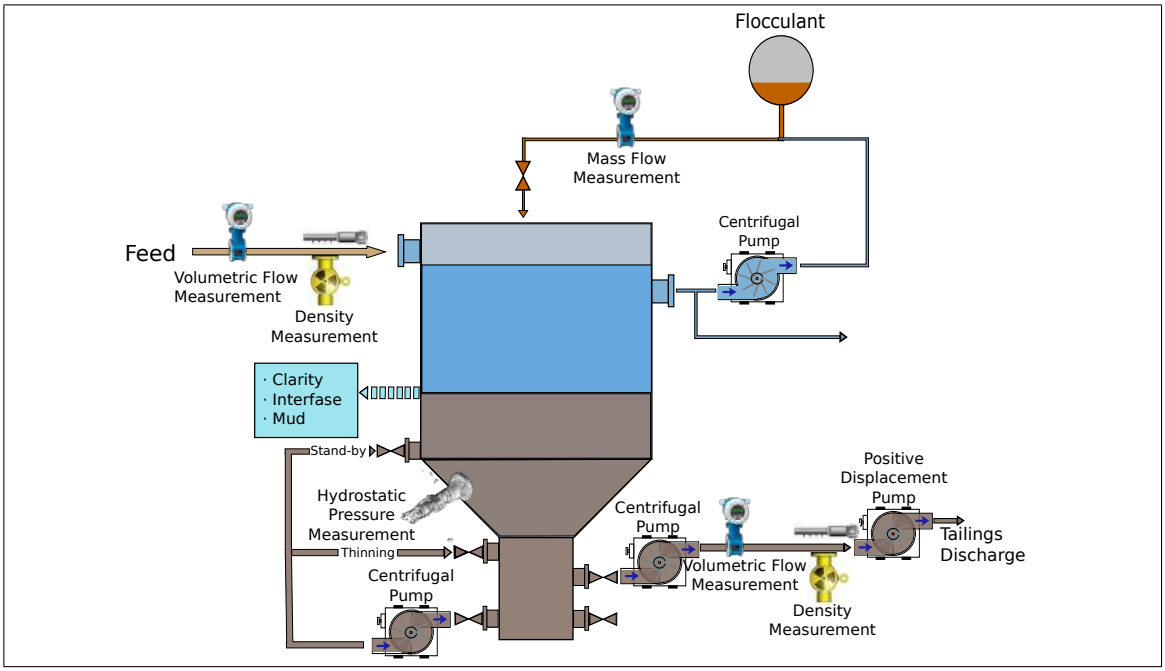


Figure 2.8. Process and instrumentation diagram of the paste thickener used for the experiments. Time series are available for the following variables: feeding and discharge rates, flocculant addition rate, feeding and discharge density, internal states of the thickener (mud, interface and clarity levels).

turning raking system. Typically, the tailings slurry is added to the tank after the ore extraction process, along with a sedimentation-promoting polymer known as flocculant, which increases the sedimentation rate to produce thickened material discharged as underflow (Núñez et al., 2020).

Due to its complexity and highly non-linear dynamics, deriving a first-principles-based mathematical model is very challenging. Therefore, an appealing approach is to use data-driven modeling techniques. However, sensors in charge of providing data are exposed to strong disturbances and noise and an effective online preprocessing technique is needed. Hence, the thickener is an interesting real process to test the proposed CBDAE.

The CBDAE was trained with 12 months of real operational data from the industrial thickener, and following the guidelines of (Núñez et al., 2020), we selected 8 key variables

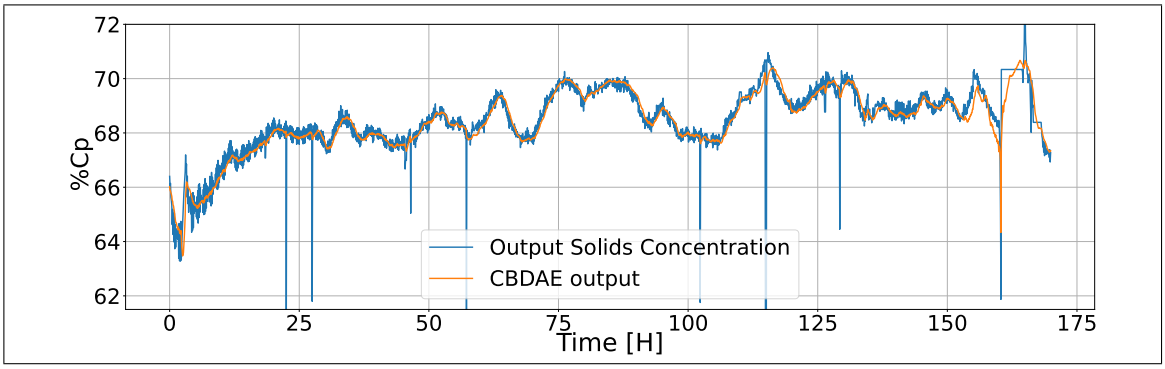


Figure 2.9. CBDAE results when blind denoising the output solids concentration, one of the key variables of the thickener for control purposes.

that have been used to model this process with data-driven techniques. Namely, the flocculant flow, the output flow (manipulated variables), the input solids concentration, input flow (measured disturbances), the bed level, the rake torque, the hydrostatic pressure and output solids concentration (process outputs).

Figs. 2.9 and 2.10 show the denoising results of the CBDAE for the output solids concentration and the input solids concentration, which are two of the most important signals for control purposes. It can be seen in Fig. 2.9 that the CBDAE successfully learns to ignore the spikes in the signal, which clearly do not belong to the dynamics of the system. Similarly, in Fig. 2.10, it can be seen that the CBDAE is capable of filtering the strong noise affecting the input solids concentration. It is also interesting to note that the resulting delay is minimum, which could be critically important for control applications built in top. Analogous to Fig. 2.7, Fig. 2.11 shows the trajectories in the latent space for two similar trajectories in the input space. Note that both the CBDAE and the CBDAE$_h$ show smooth and similar trajectories, unlike the networks that do not use NCE regularization, which show an erratic and totally unpredictable behavior.

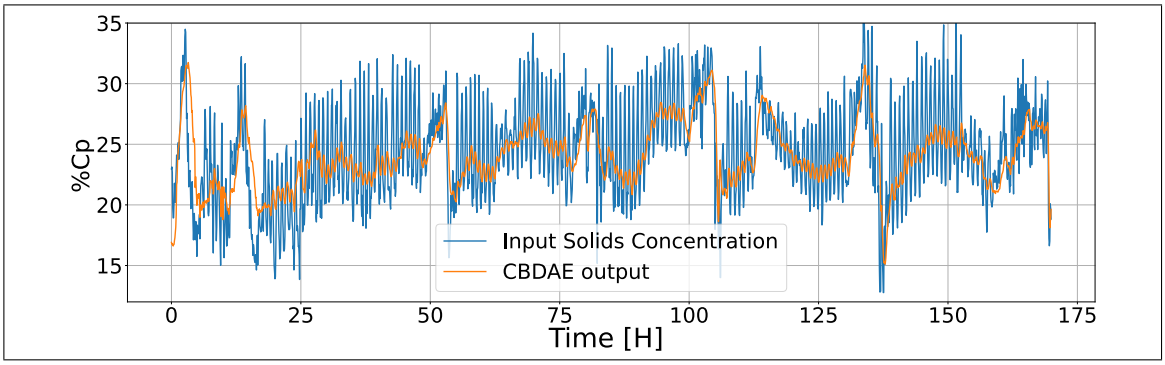


Figure 2.10. CBDAE results when blind denoising the input solids concentration, a measured disturbance with one of the most corrupted sensors.

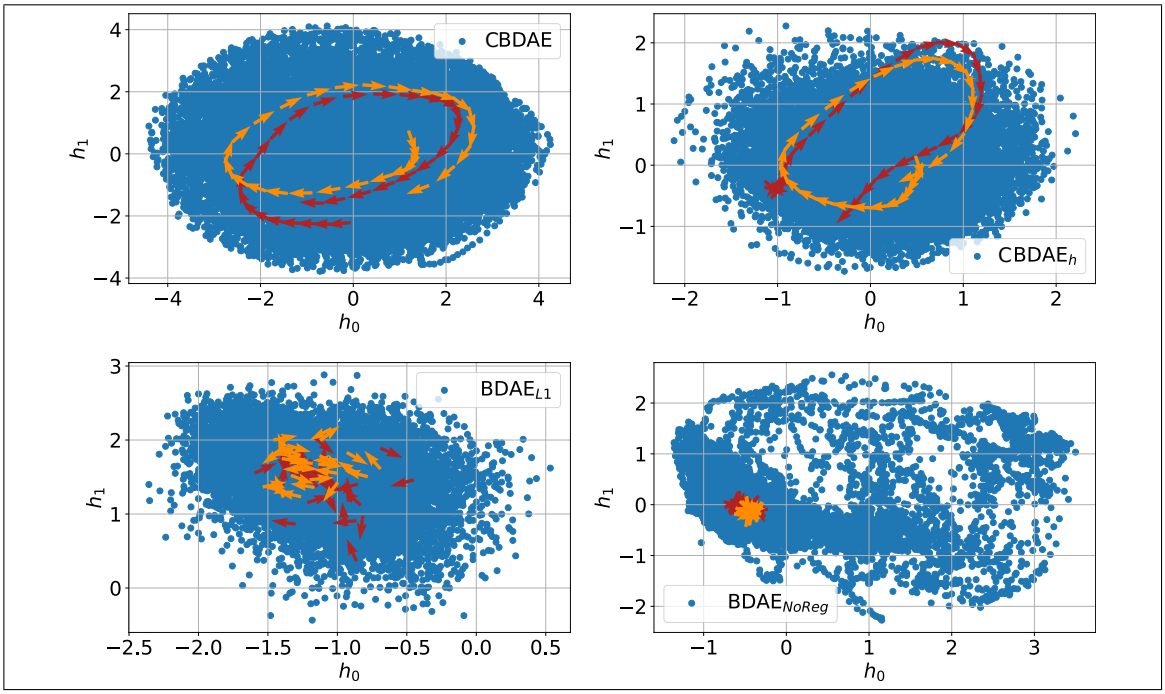


Figure 2.11. Two-dimensional latent space trajectories obtained with PCA for the different BDAE networks when the thickening process follows two similar trajectories.

2.6. Discussion

In this chapter, an intelligent general purpose data-cleaning method to be used in the lower layers of a CPS is proposed. The two experiments presented here showed the effectiveness of the CBDAE to improve data quality of multivariate time series by exploiting

the temporal smoothness and cross correlation of different variables of physical systems. Also, it was shown how the NCE regularization transferred some of the temporal smoothness of the input signals to the low-dimensional latent space of the CBDAE, which could be exploited for other downstream tasks.

Results showed the superiority of the CBDAE when compared to classical techniques commonly used in industry and other neural network approaches. In future work the CBDAE will be tested on an edge device for real time data cleaning of a real process, and the latent space of the network will be further investigated for its use in other downstream applications.

3. A META-LEARNING APPROACH TO PERSONALIZED BLOOD GLUCOSE PREDICTION IN TYPE 1 DIABETES (T1D)

This Chapter focuses on the design of an adaptive model in the context of model personalization for real-time glucose prediction in patients with T1D. Additionally, a probabilistic extension that targets glucose uncertainty estimation under meal specification errors on part of the patient, is proposed [1].

3.1. Context

Type 1 diabetes is a chronic autoimmune disease characterized by an increased blood glucose concentration due to a deficiency in the secretion of insulin, a hormone produced by pancreatic beta cells (Kahanovitz et al., 2017). Insulin promotes glycogenesis (a process that synthesizes glycogen from circulating glucose to be stored in the liver and muscle cells) and also inhibits the secretion of glucagon, a hormone that increases the concentration of glucose in the bloodstream by converting the stored glycogen back into glucose, by alpha cells (Vergari et al., 2019).

Insulin and glucagon are part of a larger system, called glucoregulatory hormones, which are responsible for keeping glucose levels in a relatively narrow range. If glucose in the bloodstream decreases below a certain threshold (e.g., in a fasting state or prolonged intense exercise), alpha cells secrete glucagon, increasing glucose concentration. On the other hand, if the concentration rises above the normal level, due to intestinal absorption of food or other causes, beta cells produce insulin, decreasing glycemia. This regulation process is called glycemic homeostasis. However, in people with T1D, the immune system destroys the beta cells and therefore, the body loses the capacity for homeostasis.

T1D patients can regulate their glucose by constantly injecting a proper amount of insulin; however, doing it manually is difficult, and if not done correctly, it can have several negative consequences. On the one hand, high blood glucose levels ($>$ 180 mg/dL) can

[1] Part of the material in this chapter has been accepted for publication in the journal Control Engineering Practice (Langarica et al., 2023)

lead to long-term complications such as poor blood circulation, cardiovascular problems and blindness among others, and on the other hand, low glucose levels (< 70 mg/dL) can lead to severe short-term complications such as dizziness, fainting or even death (De Bois et al., 2022).

Thanks to the introduction of continuous glucose monitoring devices (CGMs) and insulin pumps, it is now is possible to rely on automatic glucose management systems, also called artificial pancreas (AP) systems, which alleviate patients from the burden of having to control their glucose levels manually. Numerous clinical trials have validated these systems, showing that automatic control techniques have the potential to improve life conditions of people that suffer from T1D (Brown et al., 2019; Pinsker et al., 2022; Carlson et al., 2021).

Among the different control approaches that have been used in this context, model predictive control (MPC) is one of the most promising, with excellent results both *in vivo* and *in silico* (Shi et al., 2019; Gondhalekar et al., 2018). The MPC approach relies on a dynamical model of glucose behaviour to find, via an optimization problem, the amount of insulin that, when delivered, results in the desired level of glucose. Therefore, the more accurate the model is, the better closed-loop performance is achieved. In fact, as pointed out in a recent review on modern MPC applications (Schwenzer et al., 2021), model accuracy is one of the major factors that can limit the effectiveness of MPC approaches. In the context of T1D, due to the nonlinear, time-varying and subject-specific behaviour of glucose in the human body (Herrero et al., 2017), obtaining a sufficiently accurate model that can be used for control purposes is extremely difficult, especially when deriving it from first principles. Consequently, most of current MPC approaches for glucose control use simple linear models that allow only a minor degree of personalization (van Heusden et al., 2012) and try to compensate modeling errors with the receding horizon characteristic of MPC and by adding task-specific constraints to the optimization problem (Laguna Sanz et al., 2017).

Enabled by the significant improvements in hardware technologies, many application fields are moving towards generating more complex and accurate models using data-driven approaches, despite the fact that more complex models generally imply a higher computational burden in the optimization step. In this context, several works have attempted to generate more accurate models for glucose prediction by leveraging the increasing amount of data obtained from AP clinical trials or from the FDA-accepted UVA/PADOVA simulator (Man et al., 2014). These efforts include classical data-driven techniques as AR, ARX or ARMAX models (Turksoy et al., 2013), and machine learning (ML) models as multilayer perceptrons (MLP) (Pérez-Gandía et al., 2010), support vector regression (SVR) (Reymann et al., 2016) and random forest (RF) (Georga et al., 2012) (see (Woldaregay et al., 2019) for a thorough review of ML techniques for glucose prediction). Nevertheless, the performance of most of these techniques is limited, since they impose a fixed structure to the model, instead of letting the model decide its own structure throughout the training process.

Recently, deep learning approaches have emerged as a more flexible and better performing alternative to classical ML techniques in many fields, including glucose prediction (Zhu et al., 2021). However, these techniques require large amounts of data in order to perform well (Sun et al., 2017), which limits its range of action to population models that, indeed, have shown very promising results (Li et al., 2020; Mohebbi et al., 2020; Sun et al., 2018); However, obtaining a personalized model is infeasible since a large enough amount of data of a particular patient is very hard to obtain. This issue is a serious one given that the large inter-subject variability that characterizes glucose behaviour in T1D patients leads to population models that may not be safe nor accurate enough to develop a reliable MPC scheme (van Heusden et al., 2012).

To address this issue, in this chapter we propose a meta-learning-based approach (Hospedales et al., 2020) for fitting accurate individualized glucose prediction models along with a probabilistic extension, to also target uncertainty estimation in the presence of meal specification errors on the part of the patient.

Our proposed model is able to learn relevant cross-subject features from population data and then adapt a subset of its parameters to fit a new patient, with very few data, few training iterations, and with low risk of over-fitting. Also, we show that our model, achieves better performance on the FDA-accepted UVA/Padova simulator (Man et al., 2014) than state-of-the-art population models when faced to new patients never seen during training, even when they are fine-tuned to a particular patient using transfer-learning, an approach recently proposed to develop personalized models for glucose prediction (Seo et al., 2021). Additionally, when considering the probabilistic extension, we show how our model is able to correctly estimate the uncertainty produced by meal specification errors on part of the patient, achieving better performance on several probabilistic metrics when compared to other neural network approaches and a Gaussian Process model, a very strong baseline in probabilistic modeling.

Accordingly, the main contributions in this chapter are three-fold: i) we propose a novel framework to use meta-learning for developing personalized models for glucose prediction. To the authors' knowledge, this is the first work that uses meta-learning for this task, transferring some of the recent success of this approach (Hospedales et al., 2020) to the T1D research field; ii) we propose a probabilistic model based on Recurrent Kalman Networks to estimate the uncertainty produced by meal specification errors on part of the patient, a very common problem in T1D modeling and control; and iii) we propose a new out-of-distribution evaluation strategy to compare prediction models when different knowledge about patient data is available, and we present a thorough comparison of different data-driven approaches using this strategy.

3.2. Related work

The importance of personalized models have been widely recognized in the AP research community (see, e.g., (Oviedo et al., 2016)). Indeed, most of the successful MPC implementations have demonstrated that personalized models help in achieving better and safer control policies (van Heusden et al., 2012; Haidar, 2016). In this sense, DL is an

appealing alternative due to its recent success in many fields as a modeling tool, and by the fact that is able to learn from any kind of data, as patient-specific data. However, this tool has not been widely exploited for this purpose.

The lack of DL-based efforts is due to the fact that is very difficult to obtain large enough amounts of data of a particular patient to fit an accurate DL model (Daniels et al., 2022b). If the amount of data is insufficient there is a high risk of over-fitting, which causes a poor generalization and, consequently, results in poor performance. Nonetheless, recently, some techniques from areas such as text and image processing have been used for model personalization.

For example, (Seo et al., 2021) proposes the use of transfer-learning for fitting personalized models, i.e., using the weights of a pre-trained population model and then fine-tune them on patient-specific data. The authors use a large data-set of $n = 114$ patients with T1D to train a CNN. The fine-tuned model was compared against a population model, trained using CGM readings of 85 patients, and a network trained from scratch using only patient specific data. Results show that the fine-tuned network is (slightly) better than the other alternatives.

In the same line, (Mohebbi et al., 2020) presents a similar approach based on a Long-short-term-memory (LSTM) network. The data-set consists in 14 consecutive days of CGM readings from $n = 50$ real patients in free-living conditions. In this case the population model, trained using 35 patients, shows better performance than the fine-tuned model using transfer-learning. The authors claim that the lower performance of the fine-tuned network is probably due to the small amount of training data used in the fine-tuning stage, even though they use half of the data from each patient for this purpose.

A different approach is presented in (Armandpour et al., 2021), where the authors propose the use of an embedding layer in conjunction with an encoder-decoder network with attention mechanism to learn embedding vectors that encode patient-specific information. During testing, the embedding vectors are indexed to inform the network which patient

the predicted signals belong to. Although the model shows promising results in a database of $n = 38$ T1D patients, the authors do not compare their approach to any other DL population or personalized model. Furthermore, since the patient-specific vectors are learned during training and then are kept fixed, this approach cannot be extended to new patients unless the whole network is trained again.

Finally, and more related to our proposed approach, in (Daniels et al., 2022b) the authors propose a multitask network consisting in a CNN combined with an LSTM network and multiple patient-specific feed-forward layers. The CNN and the LSTM network are shared among all patients to learn cross-subject features, while patient-specific layers learn features relevant for a particular subject. The model shows better results than a fine-tuned population model and a model trained using only patient-specific data; however, since patient-specific layers are jointly trained with cross-subject layers, this model cannot be extended to new patients unless the networks are trained again.

3.3. Meta-Learning for personalized models in glucose prediction

3.3.1. Background on Meta-Learning

Meta-learning is a long-standing problem of interest in ML, which consists in enabling artificial agents to efficiently learn new tasks by *learning to learn*. These algorithms leverage data from previous learned tasks and use this knowledge to quickly adapt to new tasks that are assumed to belong to the same task distribution, also called *meta-distribution* (Raghu et al., 2019). A common example to illustrate the concept and power of meta-learning, is the sine waves regression problem (Finn et al., 2017). This problem consists in training the network with multiple sine waves with different amplitudes and phases (different tasks or child distributions) and then, testing the network in regressing a new sine wave by showing it only few points. In this case, instead of teaching the network to regress a particular sine wave (child distribution), as done in typical supervised-learning

approaches, the network is trained to learn the underlying sine form (meta-distribution) and then to particularize to an instance with very few data.

Among the different meta-learning algorithms, the *model agnostic meta-learning* (MAML) (Finn et al., 2017) has been highly successful in different applications, including modeling of dynamical systems with time-varying characteristics (Clavera et al., 2018) and few-shot classification (Finn et al., 2017). At a high level, the MAML algorithm finds a suitable initialization for a NN so that new tasks can be learned with very few samples and without over-fitting. This is done via two optimization loops, namely, i) an outer loop that updates the initialization of the network; and ii) an inner loop that performs adaptation or fine-tuning to new tasks using k gradient updates.

Formally, given a network f_θ with learnable parameters θ and a partially observable task distribution $\rho(\mathbf{T})$, let $\{\mathbf{T}_1, \cdots, \mathbf{T}_B\}$ be a batch of B tasks, where for each task $\mathbf{T}_n$ the data is divided into a support set S_{T_n}, which is used for the inner loop updates, and a target set R_{T_n}, which is used for the outer loop updates. Now, let θ_i^n be the parameters of the network after i gradient updates for task $\mathbf{T}_n$, and let $\theta_0^n = \theta$. Each update in the inner loop follows

$$\theta_k^n = \theta_{k-1}^n - \alpha \nabla_{\theta_{k-1}^n} L_{S_{T_n}}(f_{\theta_{k-1}^n}), \tag{3.1}$$

where $L_{S_{T_n}}(f_{\theta_{k-1}^n})$ is the loss on the support set of $\mathbf{T}_n$ after $k-1$ inner loop updates and α is the inner-loop learning rate.

As for the outer loop, the meta-loss is given by

$$L_{meta}(\theta) = \sum_{n=1}^{B} L_{R_{T_n}}(f_{\theta_k^n}), \tag{3.2}$$

where $L_{R_{T_n}}(f_{\theta_k^n})$ is the loss on the target set of $\mathbf{T}_n$ after the k inner-loop updates using S_{T_n}. The outer loop updates follow

$$\theta = \theta - \beta \nabla_\theta L_{meta}(\theta), \tag{3.3}$$

where β is the outer-loop learning rate.

In this work, we use the variation of the algorithm proposed in (Raghu et al., 2019). This modification consists in using only the "head of the network" (the last few layers) during the inner loop updates, instead of the whole network as it was originally proposed. In (Raghu et al., 2019), it was shown that with this modifications, similar results can be obtained at a fraction of the computational cost.

3.3.2. Meta-Learning for glucose prediction

In our proposed approach, a key observation is that the underlying dynamical system of glucose behaviour is basically the same for all T1D patients (Man et al., 2014). Hence, a natural application of MAML in the context of personalized models for glucose prediction, is to consider each patient-specific measurements as one task or child distribution drawn from the meta-distribution that governs glucose behavior, and then split the patient data into a support set S_{T_n} and a target set R_{T_n}, for the inner and outer optimization loops, respectively.

However, typically, T1D data-sets contain only a limited number of patients; hence, this naive approach will result in few tasks and, consequently, a poor generalization and poor estimation of the meta distribution $\rho(\mathbf{T})$ (Clavera et al., 2018).

To avoid this issue, in this work we propose to define a task as a pair of randomly chosen meal events of the same patient. To this end, a meal event is defined as

$$\mathbf{ME}_n^i = \mathbf{Y}_{n_{[t^i_{meal_n}-\Delta_i : t^i_{meal_n}-\Delta_i+D]}}, \in \mathbb{R}^{D\times m} \tag{3.4}$$

where $\mathbf{ME}_n^i$ represents the i-th meal event of patient n, $\mathbf{Y}_n$ represents the measurements of patient n, which include CGM measurements, and (possibly) insulin controls and announced meals (m signals). $t^i_{meal_n}$ is the estimated or user-reported time of meal i, Δ_i is the time difference between the start of the meal event and the reported meal time $t^i_{meal_n}$, and D is the duration of the event. In this work we choose Δ_i as a random number drawn from a uniform distribution $U(0.5, 3.5)[h]$ and $D = 8[h]$ to cover the entire response to

the meal, even when other meals may overlap. Consequently, each task is defined as

$$\mathbf{T}_n^{ij} = [\mathbf{ME}_n^i; \mathbf{ME}_n^j], \tag{3.5}$$

where $\mathbf{ME}_n^i$ and $\mathbf{ME}_n^j$ are taken as the support and target set respectively.

This definition of a task is motivated by several considerations. First, using pairs of meal events from a patient instead of the whole sequence of measurements, allows us to increase the number of tasks considerably, and also provides a natural way to perform *task augmentation* during training, by shuffling the support and target sets of meal events within and between them. Furthermore, when shuffling the patient-specific support and target meal events, we force the network not to associate any particular pair of meals with one another, but to extract the characteristic features of the underlying patient's glucose behaviour from the support meal event to predict the randomly chosen target meal event. This is further illustrated in Section IV, where we show that even when using only one training meal event, there is a general performance improvement in predicting future glucose for a patient not seen during training.

Another reason why this task definition is advantageous is that a meal event captures the most relevant and rich information of glucose dynamics, as the system is excited by an external input, leaving out less relevant information as quasi-steady state information that could reduce the effectiveness of the learning process. A natural extension of these ideas could be to define every out-of-range excursion as an event and pair it with another one to form a task, however, this is left as future work.

A complete scheme of our proposed meta-learning-based training procedure is shown in figure 3.1.

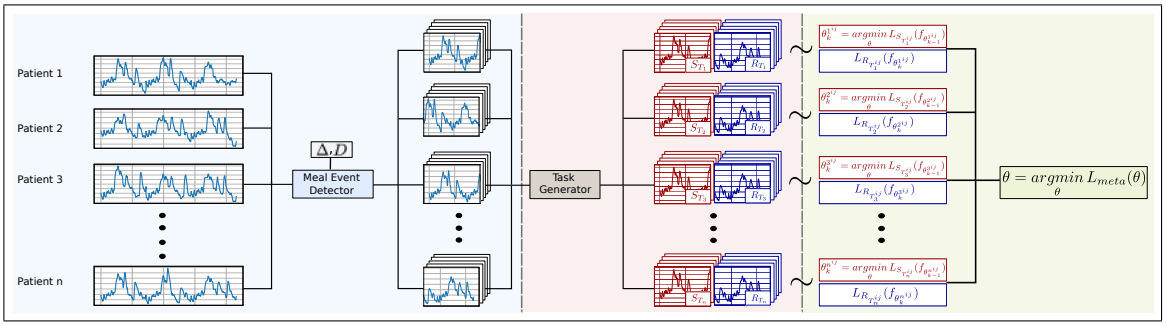


Figure 3.1. Complete schematic of the proposed approach. First, the meal event detector separates every meal event in each patient time-series, following equation 3.4. This is done without mixing data from different patients. Then, the task generator pairs two randomly chosen meal events from a particular patient to form a task $\mathbf{T}_n^{ij}$, following equation 3.5. Finally, at each iteration, a number of tasks coming from each patient are sampled to form a batch of tasks. Each element of the batch is then used for the inner loop updates (see equation 3.1) and for the outer loop updates by computing $L_{R_{T_{n^{ij}}}}(f_{\theta_k^{n^{ij}}})$ for each patient (see equation 3.3). The light blue background represents the processes that are executed only once during the training procedure, the light red background the processes executed at every epoch, and the light green background, the processes executed at every iteration.

3.3.3. Proposed architecture

To address the glucose prediction problem, we propose the network architecture depicted in figure 3.2, which is divided in two parts, namely, the *Base* and the *Head*. Following the approach presented in (Raghu et al., 2019), the parameters of the *Base*, consisting of two feed-forward layers and a Kronecker LSTM (Jose et al., 2017), are learned during training and then are kept fixed during testing, thus being shared by all the patients. This part of the network is responsible for learning cross-subject features that explain the underlying dynamical system, i.e., the blood glucose mechanics. On the other hand, the *Head*, consisting of three feed-forward layers, is trained to learn patient-specific features by adapting its parameters using a support meal event in k training iterations, following equation 3.1. Consequently, the parameters of the *Head* are able to adapt, with few data, to an arbitrary number of patients, even if they were not seen during training.

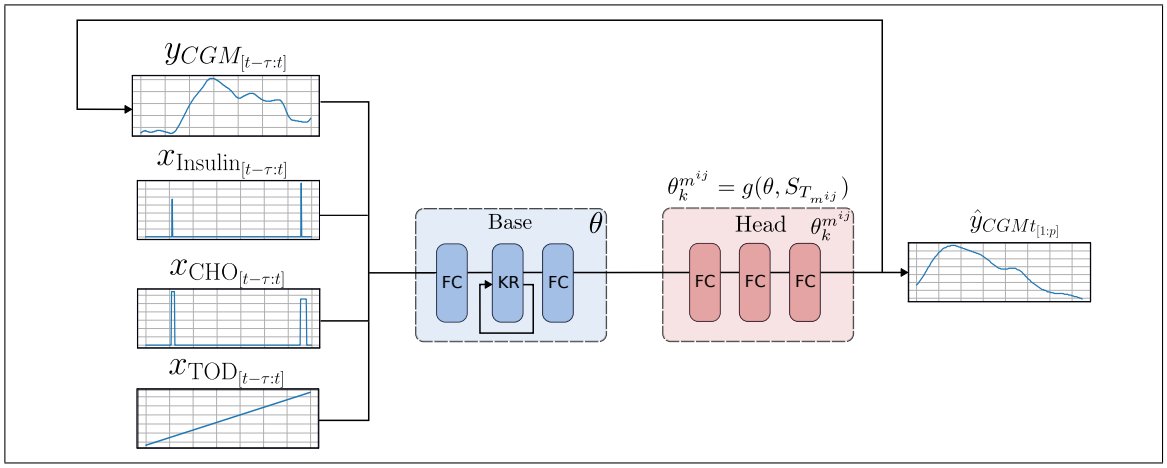


Figure 3.2. Proposed network architecture. The *Base* includes the parameters that are learned during training and then kept fixed during testing. Here, the Kronecker LSTM layer allows to account for the temporal relations in the input data but with a fraction of the parameters of a typical LSTM. The *Head* contains the parameters of the network that adapt to an arbitrary number of different patients, under the assumption that all come from the same task distribution $\rho(\mathbf{T})$. Four sliding windows of length τ are used as inputs to the network, namely, past CGM measures ($\mathbf{y}_{CGM[t-\tau:t]}$), past insulin ($\mathbf{x}_{Insulin[t-\tau:t]}$), past ingested carbohydrates ($\mathbf{x}_{CHO[t-\tau:t]}$) and the time of the day ($\mathbf{x}_{TOD[t-\tau:t]}$). The objective is to produce one-step ahead predictions, which are repeatedly fed-back as inputs in future steps to produce a sequence of predictions of length p ($\hat{\mathbf{y}}_{CGM_t[1:p]}$).

The inputs of the network are sliding windows of length τ containing past information, which in our case are four signals usually available in T1D studies: delivered insulin, measured by an insulin pump; the carbohydrates content of the meal, which should be entered manually by the patient; the blood glucose concentration, which is measured by a CGM sensor; and the time of day, which is added to help the network infer the periodic variability of insulin sensitivity (Toffanin et al., 2013). At each step, the network performs a one step ahead prediction, which is then appended to the corresponding sliding window to predict the next value. In the case of the exogenous inputs, during training, at each step the next real value of the signal is appended to its corresponding sliding window, instead of keeping them at a constant value. This prevents the network to become auto-regressive and unresponsive to the inputs and, in case the network is used in an MPC scheme, allows

simulating different future scenarios. This process is repeated until a sequence of p predictions is produced. Finally, given the complete sequence of predictions, the training loss is calculated against the CGM measurements as

$$L(t) = \frac{1}{p}\sum_{n=1}^{p}(\mathbf{y}_{CGM}(t+n) - \hat{\mathbf{y}}_{CGM_t}(n))^2, \tag{3.6}$$

where $\mathbf{y}_{CGM}(t+n)$ is the CGM measurement at time $t+n$ and $\hat{\mathbf{y}}_{CGM_t}(n)$ is the n-th value of the predicted sequence considering past measurements up to time t.

Unlike other approaches, we train the network to generate a sequence of future predictions for a particular horizon, instead of just predicting one value into the future, since we want the network to be useful for control purposes. However, for the sake of comparison with other works, both average metrics of the predicted sequence as well as point metrics for different horizons are presented in the evaluation section.

3.3.4. Proposed Out-Of-Distribution testing procedure

The practical value of any ML or DL approach is related to its ability to generalize beyond its training set, specially in safety critical applications (Berend et al., 2020). In this context, out-of-distribution (OOD) testing is an increasingly popular method for evaluating a model's ability to generalize (Teney et al., 2020; Setlur et al., 2021). OOD testing consists in training a model with data drawn from a source distribution and then, testing it with data drawn from a different distribution that shares some semantic similarities with the source distribution (i.e., two child distributions drawn from the same meta-distribution). For example, training a model to classify pictures, and then testing it on drawings. Models that are optimized with OOD testing are particularly robust to distributional shifts (Berend et al., 2020), robustness, that we argue, is particularly useful for model personalization in the context of T1D.

Even though OOD testing is a natural way for assessing how well a meta-learning model has learned the meta-distribution of the data given only partial information, there

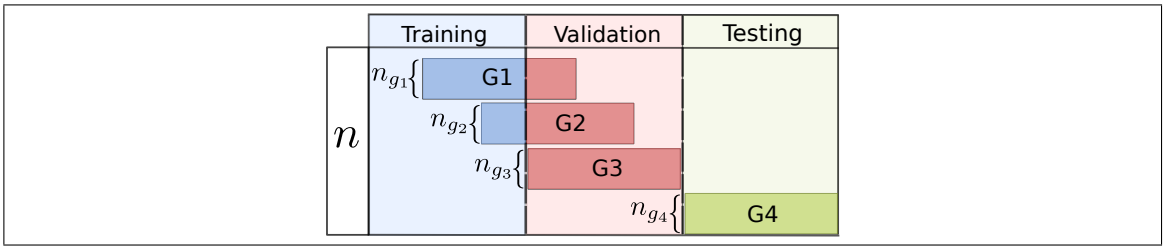


Figure 3.3. Proposed Out-Of-Distribution data-split for glucose prediction. A data-set with n patients, which represents four different distributions, is divided in four different groups. Each group has a different degree of membership to the train, validation and test set, so that the model can be evaluated with different degrees of distributional shifts.

are very few works that use this kind of evaluation (Setlur et al., 2021). This is mostly due to the fact that in many tasks is not completely clear how to divide the data to perform OOD testing (Arnold & Sha, 2021).

In this work, we propose a task-specific out-of-distribution testing procedure to evaluate the ability of an arbitrary glucose prediction model to generalize to unseen patients during training. The proposed procedure is formulated under the assumption that, since blood glucose behaviour is slightly different for each patient, each patient's CGM readings represent different child distributions drawn from the same meta-distribution, and when grouping different patients together, we can cover different parts of the meta-distribution space.

In concrete, our approach consists in dividing the data-set containing n patients in four different groups, namely **G1, G2, G3** and **G4**. **G1** is formed by n_{g_1} patients, whose data is split between training and validation sets with $r_{g_{1_{train}}} > r_{g_{1_{val}}}$, where $r_{g_{1_{train}}}$ corresponds to the fraction of the data used for training, and $r_{g_{1_{val}}}$ to the fraction used for validation. **G2** is formed by n_{g_2} patients, whose data is also split between training and validation sets; however, in this case $r_{g_{2_{train}}} < r_{g_{2_{val}}}$. **G3** is formed by n_{g_3} patients, and in this case all the data belongs to the validation set ($r_{g_{3_{val}}} = 1$). Finally, **G4**, formed by n_{g_4} patients, is used as the test set ($r_{g_{4_{test}}} = 1$). This is graphically represented in figure 3.3.

This particular data splitting is motivated by the fact that, as pointed out in (Teney et al., 2020), in general, there is a trade-off between in-distribution (ID) and OOD performance; therefore, to keep the model's ability to perform well in ID data and to allow it to generalize to OOD data, the validation set should contain both ID and OOD examples. Also, this data splitting gives us a natural way to test model's performance using varying degrees of knowledge about data from different patients, i.e, varying degrees of distributional shifts. For instance, **G1** can be considered as ID data since is mostly learned in training and, to a lesser extent, implicitly learned in validation (model uses validation data to optimize its performance). **G2** can be considered ID data as well, but to a lesser extent, since most of its data belongs to the validation set and a small fraction to the training set. On the other hand, **G3** can be partially considered OOD data but not completely, since it is implicitly learned in validation, and **G4**, can be completely considered OOD data, as it entirely belongs to the test set, and thus unknown during the training process. Therefore, we can evaluate performance on each group's validation (test) subset for measuring model's generalization capabilities at different degrees. Finally, patients belonging to each set can be shuffled in a cross-validation fashion to obtain more robust results. The specific patient data splitting used in this work is detailed in the next section.

3.4. Experiments

To illustrate the potential of the proposed meta-learning approach, we evaluate and compare it against other DL and classical ML baselines.

3.4.1. Data-set

For evaluation purposes, a data-set was generated using the FDA-accepted UVA/-PADOVA simulator (Man et al., 2014) that features a cohort of 10 virtual adult patients, which we use for our experiments. For each patient, 120 days were simulated with randomized meal sizes, meal times and meal occurrences, according to the distributions shown in Table 3.1. Additionally, to make the simulation more realistic, other sources

Table 3.1. Distributions of the randomized scenario. Here, Ber(p) stands for a Bernoulli distribution with parameter p, and Norm(μ, σ) represents a normal distribution with its corresponding mean and standard deviation.

Meal	Prob. of occurrence	Meal size [g]	Meal time [h]
Breakfast	Ber(0.95)	Norm(45, 10)	Norm(7, 1)
Snack 1	Ber(0.3)	Norm(10, 5)	Norm(9.5, 0.5)
Lunch	Ber(0.95)	Norm(70, 10)	Norm(12, 1)
Snack 2	Ber(0.3)	Norm(10, 5)	Norm(15, 0.5)
Dinner	Ber(0.95)	Norm(80, 10)	Norm(18.5, 1)

Table 3.2. Meal randomization details. MS and MT represent the true value of the meal size and meal time, respectively. Both values are obtained by sampling from their corresponding distribution shown in table 3.1.

Parameter	Distribution
Meal duration (MD) [min]	$\max(\min(\text{MS}/7 + \text{Norm}(0, 3), 2), 20)$
Meal size specification [g]	$\text{Uniform}(0.8\text{MS}, 1.2\text{MS})$
Meal time specification [h]	$\text{MT} + \text{Uniform}(-0.15, \text{MD}/60)$

of randomness, that are usually present in real life were introduced for each meal, such as variable meal duration, and meal size and time misspecifications by the patient, the details of which are shown in Table 3.2. Furthermore, insulin sensitivity for each patient was varied throughout the day, following the profile suggested by (Toffanin et al., 2013).

The data-set was then divided into train, validation and test sets, following the OOD data splitting introduced above, with $r_{g_{1_{train}}} = 0.85$, $r_{g_{1_{val}}} = 0.15$, $r_{g_{2_{train}}} = 0.15$, and $r_{g_{2_{val}}} = 0.85$. For our experiments, we used a 5-fold cross-validation scheme, where **G1**, formed by the first five virtual adult patients, was kept fixed, and the remaining groups varied according to table 3.3.

3.4.2. Baselines

As for the baselines, we consider several successful DL and ML architectures that have been presented in previous studies, some of which have reached promising results for the task of glucose prediction. As for the DL architectures, in the following we provide a brief

Table 3.3. Data-split used for the experiments.

k	G1	G2	G3	G4
1	Adult#001 - #005	Adult#006	Adult#007,#008	Adult#009,#010
2	Adult#001 - #005	Adult#007	Adult#006,#010	Adult#008,#009
3	Adult#001 - #005	Adult#008	Adult#009,#010	Adult#006,#007
4	Adult#001 - #005	Adult#009	Adult#006,#008	Adult#007,#010
5	Adult#001 - #005	Adult#010	Adult#007,#009	Adult#006,#008

summary of each one of them and we refer the reader to the original papers for additional details.

3.4.2.1. CNN-LSTM

(Li et al., 2020) This architecture is composed by four convolutional layers, each of them followed by a max-pooling operator. Then, a modified LSTM layer is used to capture the temporal relationships of the features extracted by the convolutional part, and finally, three Feed-Forward layers are used to produce the prediction.

3.4.2.2. LSTM

(Mohebbi et al., 2020) This is a two-layer LSTM architecture followed by a feed-forward layer to transform the output of the LSTM and produce the predictions.

3.4.2.3. Bi-LSTM

(Sun et al., 2018) In this case, a LSTM layer is used first, followed by a bidirectional LSTM to capture the context of the features produced by the LSTM in both directions. Finally, four feed-forward layers are used to produce the output at each step.

3.4.2.4. FFN

In this case, a feed-forward network with the exact same architecture than our proposed network, but with a feed-forward layer instead of the Kronecker layer is implemented. This is done to compare our approach with a commonly used FFN.

As for the classical ML baselines, SVR with linear kernel (SVRL), a SVR with radial basis kernel (SVRRB), random forest (RF) and an ARX model with the same hyperparameters recommended by (Xie & Wang, 2020) were used. Additionally, a XGBOOST model with 300 estimators was used as well due to its recent success in several tasks, outperforming even DL models.

Finally, is worth mentioning that for all DL architectures, their size was modified in order to have a comparable number of parameters with the proposed architecture, that is $\sim$ 47,000 parameters. Also, the same inputs, window sizes, loss and prediction horizon where used for all the baselines.

3.4.3. Evaluation metrics

Several evaluation metrics were used to test the performance of the proposed approach. Average and point root mean squared error (RMSE) and mean absolute relative difference (MARD) were used as primarily indicators of model's accuracy. The average RMSE is given by

$$\text{RMSE}_{avg} = \frac{1}{p}\sum_{k=1}^{p}\left(\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\mathbf{y}_{CGM}(t+k) - \hat{\mathbf{y}}_{CGM_t}(k)\right)^2}\right), \tag{3.7}$$

Similarly, the average MARD is given by

$$\text{MARD}_{avg} = \frac{1}{p}\sum_{k=1}^{p}\left(\frac{100}{N}\sum_{t=1}^{N}\frac{|\mathbf{y}_{CGM}(t+k) - \hat{\mathbf{y}}_{CGM_t}(k)|}{\mathbf{y}_{CGM}(t+k)}\right), \tag{3.8}$$

where p is the length of the predicted sequence. Likewise, the point versions of these metrics (RMSE_p and MARD_p) are given by evaluating the average versions at a particular

horizon. Even though these metrics are standard indicators of model's accuracy, they are limited in the clinical insight they provide, therefore task-specific metrics were used as well, such as temporal gain (TG) (Facchinetti et al., 2011) and the Clark error grid analysis (C-EGA) (Clarke et al., 1987).

The temporal gain is defined as the average time gained for early detection of a potential hypo/hyper glycemia event and is given by

$$\Delta = \underset{i\in[0,p]}{\operatorname{argmin}} \left(\frac{1}{N-p} \sum_{t=1}^{N-p} (\mathbf{y}_{CGM}(t) - \hat{\mathbf{y}}_{CGM_t}(i))^2 \right), \tag{3.9}$$

$$TG = (p - \Delta)T_s, \tag{3.10}$$

where T_s represents the sampling rate. Here, Δ measures the temporal shift that minimizes the distance between the prediction and the actual CGM value. A larger TG indicates an earlier detection of a potential adverse event, and a TG$= 0$ indicates no temporal advantage of using the model and therefore, a model with TG$= 0$ would be useless from a clinical perspective.

On the other hand C-EGA (Clarke et al., 1987), which was originally developed to evaluate the clinical acceptability of CGM devices, considers the clinical significance of the difference between the real and the predicted blood glucose values generated by a model. This is done by defining a grid between both quantities, and dividing it in five different zones (A, B, C, D, E) with their corresponding clinical interpretation. Zone A corresponds to clinically accurate predictions, which would led to a correct treatment. Predictions that fall in Zone B are considered clinically acceptable and would led to benign treatment. Zone C predictions would led to an over-correcting treatment. Zone D predictions represent a failure to detect (and in consequence to treat) hypoglycemic or hyperglycemic events. Finally, predictions in Zone E would lead to an erroneous or opposite treatment given the real values of blood glucose and thus, predictions falling in this zone are clinically unacceptable.

Table 3.4. Hyper-parameters of the proposed architecture (parameters with an asterisk are shared by all the models).

Parameter	Value
N° neurons	100-300-100 50-50-1
N° parameters	47,520
α (inner loop Lr)	1×10^{-2}
β (outer loop Lr)	1×10^{-3}
τ*	30 (2.5 hours)
T_s* (sampling rate)	5 min
p*	30 min, 60 min, 90 min

For our evaluation, we will rank the models according to whether they maximize the membership of their predictions to Zone A and to Zones A + B (overall clinically acceptable predictions) and minimize the membership to Zone C + D, which could lead to a deficient treatment. Additionally, membership to Zone E can be calculated from the aforementioned results considering that all memberships add up to 100%. A model with a positive membership to Zone E should be immediately red-flagged and eliminated.

3.4.4. Training details

For the implementation of all the models, Python was used as the main programming language. DL models were implemented using Pytorch as the main library, and Scikit-learn was used for the ML models. Additionally, the *learn2learn* (Arnold et al., 2020) implementation of the MAML algorithm was used for our meta-learning network.

All the DL models were optimized using Adam optimizer. For the baselines, a learning rate of 1×10^{-3} was used in the initial training stage, and a learning rate of 1×10^{-5} was used for fine-tuning. Other training details are shown in table 3.4, which shows both hyper-parameters specific to our meta-learning network and hyper-parameters shared by all the implemented models.

3.4.5. Experimental Results

For the subsequent discussion we report the results of our proposed model in two ways, namely *MetaL* and *MetaL*$_1$. *MetaL* is the proposed model without adapting the head, so it can be considered as a population model, and *MetaL*$_1$, which is as *MetaL* but with the head adapted using only one meal event per patient.

3.4.5.1. OOD evaluation results

Table 3.5 shows the results of our proposed OOD evaluation procedure. Each sub-column under the respective evaluation metric shows the value of that metric when the model was trained for different prediction horizons, namely $p = 30[min]$, $p = 60[min]$ and $p = 90[min]$.

From the table, it can be seen that the ARX model presents worse results than all other models in virtually all metrics, difference that is accentuated for longer horizons and larger distributional shifts. These results are consistent with those of other studies, such as (De Bois et al., 2022), which also show how more complex ML and DL models surpass other linear techniques.

As for the ML models, only XGBOOST and SVR with linear kernel (SVRL) achieve comparable performance to the DL models; however, their performance tends to degrade faster as the prediction horizon increases. Also, is worth mentioning that tree-based models, as XGBOOST and random forest (RF), although may have acceptable and even great performance on average (as in the case of XGBOOST), they tend to have large errors for extreme values, produced for example by a large meal. This is evidenced in figure 3.5 where the XGBOOST error box plots tend to have the largest extreme values, and by the fact that both the RF and XGBOOST are the only models which produce predictions that fall in Zone E in the C-EGA plot (not shown in the table), which is clinically unacceptable. This may be due to the averaging nature of these models, where the effect of accurate trees

might be smoothed or cancelled out by other less accurate trees, producing a low-pass filter effect for extreme values.

Regarding the DL models, with the only exception of the FNN network, all show very strong performance in every metric. In particular, the Bi-LSTM network is the best performing model when evaluating in group G1 (ID data). However, as the distributional shift increases, its performance degrades much faster than our proposed meta-learning model, which maintains a robust performance in every metric across all prediction horizons and different degrees of distributional shifts, being the best performing model when evaluating on groups G2, G3 and G4. It is worth noting that even without adaptation (MetaL), our approach performs better than other population models when there is a high degree of distributional shift, showing that training models with meta-learning produces models that generalize better and may be useful in real environments where distributional shifts may be high.

Finally, we compare the results with other works as (Li et al., 2020), where a similar data-set is generated with the same adult cohort of the the UVA/PADOVA simulator. They obtained a punctual RMSE value at horizon $p = 60[min]$ with the CNN-LSTM architecture of 18.87 [mg/dl], using the same patients for training and testing. This is considerable higher than the results obtained in this work with the same network architecture in groups G1 and G2, which represent a similar setting (9.43 [mg/dl] and 14.23 [mg/dl], respectively). This performance improvement may be due to the meal event separation in our data-set for training the models, which eliminates quasi-steady state information from the data-set and a causal relationship between inputs and outputs is enforced. However, this research question is out of the scope and must be left for future work.

3.4.5.2. Personalized model results

Tables 3.6 and 3.7 show how well the best performing models from table 3.5 compare when evaluated with data of different patients from group G4 (patients never seen during training) in terms of average RMSE (table 3.6) and predictions falling in Zone A (table

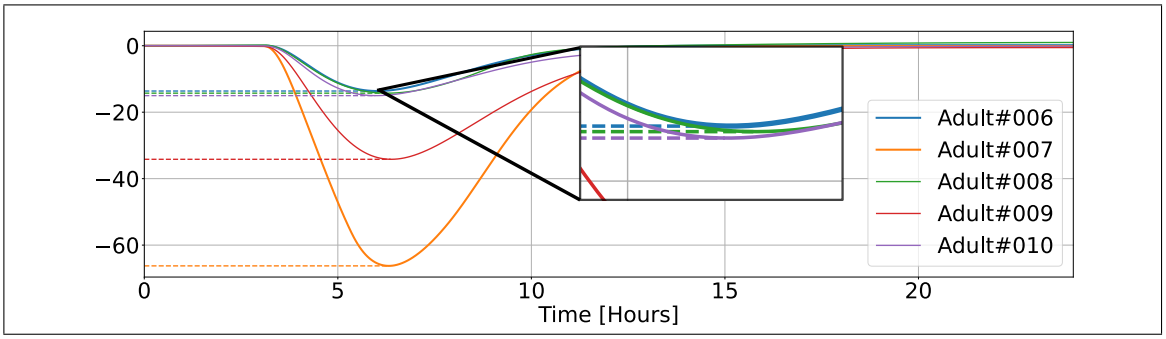


Figure 3.4. Patients' glucose response to an insulin bolus of one unit [U] when their nominal insulin sensitivity is used. The y-axis represents the glucose values in [mg/dl] minus each patient's fasting blood glucose value. The minimum of each curve indicates how sensitive the respective individual is to insulin.

3.7). We can see that while MetaL and MetaL$_1$ achieve the best results for most patients at different horizons, these differences are more pronounced for Adult#006 and Adult#007 at long prediction horizons, which are the least and most insulin sensitive patients among the group (see figure 3.4). This demonstrates the generalization capabilities of the proposed approach and how MetaL$_1$ is capable of extracting important information from a single meal event to improve the performance even further.

Additionally, figure 3.5 shows the error distribution of each model for a prediction horizon $p = 60[min]$. As mentioned before, we can see that even though XGBOOST has a low average error in general, it produces the largest extreme values among the models, which could be unsafe in real-life scenarios. On the other hand, both LSTM and Bi-LSTM tend to have tighter distributions than the other baselines even when their average performance is not significantly better. However, unlike other models, MetaL and MetaL$_1$ tend to a tighter error distributions and a low average error for all patients, which are good indicators of the superiority of the proposed models.

3.4.5.3. Transfer-Learning vs Meta-Learning

Finally, we evaluate how well a DL model trained with meta-learning and adapted with just one meal event and five training iterations (MetaL$_1$), compare with fine-tuned models

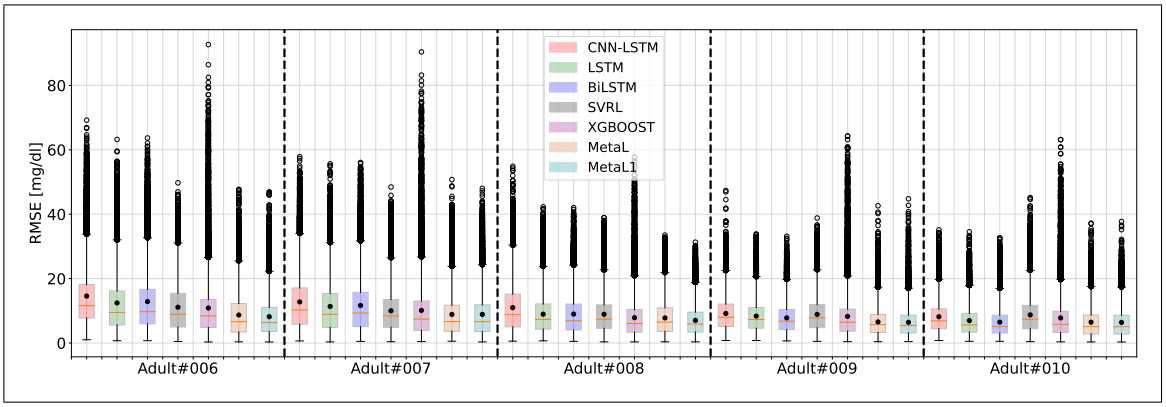


Figure 3.5. Box plots of RMSE_{avg} for different models with patients of group G4. The black dot inside the box represents the mean and the orange line the median. The limits of the box are given by quartiles 25 (Q_1) and 75 (Q_3), the position of the whiskers is given by $Q_1 - 1.5(Q_3 - Q_1)$ and $Q_3 + 1.5(Q_3 - Q_1)$, and the points above the upper whisker represent the statistical outliers or extreme values.

on patients never seen during training when there is an increasing amount of available data and an unlimited number of training iterations (in this case an early stopping strategy is followed).

From figure 3.6 we can see that both FNN and CNN-LSTM never surpass the performance of MetaL_1, even with 20 times more data from a particular patient. On the other hand, LSTM and Bi-LSTM perform increasingly better when fine-tuned with more and more meal events. However, it takes about 6 times more data and thus, it would be six times slower for these models to outperform MetaL_1, which may not be fast enough to adapt to other eventual real-life perturbations like exercise or sleep quality the night before (Porter, 2020).

3.5. Probabilistic extension

Even though, an accurate and personalizable predictive model could be very useful for the AP, there are some challenges that remain, especially for hybrid control approaches in

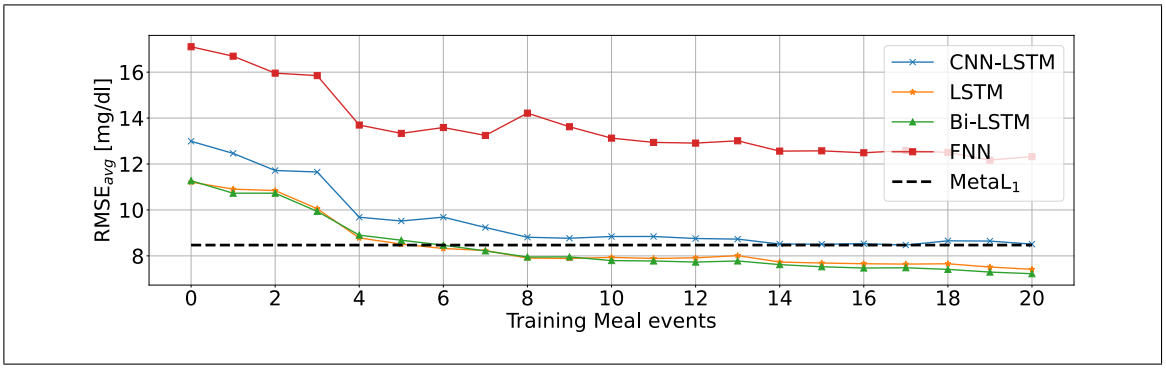


Figure 3.6. Transfer-learning performance versus meta-learning. The dashed black line represents the performance of the proposed network when the head is adapted with only one meal event. The other traces represent the performance of the DL baselines when fine-tuned with a variable number of meal events with an unlimited number of training iterations, using and early stopping criterion to stop training. For fine-tuning a learning rate of 1×10^{-5} and a training-validation ratio of 0.8 were used.

which a meal announcement and the carbohydrate content of a meal is expected from the user (Shi et al., 2019).

The need for user intervention introduces important uncertainties to the problem, that when are not considered, lead to sub-optimal outcomes of the controller (Daniels et al., 2022a). In fact, multiple studies have shown that individuals with T1D tend to have significant rates of late meal insulin boluses (20-45% of the meals) (Robinson et al., 2021) and meal size estimation errors between 20% and 59% (Brazeau et al., 2012; Meade & Rushton, 2016), which leads to higher glucose variability and decreased time with glucose values in a desirable range (80 mg/dl - 180 mg/dl) (Brazeau et al., 2012).

This motivates the design and implementation of a probabilistic model which can estimate the uncertainties produced both by late meal announcements and by meal size misspecifications.

For this purpose, we follow exactly the same data prepossessing mentioned above, with multiple tasks, and each of them divided in a support and a target set to allow personalization. However, a new model architecture for probabilistic modeling is proposed, since

Table 3.5. OOD evaluation results for different horizons: $p = 30[min]$ | $p = 60[min]$, | $p = 90[min]$. Numbers in bold represent the best result for each metric at a specific horizon.

G1

Model	RMSE$_{avg}$ [mg/dl]			RMSE$_p$ [mg/dl]			MARD$_{avg}$ [%]			MARD$_p$ [%]			TG [%]			Zone A [%]			Zone A+B [%]			Zone C+D [%]		
Bi-LSTM	**3.69**	5.76	**6.77**	**5.06**	**7.79**	**9.33**	**2.36**	3.71	**4.31**	**3.23**	**5.13**	**6.3**	**100.0**	**99.0**	**98.89**	**99.87**	**98.53**	**96.77**	99.96	**99.44**	98.88	0.04	**0.55**	1.11
LSTM	3.99	6.14	7.21	5.55	8.39	9.8	2.56	3.98	4.65	3.55	5.58	6.68	98.67	97.67	97.33	99.84	98.13	96.67	**99.98**	99.36	**98.92**	**0.02**	0.66	**1.08**
CNN-LSTM	4.89	7.15	8.3	6.39	9.43	11.23	3.11	4.57	5.31	4.09	6.25	7.66	98.0	97.0	97.33	99.6	97.25	94.73	99.92	99.17	98.5	0.09	0.82	1.5
MetaL	4.12	6.28	7.79	6.12	9.69	11.77	2.64	4.07	5.09	3.92	6.51	8.24	95.55	97.33	97.78	99.72	97.37	94.32	99.95	99.16	98.42	0.05	0.84	1.57
MetaL$_1$	3.78	**5.73**	7.19	5.56	8.71	10.73	2.44	**3.70**	4.57	3.57	5.82	7.23	96.3	97.17	97.11	99.77	98.03	95.69	99.97	99.39	98.91	**0.02**	0.61	1.09
FNN	10.36	11.66	15.3	12.48	13.66	12.81	7.06	8.03	8.72	8.55	9.84	11.31	93.32	97.0	98.67	94.65	92.14	89.15	97.18	93.05	96.33	2.53	2.86	3.66
ARX	8.15	14.43	18.82	14.51	24.16	28.45	5.13	9.3	12.38	9.2	16.6	20.91	33.33	17.33	15.33	90.79	69.97	59.7	98.44	96.31	36.16	1.56	3.67	4.14
RF	7.34	12.67	16.62	12.72	20.99	25.37	4.33	7.32	9.54	7.29	12.12	15.32	51.33	47.67	59.33	94.55	83.44	74.99	99.68	98.66	97.55	0.33	1.34	2.43
SVRL	5.29	8.78	11.0	9.13	13.86	16.09	3.41	5.84	7.39	6.0	9.62	11.44	82.0	93.67	96.89	97.87	92.28	87.95	99.63	98.14	97.07	0.38	1.87	2.94
SVRRB	6.72	10.02	12.02	10.59	14.9	16.54	3.9	5.94	7.15	6.29	9.16	10.47	90.67	96.0	97.33	97.26	91.19	88.2	99.81	99.35	98.90	0.2	0.66	1.1
XGBOOST	4.58	7.33	9.59	7.7	11.98	14.84	2.67	4.36	5.69	4.44	7.36	9.44	86.68	93.67	97.11	98.68	95.64	91.4	99.87	99.2	95.86	0.13	0.8	1.49

G2

Model	RMSE$_{avg}$ [mg/dl]			RMSE$_p$ [mg/dl]			MARD$_{avg}$ [%]			MARD$_p$ [%]			TG [%]			Zone A [%]			Zone A+B [%]			Zone C+D [%]		
Bi-LSTM	5.32	8.33	10.12	7.55	**11.92**	14.49	3.49	5.59	6.85	4.92	**8.35**	10.74	83.33	93.30	96.67	99.14	**93.72**	87.92	99.72	98.03	96.6	0.28	1.96	3.4
LSTM	5.4	8.88	10.8	7.75	12.92	15.6	3.6	6.05	7.5	5.14	9.24	11.86	86.66	93.32	94.44	99.12	92.36	85.64	99.74	98.04	96.34	0.28	1.98	3.64
CNN-LSTM	6.69	10.39	12.31	9.09	14.23	16.87	4.29	6.91	8.36	5.82	10.04	12.68	80.0	91.67	94.44	98.4	89.76	82.42	99.66	97.86	96.18	0.32	2.16	3.82
MetaL	5.23	8.25	10.47	7.93	13.0	15.48	3.43	5.42	6.97	5.17	8.9	11.47	90.0	91.67	95.56	99.34	92.98	86.06	99.86	98.2	97.0	0.12	1.76	3.0
MetaL$_1$	**4.88**	**7.76**	**9.48**	**7.35**	12.16	**14.36**	3.22	**5.17**	**6.33**	**4.81**	8.47	**10.41**	**90.33**	92.17	95.56	**99.38**	93.7	**88.91**	**99.9**	**98.45**	**97.40**	**0.1**	**1.55**	**2.62**
FNN	12.17	15.86	17.6	14.81	19.37	21.93	8.95	11.57	12.83	10.96	14.90	17.48	86.64	**93.83**	95.56	88.18	77.74	71.48	94.64	93.82	92.8	5.36	6.2	7.24
ARX	8.96	16.4	21.77	16.2	28.26	33.79	6.09	11.38	15.57	11.17	21.11	27.4	33.33	18.33	16.67	86.38	59.08	47.04	96.78	92.32	91.64	3.26	7.62	8.38
RF	8.11	14.77	19.45	14.08	25.13	30.04	5.09	8.93	11.84	8.57	15.33	19.48	50.0	48.33	57.78	92.18	76.1	67.82	99.08	96.52	94.48	0.92	3.4	5.2
SVRL	5.55	9.46	12.14	9.69	15.35	18.34	3.77	6.67	8.76	6.7	11.48	14.36	73.33	90.0	94.44	97.02	86.74	79.88	99.26	96.2	94.14	0.72	3.84	5.84
SVRRB	7.67	11.33	13.74	11.43	16.61	19.12	4.49	7.06	8.69	7.01	11.1	13.27	86.60	93.34	94.44	96.24	86.28	81.0	99.5	97.84	96.86	0.5	2.12	3.12
XGBOOST	5.45	9.04	11.87	9.09	15.07	18.51	**3.17**	5.4	7.14	5.24	9.25	12.05	83.33	90.0	**96.69**	97.28	91.72	85.46	99.52	98.4	96.92	0.46	1.64	3.04

G3

Model	RMSE$_{avg}$ [mg/dl]			RMSE$_p$ [mg/dl]			MARD$_{avg}$ [%]			MARD$_p$ [%]			TG [%]			Zone A [%]			Zone A+B [%]			Zone C+D [%]		
Bi-LSTM	6.91	11.17	13.44	9.85	15.97	18.73	4.32	7.21	8.88	6.15	10.92	13.81	81.67	90.0	95.0	97.67	87.23	78.75	99.46	97.48	95.94	0.53	2.5	4.06
LSTM	6.82	10.73	13.12	9.97	15.71	18.58	4.35	7.16	8.9	6.31	11.04	14.0	80.0	87.5	94.44	97.70	86.32	77.44	99.51	97.6	96.08	0.48	2.41	3.91
CNN-LSTM	8.11	12.29	14.41	11.13	16.57	19.28	4.96	7.91	9.5	6.82	11.44	14.27	73.33	87.5	92.22	96.45	85.55	77.33	99.47	97.36	95.89	0.54	2.6	4.1
MetaL	5.68	8.45	10.71	8.58	13.07	15.99	3.67	5.40	6.94	5.53	8.72	11.28	**86.67**	93.34	95.56	98.76	93.87	86.59	99.72	**98.58**	**97.44**	0.27	**1.41**	**2.56**
MetaL$_1$	**5.39**	**7.93**	**10.07**	**8.1**	**12.15**	**15.02**	**3.52**	**5.17**	**6.69**	**5.29**	**8.3**	**10.9**	85.5	**93.75**	**97.25**	**98.92**	**94.49**	**88.1**	**99.75**	98.48	97.12	**0.25**	1.52	2.89
FNN	13.67	16.39	17.95	16.79	19.66	21.86	9.89	12.00	13.03	12.22	15.23	17.55	81.67	93.30	97.20	84.79	77.0	71.5	94.66	93.52	92.86	5.31	6.49	7.16
ARX	9.07	16.79	22.21	16.42	29.01	34.63	6.18	11.56	15.73	11.36	21.43	27.62	33.33	17.5	15.56	85.62	57.9	46.12	96.53	92.81	92.26	3.46	7.16	7.77
RF	8.51	15.5	20.46	14.87	26.53	32.06	5.4	9.71	12.92	9.25	16.84	21.74	51.67	44.17	48.89	90.45	72.13	61.53	98.55	96.24	94.53	1.43	3.64	5.23
SVRL	5.88	10.57	13.33	10.43	16.94	20.46	3.96	7.1	9.37	7.12	12.34	15.53	73.33	87.5	93.32	96.36	84.11	75.58	99.12	96.04	94.16	0.86	3.96	5.82
SVRRB	10	14.81	17.95	14.96	21.69	24.53	5.85	9.12	11.38	9.33	14.52	17.39	76.67	89.17	92.22	90.99	75.81	68.94	98.79	97.38	96.16	1.23	2.61	3.8
XGBOOST	6.18	10.92	14.64	10.54	18.58	23.4	3.77	6.65	9.0	6.42	11.66	15.58	75.0	82.5	91.11	96.4	86.18	75.18	99.14	97.61	96.15	0.88	2.41	3.82

G4

Model	RMSE$_{avg}$ [mg/dl]			RMSE$_p$ [mg/dl]			MARD$_{avg}$ [%]			MARD$_p$ [%]			TG [%]			Zone A [%]			Zone A+B [%]			Zone C+D [%]		
Bi-LSTM	7.29	11.27	13.61	10.39	16.07	18.95	4.46	7.23	8.9	6.35	10.88	13.73	78.33	88.33	94.44	97.13	86.85	78.73	99.57	97.46	95.97	0.43	2.55	4.14
LSTM	7.19	11.21	13.61	10.44	16.21	18.97	4.41	7.25	8.92	6.37	11.03	13.73	76.67	89.17	95.0	97.3	86.6	78.87	99.56	97.48	96.04	0.45	2.52	3.98
CNN-LSTM	8.23	12.99	15.19	11.23	17.45	20.18	5.04	8.38	9.98	6.91	12.02	14.75	78.33	91.67	95.0	96.62	83.97	76.15	99.38	97.27	96.13	0.62	2.74	3.88
MetaL	5.9	8.93	11.34	8.94	13.86	16.95	3.76	5.67	7.32	5.67	9.18	11.92	85.0	92.5	95.56	98.82	92.34	84.85	**99.77**	**98.39**	97.19	**0.24**	**1.62**	2.83
MetaL$_1$	**5.5**	**8.41**	**10.49**	**8.28**	**12.97**	**15.62**	**3.54**	**5.46**	**6.87**	**5.31**	**8.82**	**11.16**	**85.3**	**94.16**	95.0	**98.94**	**92.79**	**86.90**	99.74	98.16	**97.29**	0.26	1.85	**2.71**
FNN	13.22	17.11	18.89	16.15	20.67	23.15	9.83	12.43	13.56	12.14	15.84	18.17	80.0	91.67	**97.22**	86.32	74.66	68.89	94.27	93.67	93.31	5.74	6.32	6.69
ARX	9.1	16.63	22.15	16.45	28.76	34.46	6.13	11.54	15.63	11.27	21.41	27.31	33.33	19.17	16.11	85.9	58.0	45.96	96.76	92.34	92.48	3.26	7.67	7.54
RF	8.59	15.3	20.23	14.99	26.14	31.56	5.34	9.45	12.58	9.15	16.42	21.01	50.0	44.17	48.89	90.67	72.91	62.67	98.93	96.54	94.9	1.06	3.32	4.94
SVRL	5.92	10.28	13.49	10.51	17.03	20.81	3.97	7.12	9.48	7.16	12.42	15.7	73.33	85.83	93.89	96.31	84.18	75.24	99.15	96.05	94.35	0.86	3.96	5.63
SVRRB	9.9	14.77	17.85	14.96	21.75	24.58	5.87	9.34	11.57	9.39	14.92	17.65	78.33	86.65	91.67	90.49	75.45	69.64	98.83	97.12	96.17	1.19	2.82	3.81
XGBOOST	6.39	10.72	14.25	10.86	18.3	22.72	3.99	6.72	9.0	6.84	11.9	15.63	76.67	83.33	93.30	95.8	85.89	76.27	99.07	97.05	95.71	0.93	2.98	4.28

it was seen in practice that the proposed meta-learning model tends to be overconfident with its predictions when meal misspecifications occur (see section 3.5.3.4), which is not acceptable for safety critical applications.

Table 3.6. RMSE_{avg}[mg/dl] for each patient in group G4 at horizons $p = 30[min] \mid p = 60[min], \mid p = 90[min]$. Numbers in bold represent the best result at each prediction horizon.

$\text{RMSE}_{avg}[mg/dl]$															
Model	**Adult#006**			**Adult#007**			**Adult#008**			**Adult#009**			**Adult#010**		
Bi-LSTM	10.96	15.58	18.5	8.5	13.88	16.7	6.61	10.62	13.17	5.41	8.69	10.6	4.98	7.61	9.09
LSTM	10.6	15.01	17.83	8.17	13.51	16.29	6.6	10.28	12.92	5.51	9.25	11.34	5.07	8.01	9.69
CNN-LSTM	10.97	17.23	19.42	9.66	15.23	17.73	7.91	12.96	15.61	6.43	10.22	12.40	6.19	9.31	10.8
MetaL	7.41	10.48	13.16	6.73	10.65	13.45	5.74	8.77	11.37	4.75	7.23	9.34	4.87	7.51	9.38
MetaL_1	7.01	**9.79**	**12.24**	6.33	**10.61**	**12.18**	5.11	**7.60**	**10.08**	**4.23**	**6.81**	**8.50**	**4.82**	**7.25**	**9.46**
SVRL	**6.67**	12.15	16.3	**6.04**	10.82	14.38	5.79	9.63	12.48	5.43	9.36	12.23	5.69	9.44	12.08
XGBOOST	8.19	12.92	16.81	7.7	12.52	16.77	**5.05**	9.13	12.49	5.73	9.73	13.13	5.28	9.28	12.04

Table 3.7. Predictions falling in Zone A of the C-EGA plot for each patient in group G4 at horizons $p = 30[min] \mid p = 60[min], \mid p = 90[min]$. Numbers in bold represent the best result at each prediction horizon.

Zone A[%]															
Model	**Adult#006**			**Adult#007**			**Adult#008**			**Adult#009**			**Adult#010**		
Bi-LSTM	91.8	77	67.05	97.1	81.5	73.3	99.55	94.49	87.4	99.0	91.2	82.7	98.2	90.05	83.2
LSTM	92.9	77.85	69.05	97.55	82.45	75.1	99.5	94.94	88.95	98.6	88.55	78.75	97.95	89.2	82.5
CNN-LSTM	92.0	72.3	64.75	96.65	80.65	73.05	99.4	90.9	82.85	97.55	86.7	76.65	97.5	89.3	83.45
MetaL	97.95	91.15	83.05	98.5	**89.95**	81.85	99.6	96.65	89.94	99.4	93.4	85.4	**98.65**	90.55	**84.0**
MetaL_1	**98.3**	**91.35**	**84.50**	**98.57**	89.65	**85.85**	**99.79**	**97.85**	**93.8**	**99.53**	**94.3**	**87.75**	98.52	**90.80**	82.6
SVRL	94.55	75.25	66.1	97.3	85.9	76.2	98.1	93.4	86.85	96.5	83.85	74.45	95.2	82.5	72.6
XGBOOST	91.5	78.45	67.1	96.3	84.6	74.2	98.8	94.4	87.3	96.45	86.8	74.15	95.95	85.2	78.6

The proposed probabilistic model is inspired in the ideas of (Becker et al., 2019; Shaj et al., 2020, 2022; Gillijns & De Moor, 2007).

3.5.1. Related work

In (Becker et al., 2019), the integration of deep time-series modeling with Kalman Filters is proposed for uncertainty estimation under the name of *Recurrent Kalman Networks* (RKN). Unlike other similar approaches, that require the use of approximate inference techniques due to nonlinear approximations, such as variational inference, in (Becker et al., 2019) locally linear models are used to propagate uncertainty using Kalman equations over the latent space of a neural network. This has the advantage of obtaining closed form solutions for the posterior distributions, so that, approximation errors are avoided, and by the use of factorized state representations, the Kalman updates are simplified to scalar

operations that avoid the hard to backpropagate, computational heavy and potentially unstable matrix inversions.

Concretely, in (Becker et al., 2019) the authors propose to learn a mapping from the input space $\mathcal{I}$ to an observation space $\mathcal{W} = \mathbb{R}^m$ by the use of an observation encoder. This encoder outputs both the transformation of the observation at time t, given by $\hat{\mathbf{w}}(t)$, and a vector $\hat{\mathbf{r}}(t)$, representing the diagonal of the measurement noise covariance matrix $\mathbf{R}(t) \in \mathbb{R}^{m\times m}$. On the other hand, the latent state space $\mathcal{Z} = \mathbb{R}^n$, with $n > m$, is related to the observation space by the linear latent transformation $\mathbf{C} = [\mathbf{C}_u \mathbf{0}] \in \mathbb{R}^{m\times n}$ with $\mathbf{0}$ a zero matrix in $\mathbb{R}^{m\times(n-m)}$ and $\mathbf{C}_u$ a diagonal matrix in $\mathbb{R}^{m\times m}$, with associated vector $\mathbf{c}_u \in \mathbb{R}^m$, containing its diagonal values. This choice for $\mathbf{C}$, naturally divides the state in two parts, namely, $\mathbf{z}_u(t) \in \mathbb{R}^m$ which carries information that can be directly extracted from the observations, and $\mathbf{z}_l(t) \in \mathbb{R}^{n-m}$ which holds information inferred over time, similar to velocities or other derivatives in state space representations of physical processes. By convenience, the authors used $n = 2m$ and $\mathbf{c}_u = \mathbf{1}_m$, the m-dimensional $\mathbf{1}$ vector.

As for the transition matrix $\mathbf{A}(t) \in \mathbb{R}^{n\times n}$, to obtain a locally linear transformation, the authors proposed to learn K constant transition matrices $\mathbf{A}^{(k)}$ and combine them using state dependent coefficients $\alpha^{(k)}(\tilde{\mathbf{z}}(t-1))$, obtained by using a linear learnable transformation with softmax activation function, i.e.

$$\mathbf{A}(t) = \sum_{k=1}^{K} \alpha^{(k)}(\tilde{\mathbf{z}}(t-1))\mathbf{A}^{(k)} \tag{3.11}$$

Where $\tilde{\mathbf{z}}(t-1)$ is the estimated state after the update step of the Kalman filter at time $t-1$.

For the process noise covariance matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$, a diagonal structure is considered with $\mathbf{q} \in \mathbb{R}^n$ a learnable constant vector containing the digonal values of $\mathbf{Q}$. And for the state covariance matrix $\hat{\mathbf{\Sigma}}(t) \in \mathbb{R}^{n\times n}$, it is proposed the use of a block structure of the form:

$$\hat{\mathbf{\Sigma}}(t) = \begin{bmatrix} \hat{\mathbf{\Sigma}}_u(t) & \hat{\mathbf{\Sigma}}_s(t) \\ \hat{\mathbf{\Sigma}}_s(t) & \hat{\mathbf{\Sigma}}_l(t) \end{bmatrix} \tag{3.12}$$

Where each of $\hat{\mathbf{\Sigma}}_u(t), \hat{\mathbf{\Sigma}}_s(t), \hat{\mathbf{\Sigma}}_l(t) \in \mathbb{R}^{m \times m}$ is a diagonal matrix with associated vectors $\hat{\sigma}_u(t)$, $\hat{\sigma}_s(t)$ and $\hat{\sigma}_l(t)$ which contain the diagonal values of their respective matrices.

After applying the Kalman equations to this simplified structure, the one-step ahead prediction and an uncertainty estimate can be obtained by transforming back the state $\hat{\mathbf{z}}(t+1)$ and the uncertainty estimates $\hat{\sigma}_u(t+1)$, $\hat{\sigma}_s(t+1)$ and $\hat{\sigma}_l(t+1)$ to the input space, by the use of an output decoder. Finally, the network can be trained in an end-to-end manner by minimizing any type of probabilistic loss, as the negative Gaussian log-likelihood loss, which gives:

$$L_y(t+1) = -\log \mathcal{N}(\mathbf{y}(t+1) | \hat{\mathbf{y}}(t+1), \hat{\sigma}(t+1)) \tag{3.13}$$

Authors showed the superiority of RKNs over other approaches in probabilistic modeling of time-series in several tasks. Figure 3.7 shows a simplified scheme of a RKN when adding an external input (Shaj et al., 2020). The reader is referred to the original paper for further details on the simplified Kalman equations and its implementation on a neural network.

Additionally, (Shaj et al., 2022) proposed an extension of the work of (Becker et al., 2019) and (Shaj et al., 2020), by adding a Bayesian Context Aggregation (BCA) (Volpp et al., 2021) to the RKN architecture, in order to allow the adaptation of the network to changing dynamic scenarios. The BCA consists in inferring a latent variable $\mathbf{l_n} \sim N(\mathbf{l}_n | \mu_{ln}, \text{diag}(\sigma_{ln}))$ from a context set $\mathbf{Cl_n} = \mathbf{Y}_{\mathbf{n}[j-\tau_c:j]}$ by using the Bayes rule. The observation model is a factorized Gaussian model of the form $p(\mathbf{r}_{in} | \mathbf{l}_n)$, where:

$$\mathbf{r}_{in} = enc_r(\mathbf{Y_n}(j-i)) \tag{3.14}$$

$$\sigma_{in} = enc_\sigma(\mathbf{Y_n}(j-i)) \tag{3.15}$$

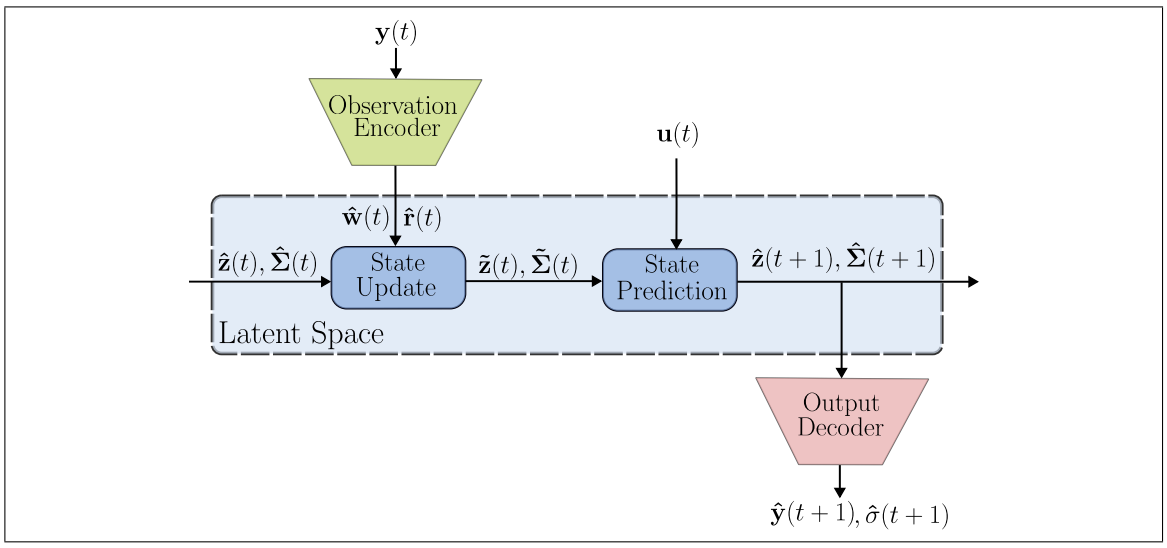


Figure 3.7. Recurrent Kalman network architecture proposed in (Becker et al., 2019) and updated in (Shaj et al., 2020) to incorporate an external input. In this architecture, both past target data represented by $\mathbf{y}(t)$ and an external input represented by $\mathbf{u}(t)$, are combined using the Kalman equations on the latent space of a neural network to generate the next step prediction $\mathbf{y}(t+1)$ and a confidence level represented by $\hat{\sigma}(t+1)$. In this case, is not possible to incorporate context data for adaptation or personalization.

With enc_r and enc_σ as two encoders. Then, given a prior for $\mathbf{l}_n$, $p_0(\mathbf{l}_n) = N(\mathbf{l}_n|\mu_\mathbf{0}, \text{diag}(\sigma_0))$, the posterior distribution for $\mathbf{l}_n$ can be obtained by the application of the Bayes rule, which by Gaussianity assumption simplifies to:

$$\sigma_{ln}^2 = \frac{1}{\left(\frac{1}{\sigma_0^2} + \sum_{i=0}^{\tau_c} \frac{1}{\sigma_{ln}^2}\right)} \tag{3.16}$$

$$\mu_{ln} = \mu_0 + \sigma_{ln}^2 \sum_{i=0}^{\tau_c} \frac{\left(\mathbf{r}_{in} - \mu_0\right)^2}{\sigma_{in}^2} \tag{3.17}$$

Once obtained, μ_{ln} and σ_{ln} are integrated in the prediction step of the Kalman equations in the following way:

$$\hat{\mathbf{z}}(t+1) = \mathbf{A}(t)\tilde{\mathbf{z}}(t) + f_b(\mathbf{u}(t)) + f_\mu(\mu_{ln}) \tag{3.18}$$

$$\hat{\mathbf{\Sigma}}(t+1) = \mathbf{A}(t)\tilde{\mathbf{\Sigma}}(t)\mathbf{A}^T(t) + \mathbf{F}_\sigma + \mathbf{Q} \tag{3.19}$$

Where f_b and f_μ are nonlinear learnable transformations to incorporate the influence of the external input $\mathbf{u}(t)$ (Shaj et al., 2020), and the task information, respectively. Also, $\mathbf{F}_\sigma$ is a diagonal matrix with the values of $f_\sigma(\sigma_{ln})$ in its diagonal, with f_σ also a nonlinear learnable transformation.

The authors compared several approaches, including meta-learning, for adaptation to changing dynamics in uncertain scenarios. BCA was found superior in this context and faster to train.

Following (Shaj et al., 2022), is possible to use a RKN for probabilistic glucose prediction, and by the incorporation of BCA, patient information can be extracted from past data, in order to achieve model personalization. However, one challenge remains: how to inform the network that the meal input is uncertain. One option is to use BCA to infer the latent distribution of the meals, however, since the meal input signal is mostly constant at zero with only few spikes, this would produce mode collapse of the distribution towards a constant value (Lucas et al., 2019). Other option is to use a variational approximation of the posterior distribution, but that would invalidate the uncertainty propagation with the Kalman equations. Additionally, is also possible to just treat the meal input as deterministic, as a naive application of RKN with BCA would do it. However, in this work we argue that this limits the performance of the model and some knowledge about the uncertain nature of the meal input should be incorporated.

3.5.2. Model design

In the same line of (Becker et al., 2019) and (Shaj et al., 2022), our approach consists in the integration of a Kalman filter with a neural network to propagate the uncertainty

on the latent space of the network. Also, the use of BCA for model personalization with $\mathbf{Y}_n = \mathbf{ME}_n^i$ (the support meal) and all the simplifications in the matrices structures are followed. However, in this case to make the network aware about the uncertain nature of the meal signal, we propose the use of an input and state estimation Kalman filter (ISRKN) (Gillijns & De Moor, 2007) in the latent space of the network, so that, the meal signal is treated as a noisy output whose distribution is to be estimated rather than as an input. This has the advantage that any probabilistic loss to estimate the meal distribution can be used, and due to the use of back-propagation, the uncertain nature of the meals will be encoded in the filter parameters.

The input and state estimation Kalman filter that will be used in our model, has the following form (Gillijns & De Moor, 2007):

$$\hat{\mathbf{d}}(t-1) = \mathbf{M}(t)\left(\mathbf{y}(t) - \mathbf{C}\hat{\mathbf{z}}(t)\right) \quad \text{(Input estimation)}, \tag{3.20}$$

$$\tilde{\mathbf{z}}^*(t) = \hat{\mathbf{z}}(t) + \mathbf{G}(t)\hat{\mathbf{d}}(t-1) \quad \text{(State update P.1)}, \tag{3.21}$$

$$\tilde{\mathbf{z}} = \tilde{\mathbf{z}}^*(t) + \mathbf{K}(t)\left(\mathbf{y}(t) - \mathbf{C}\tilde{\mathbf{z}}^*(t)\right) \quad \text{(State update P.2)}, \tag{3.22}$$

$$\hat{\mathbf{z}}(t+1) = \mathbf{A}(t)\tilde{\mathbf{z}}(t) + \mathbf{B}(t)(\mathbf{u}(t)) \quad \text{(State prediction)}, \tag{3.23}$$

where $\hat{\mathbf{d}}(t-1) \in \mathbb{R}^a$, is the estimated disturbance at time $t-1$, $\mathbf{G}(t) \in \mathbb{R}^{n\times a}$ the interaction matrix between the disturbance and the state, and $\mathbf{M}(t) \in \mathbb{R}^{a\times m}$ and $\mathbf{K}(t) \in \mathbb{R}^{n\times m}$ the two time-varying matrices to be estimated by the filter. A sufficient condition for the existence of an unbiased state estimator for this filter, is that $rank(\mathbf{CG}) = rank(\mathbf{G}) = a$, which implies that $n \geq a$ and $a \geq m$.

We choose $a = m$, since in this particular case, it can be demonstrated that $\mathbf{K}(t) = \mathbf{0}\,\forall t$, and therefore, $\tilde{\mathbf{z}}(t) = \tilde{\mathbf{z}}^*(t)\,\forall t$ (Abooshahab et al., 2022), and thus, the filter is greatly simplified, only needing one state update. Also, similar to (Becker et al., 2019) we choose $n = 2m$.

Additionally, since we can choose the size and structure of $\mathbf{G}(t)$, we chose it as a block constant matrix composed of two diagonal matrices of size $m \times m$, namely,

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_u \\ \mathbf{G}_l \end{bmatrix}, \in \mathbb{R}^{n \times m} \tag{3.24}$$

This, in order to satisfy the sufficient condition by construction, given that we maintain the same diagonal structure for $\mathbf{C}$ than (Becker et al., 2019). These diagonal matrices have associated vectors $\mathbf{g}_u, \mathbf{g}_l \sim N(\mathbf{g}|\mu_{gn}, \sigma_{gn})$, with:

$$\mu_{gn} = f_{g\mu}(\mu_{ln}) \tag{3.25}$$

$$\sigma_{gn} = f_{g\sigma}(\sigma_{ln}) \tag{3.26}$$

Where, $f_{g\mu}$ and $f_{g\sigma}$ are nonlinear learnable transformations and $\mathbf{g} = [\mathbf{g}_u \oplus \mathbf{g}_l]$. This particular dependence on μ_{ln} and σ_{ln}, is motivated by the fact that these vectors carry information about a past meal response of the particular patient, and thus, can help to estimate this gain.

3.5.2.1. Input estimation

For the input estimation (equation 3.20), it can be shown that $\mathbf{M}(t)$ is given by the following expression (Gillijns & De Moor, 2007):

$$\mathbf{M}(t) = \left(\mathbf{F}\tilde{\mathbf{R}}(t)\mathbf{F}\right)^{-1} \mathbf{F}^T \tilde{\mathbf{R}}^{-1}(t) \tag{3.27}$$

With,

$$\mathbf{F} = \mathbf{CG} \tag{3.28}$$

$$\tilde{\mathbf{R}}(t) = \mathbf{C}\hat{\mathbf{\Sigma}}(t)\mathbf{C}^T + \mathbf{R}(t) \tag{3.29}$$

Which with the proposed simplifications, simply leads to a diagonal matrix with associated constant vector $\mathbf{m}_u$, given by:

$$\mathbf{m}_u = \frac{1}{\mathbf{c}_u \mathbf{g}_u} \tag{3.30}$$

Introducing this result in equation 3.20, gives us the following expression for $\hat{\mathbf{d}}(t-1)$

$$\hat{\mathbf{d}}(t-1) = \frac{\hat{\mathbf{w}}(t) - \mathbf{c}_u \hat{\mathbf{z}}_u(t)}{\mathbf{c}_u \mathbf{g}_u} \tag{3.31}$$

With respective variance given by $\left(\mathbf{F}^T(t)\tilde{\mathbf{R}}^{-1}(t)\mathbf{F}(t)\right)^{-1}$, which in our case leads to:

$$\hat{\sigma}_{\hat{\mathbf{d}}}(t-1) = \frac{\mathbf{c}_u \hat{\sigma}_u(t) + \hat{\mathbf{r}}(t)}{(\mathbf{c}_u \mathbf{g}_u)^2} \tag{3.32}$$

Once estimated, both $\hat{\mathbf{d}}(t-1)$ and $\hat{\sigma}_{\hat{\mathbf{d}}}(t-1)$ are transformed back to the original space by an input decoder, so that any probabilistic loss (in our case the negative Gaussian log-likelihood loss) can be used to estimate the input meal distribution. That is,

$$\hat{\mathbf{m}}(t-1) = dec_{\hat{\mathbf{d}}}(\hat{\mathbf{d}}(t-1)) \tag{3.33}$$

$$\hat{\sigma}_{\hat{\mathbf{m}}}(t-1) = dec_{\hat{\sigma}_{\hat{\mathbf{d}}}}(\hat{\sigma}_{\hat{\mathbf{d}}}(t-1)) \tag{3.34}$$

$$L_{meal}(t) = -\log \mathcal{N}(\mathbf{m}(t-1)|\hat{\mathbf{m}}(t-1), \hat{\sigma}_{\hat{\mathbf{m}}}(t-1)) \tag{3.35}$$

3.5.2.2. State update

As for the state update (equation 3.21), the state equation is simplified to:

$$\tilde{\mathbf{z}}_u(t) = \hat{\mathbf{z}}_u(t) + \mathbf{g}_u \hat{\mathbf{d}}(t-1) \tag{3.36}$$

$$\tilde{\mathbf{z}}_l(t) = \hat{\mathbf{z}}_l(t) + \mathbf{g}_l \hat{\mathbf{d}}(t-1) \tag{3.37}$$

On the other hand, the state covariance update has the following form (Gillijns & De Moor, 2007):

$$\tilde{\mathbf{\Sigma}}(t) = (\mathbf{I}_n - \mathbf{GM(t)C})\,\hat{\mathbf{\Sigma}}(t)\,(\mathbf{I}_n - \mathbf{GM(t)C})^T + \mathbf{GM(t)RM}^T(t)\mathbf{G}^T \tag{3.38}$$

Which in our case, gives us the following expressions for the associated vectors of the block covariance matrix (see equation 3.12):

$$\tilde{\sigma}_u(t) = \frac{\hat{\mathbf{r}}(t)}{\mathbf{c}_u^2} \tag{3.39}$$

$$\tilde{\sigma}_s(t) = \frac{\mathbf{g}_l\hat{\mathbf{r}}(t)}{\mathbf{c}_u^2\mathbf{g}_u} \tag{3.40}$$

$$\tilde{\sigma}_l(t) = \frac{\mathbf{g}_l^2}{\mathbf{g}_u^2}\left(\frac{\mathbf{r}(t)}{\mathbf{c}_u^2} + \hat{\sigma}_u(t)\right) - 2\frac{\mathbf{g}_l}{\mathbf{g}_u}\hat{\sigma}_{\mathbf{s}}(t) + \hat{\sigma}_l(t) \tag{3.41}$$

3.5.2.3. State prediction

For the state prediction (equation 3.23), this filter leads to the same equations than (Shaj et al., 2022), that is, equations 3.18 and 3.19. The respective simplifications of which, can be found in (Shaj et al., 2022).

3.5.2.4. Loss calculation

Just as in the deterministic case, the inputs of the network are sliding windows of length τ that are processed iteratively by the filter up to time t. Then, to produce p-step ahead predictions, at each step, the output of the network is sampled and iteratively fed back as a new observation to produce the next-step ahead prediction, that is:

$$\tilde{\mathbf{y}}(t+k) \sim \mathcal{N}(\tilde{\mathbf{y}}(t+k)|\hat{\mathbf{y}}(t+k), \hat{\sigma}(t+k)) \tag{3.42}$$

$$\mathbf{y}(t+k) \approx \tilde{\mathbf{y}}(t+k), \forall k \in [1, p] \tag{3.43}$$

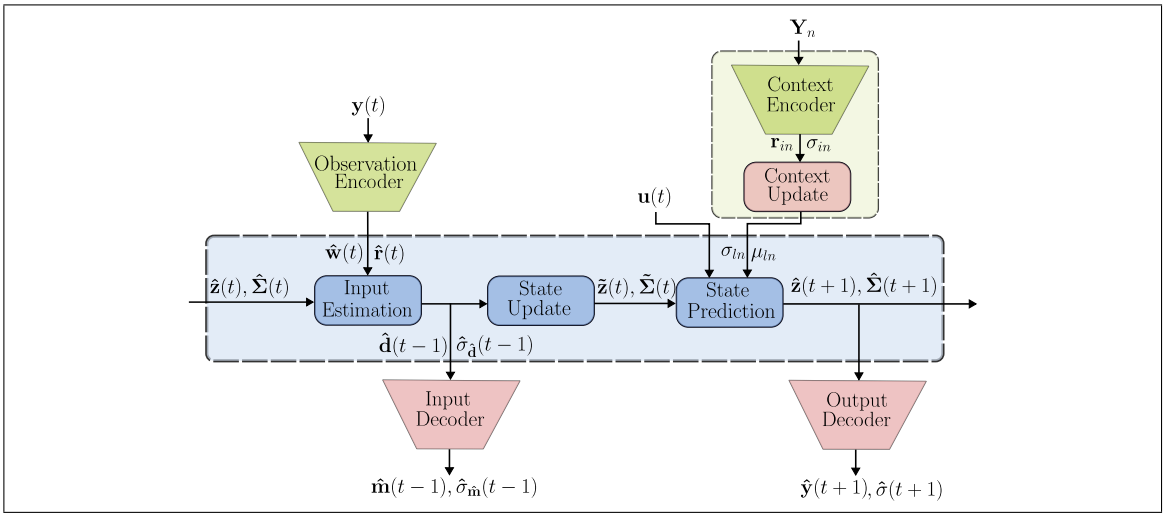


Figure 3.8. Proposed recurrent Kalman network with input estimation (IS-RKN). In this case the input estimation step produce two vectors, namely, $\hat{\mathbf{d}}(t-1)$ and $\hat{\sigma}_{\hat{\mathbf{d}}}(t-1)$, which are transformed by the input decoder, such that the meal distribution can be estimated.

Then, the final loss is given by:

$$L = \frac{1}{p}\sum_{k=1}^{p} L_y(t+k) + \frac{1}{p+\tau}\sum_{k=1-\tau}^{p} \gamma(t+k) L_{meal}(t+k) \tag{3.44}$$

Where the meal loss is calculated over all the steps of the filter, and the target loss only over the predictions. Also, $\gamma(t+k)$ is a weight function with $\gamma(t) = \eta, \forall t \in [\hat{t}_{meal} - \hat{\delta}_t : \hat{t}_{meal} + \hat{\delta}_t]$, where $\hat{t}_{meal}$ is the time of the meal announced by the patient and $\hat{\delta}_t$ is chosen as 20 minutes. For the rest of the sequence, $\gamma(t) = \frac{\eta}{100}$ with $\eta = 1$. This structure was followed in order to account for the meal time uncertainties.

Finally, figure 3.8 shows the structure of the proposed model for probabilistic glucose prediction with all the elements described above.

3.5.3. Experiments

For the experiments, exactly the same data preprocessing than in the deterministic case is followed (see figure 3.1), since it was seen that led to better personalization performance.

However, in the case of the RKN models, the support meals are processed in the BCA, and no inner loop updates are required.

Also, the proposed Out-Of-Distribution testing procedure was used to test the generalization capabilities of the proposed model, but using probabilistic performance metrics.

3.5.3.1. Data-set generation

The data-set was generated following the same procedure described in section 3.4.1. However, in this case the only sources of randomness were related to the meal time, and meal size misspecifications by the patient, following the distributions found by (Robinson et al., 2021) and (Brazeau et al., 2012). That is, a meal time misspecification occurred with a probability of 0.45, and when occurred, the following uniform distribution for the delay δ_t was used (Robinson et al., 2021):

$$\delta_t \sim Uniform(-5, 20)[min] \tag{3.45}$$

Therefore, the time of the meal announced by the patient is given by:

$$\hat{t}_{meal} = t_{meal} + \delta_t \tag{3.46}$$

Where t_{meal} is the real time of the meal, which is given as an input to the simulator, and $\hat{t}_{meal}$ is the time of the meal announced by the patient, which is given to the network in the input data.

On the other hand, for the meal size misspecification, the following error distribution was used (Brazeau et al., 2012):

$$|e_{meal}| \sim max(\mathcal{N}(0.209, 0.097), 0) \tag{3.47}$$

$$sign_e = \begin{cases} 1 & s \sim Bernoulli(0.38) > 0 \\ -1 & s \sim Bernoulli(0.38) = 0 \end{cases} \tag{3.48}$$

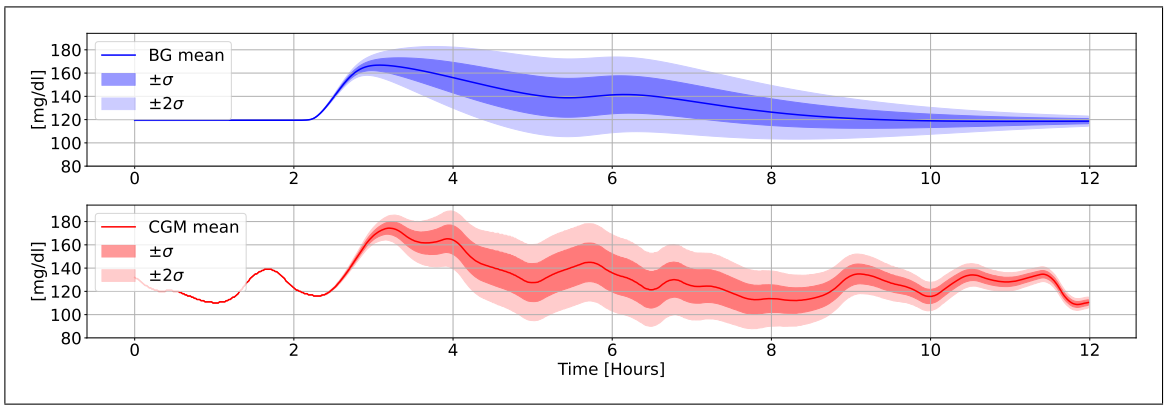


Figure 3.9. Glucose distribution for Adult#003 when the meal input is uncertain. In this case a meal of size $M_s = 60$[gr] is given at time $t_{meal} = 2$[Hours]. However, the insulin bolus is given at $\hat{t}_{meal}$ (equation 3.46) and calculated using $\hat{M}_s$ (equation 3.49). The upper graph shows how BG (unmeasured) is distributed and the lower graph shows how CGM (measured) is distributed. The experiment is repeated 120 times to approximate the moments of the distribution.

With this, the meal size estimated by the patient is given by:

$$\hat{M}_s = M_s(1 + sign_e|e_{meal}|) \tag{3.49}$$

Where M_s is the real size of the meal, which is given as the input to the simulator, and $\hat{M}_s$ is the meal size estimated by the patient, which is used to calculate the insulin bolus introduced in the simulator and is the meal size given to the models.

Figure 3.9 shows how BG and CGM vary for one of the patients of the simulator when these distributions are used to model meal uncertainty.

3.5.3.2. Baselines

The chosen baselines for this task are the following:

- **MetaL1$_{\text{prob}}$**: As the first baseline, we used the deterministic personalizable model based on meta-learning proposed above, but with an uncertain meal input. In this case, the training procedure remains the same, however, during testing we

assume that the meal input distribution is known, so that different meal values are sampled from this distribution and the obtained predictions are used to approximate the glucose output distribution.

- **RKN$_{\text{BA}}$**: In this case, we used the model proposed by (Shaj et al., 2022) which includes BA for model personalization and the meal signal as an external input along with the insulin. This model is used to compare how our model performs against a similar model which treats the meal input as deterministic.
- **RKN$_{\text{BANM}}$**: This model is similar to **RKN$_{\text{BA}}$**, however, in this case the meal signal is ignored and only the insulin is given as the external input.
- **Gaussian Process (GP)**: Finally, a Gaussian process regressor is used to compare our model with a strong classical probabilistic model. The GP is trained from scratch for each patient, both with population training data and all the support data of the corresponding patient in order to personalize.

3.5.3.3. Evaluation metrics

As for the probabilistic evaluation metrics, the main metric considered was the Expected calibration error (ECE), which according to several authors is the most important metric for evaluating probabilistic models (C. Guo et al., 2017). The ECE is the expected difference between the confidence predicted by the model and the real percentage of the data that falls on the predicted boundaries. Mathematically,

$$ECE = \frac{1}{B}\sum_{b=1}^{B} |\alpha_b - P_{\alpha^b}^{\hat{\sigma}}(D)| \tag{3.50}$$

Where α_b is a chosen level of confidence and $P_{\alpha^b}^{\hat{\sigma}}(D)$ is the probability that the ground truth data falls inside the boundaries established by the confidence level α_b and the predicted standard deviation $\hat{\sigma}$. In our case, we chose ten different values of confidence from 0% to 100%, separated by 10%, and the predicted boundaries are obtained by assumming a Gaussian distribution of the error.

The second metric used for the evaluation is the Maximum Calibration Error (MCE) (C. Guo et al., 2017), which is important in high-risk applications where we may wish to minimize the worst case-deviation. The MCE is the maximum deviation of the averaged terms in the ECE, that is,

$$MCE = \max_{b \in [1,B]} |\alpha_b - P^{\hat{\sigma}}_{\alpha^b}(D)| \tag{3.51}$$

A third metric is the sharpness (SHA), which refers to the concentration of the predictive distributions. The more concentrated the predictive distributions are, the sharper the predictions, and the sharper the better, subject to calibration (Gneiting et al., 2007). In practice, the sharpness is measured as the length of the interval between the boundaries established by a confidence level and a predicted uncertainty. In our case, sharpness will be measured only at two levels of confidence, namely 68% ($\pm\hat{\sigma}$) and 95% ($\pm 2\hat{\sigma}$).

The fourth metric is the average RMSE (see equation 3.7) to measure how far the predicted mean of the distribution is from the ground truth data.

Finally, the average negative Gaussian log-likelihood (NGLL) of the predicted sequences will be also used, which considers both the predicted mean and the predicted uncertainty. Mathematically,

$$NGLL = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{p}\sum_{k=1}^{p}\frac{1}{2}\left(\log\left(\hat{\sigma}(t+k)\right) + \frac{(\hat{\mathbf{y}}(t+k) - \mathbf{y}(t+k))^2}{\hat{\sigma}(t+k)}\right) \tag{3.52}$$

3.5.3.4. Experimental results

Table 3.8 shows the results of the OOD evaluation procedure for probabilistic glucose prediction. In this case, all the experiments were conducted for only one prediction horizon $p = 60$[min].

From the table, is possible to see that all the Recurrent Kalman Network models are far superior than the other baselines. On one hand, the MetaL1$_{prob}$, even when is very accurate in terms of RMSE$_{avg}$, tends to be overconfident of its predictions, which translates in very

Table 3.8. OOD evaluation results for probabilistic glucose prediction at horizon $p = 60[min]$. Numbers in bold represent the best result for each metric.

$G1$						
Model	**ECE** [%]	**MCE** [%]	**NGLL**	**SHA**$_{0.68}$ [mg/dl]	**SHA**$_{0.95}$ [mg/dl]	**RMSE**$_{avg}$ [mg/dl]
MetaL1$_{prob}$	44.01	71.76	128.30	1.78	3.50	**7.05**
GP	8.19	14.07	4.29	45.57	89.81	26.05
RKN$_{BA}$	5.31	8.31	3.64	22.37	44.09	10.45
RKN$_{BANM}$	4.68	7.26	3.64	21.64	42.66	10.36
ISRKN	**1.90**	**3.29**	**3.56**	**18.3**	**36.07**	9.83
$G2$						
Model	**ECE** [%]	**MCE** [%]	**NGLL**	**SHA**$_{0.68}$ [mg/dl]	**SHA**$_{0.95}$ [mg/dl]	**RMSE**$_{avg}$ [mg/dl]
MetaL1$_{prob}$	47.38	77.48	300.47	1.49	2.94	**9.35**
GP	7.75	14.97	4.81	50.14	98.83	33.87
RKN$_{BA}$	1.96	3.75	3.85	23.99	47.29	13.01
RKN$_{BANM}$	2.67	7.15	3.81	23.13	45.59	12.55
ISRKN	**1.09**	**2.47**	**3.69**	**18.99**	**37.44**	11.5
$G3$						
Model	**ECE** [%]	**MCE** [%]	**NGLL**	**SHA**$_{0.68}$ [mg/dl]	**SHA**$_{0.95}$ [mg/dl]	**RMSE**$_{avg}$ [mg/dl]
MetaL1$_{prob}$	47.56	78.3	785.19	1.59	3.14	**10.99**
GP	7.5	13.69	4.82	50.49	99.51	34.01
RKN$_{BA}$	3.46	5.96	3.92	24.41	48.11	14.57
RKN$_{BANM}$	2.21	4.14	3.88	23.04	45.41	13.66
ISRKN	**1.97**	**3.69**	**3.78**	**20.6**	**40.6**	12.51
$G4$						
Model	**ECE** [%]	**MCE** [%]	**NGLL**	**SHA**$_{0.68}$ [mg/dl]	**SHA**$_{0.95}$ [mg/dl]	**RMSE**$_{avg}$ [mg/dl]
MetaL1$_{prob}$	47.73	78.76	749.09	1.59	3.12	**10.79**
GP	7.47	13.68	4.83	50.38	99.3	34.13
RKN$_{BA}$	3.43	5.78	3.94	24.11	47.51	14.42
RKN$_{BANM}$	2.9	5.05	3.92	23.32	45.97	14.02
ISRKN	**2.71**	**4.56**	**3.87**	**20.58**	**40.56**	13.26

narrow prediction distributions (see SHA$_{0.68}$ and SHA$_{0.95}$) but a very high ECE, MCE and NGLL. In fact, when plotting the calibration error for the different confidence values (see figure 3.10), it is possible to see how far the MetaL1$_{prob}$ predicted distribution is from the empirical distribution of the data. On the other hand, the GP even when it has low ECE, MCE and NGLL values, tends to be very under-confident of its predictions (see figure 3.10), which translates in a very wide predicted distribution, and also, the mean tends to be very far from the ground-truth data, as it can be seen from the RMSE$_{avg}$ metric.

As for the RKN$_{BA}$ and RKN$_{BANM}$ both models have very good performance across all metrics, with RKN$_{BANM}$ being slightly better than RKN$_{BA}$. This shows that handling the uncertain meal signal as a deterministic input could harm model performance, and is better to just ignore it. However, it is worth noting that the insulin input already contains some information of the estimated meal size, since the insulin bolus is calculated using this

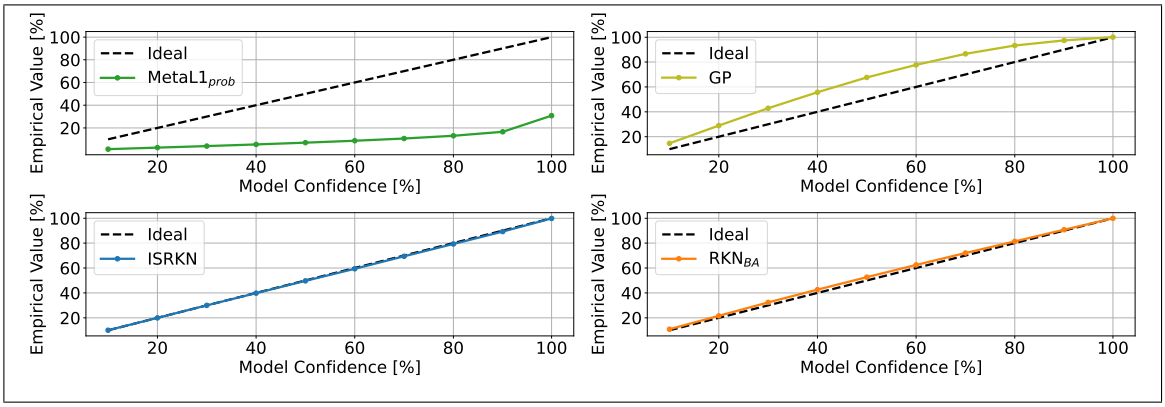


Figure 3.10. Calibration error for the different probabilistic models using data from one patient in G4. The x-axis shows the confidence of the model that the ground-truth data fall within the boundaries associated to the estimated standard deviation of the predicted distribution. The y-axis is the percentage of the ground-truth data that actually fall inside those boundaries.

value. Therefore, is possible that the RKN$_{BANM}$ is extracting all the necessary information from the insulin and past data, and the uncertain meal input becomes unnecessary.

Regarding the ISRKN, it is observed that this model is superior to all others in all probabilistic metrics for all groups. Only MetaL1$_{prob}$ is superior in terms of RMSE$_{avg}$ and in SHA$_{0.68}$ and SHA$_{0.95}$, but this last two metrics are subject to a valid calibration, therefore their values are discarded for this model. From the table and from figure 3.10 it is possible to see that the proposed model is the one that better estimates the empirical distribution of the data, also is the model with tighter predicted distributions and has competitive performance in terms of RMSE$_{avg}$ when compared to the MetaL1$_{prob}$.

Finally, figure 3.11 shows a prediction of the ISRKN for a randomly selected meal. It can be seen that the model is able to accurate predict the glucose signal with its corresponding distribution, and is able to correctly estimate the time and size of the meal, even when this is just a secondary task. This shows that the network is able to extract more meaningful information from the meal uncertain input when treating it as an uncertain output whose distribution is to be estimated, instead of as a deterministic input or just ignore it . Also, this shows that the ISRKN could eventually be used to estimate the meal

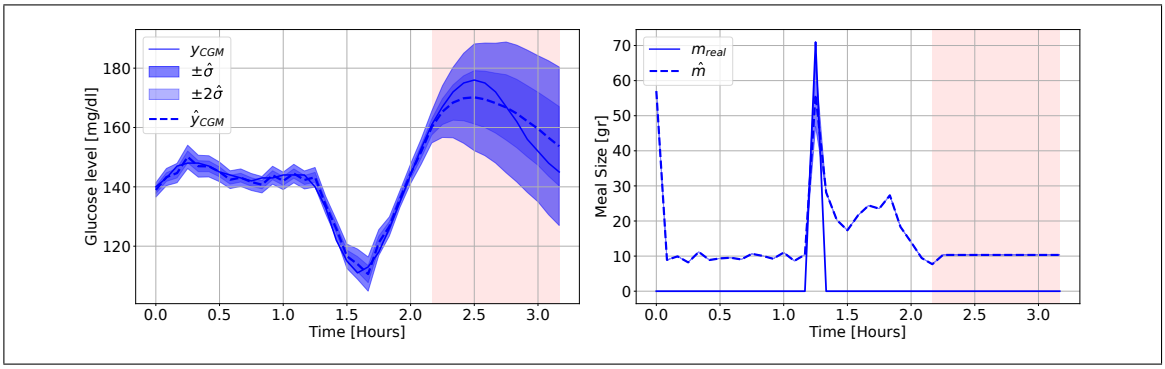


Figure 3.11. Example prediction of the ISRKN for an arbitrary patient in group G4. The solid and the dashed lines are the ground-truth data and the predictions of the network, respectively. The white background indicates the context data given to the network, that is, data for a time $k \leq t$ (see equation 3.44) and the light-red background indicates the prediction time ($k > t$).

characteristics from the glucose signal and some meal labels, which could help to fully automate the AP, but this is left as future work.

3.6. Discussion

In this chapter and adaptative-learning based model capable of personalizing to different T1D patients with minimal amount of additional data and with low risk of overfitting by the use of meta-learning is presented first. This model not only showed superior performance than other ML or DL population models trained with the same amount of data, but also it was shown how our model achieves better performance with less patient-specific data than other adaptative methods as transfer learning. This is important in safety critical applications as T1D prediction and control in which low predictive performance at any point could be fatal.

Additionally, a novel probabilistic model is proposed to estimate the uncertainty produced by the user's misspecification of meal time and size, a very common problem in T1D that limits the performance of modern APs. The model is based on a Recurrent Kalman network along with an input and state estimation Kalman Fiter, such that the meal

input is considered as an uncertain output variable whose distribution is to be estimated by the filter. Results show that this model achieves superior performance than other techniques in several probabilistic metrics and can correctly estimate the output distribution of the glucose.

Future work includes using both modeling approaches along with a model-based controller in closed loop, and possibly performing clinical trials with these data-driven systems.

4. NEUROEVOLUTIVE CONTROL OF INDUSTRIAL PROCESSES THROUGH MAPPING ELITES

This Chapter focuses on the development of a neuroevolutive controller with learning capabilities and great flexibility during training and during execution, meant to be used in a wide range of iCPSs applications [1].

4.1. Context

Among the different techniques used in process control, model-based control has seen increasing usage in the last decades, in particular model predictive control (MPC), showing great success in oil, paper and pulp, and mining processes (Qin & Badgwell, 2003). A traditional MPC setup involves three elements (Rawlings & Mayne, 2009): i) a dynamic model of the process used to predict the effect of the inputs on the process outputs; ii) an optimizer in charge of calculating a control sequence to drive the output to a desired reference, based on a cost function involving the inputs, the outputs and their set-points; and iii) the use of the receding horizon strategy, which consists in applying only the first element of the control sequence and repeating the optimization at each time-step. In typical MPC approaches the search for control sequences is done directly in the subspace spanned by the manipulated variables, which represents a high computational burden, particularly when using a complex nonlinear model.

Recently, learning-based techniques that instead of calculating a control sequence look to generate a controller in the form of a parameterized function have shown important progress. These techniques perform an optimization over the subspace spanned by the parameters of a candidate function, and their main advantage over typical model-based approaches is that when a good-performing controller is found, there is no need to conduct continuously the optimization process, in fact, in the limit, optimization can be stopped,

[1]The material in this chapter has been published in the journal IEEE Transactions On Industrial Informatics (Langarica & Núñez, 2021).

which significantly reduces the computational cost. Two popular techniques from this class are Reinforcement Learning and Neuroevolution (NE).

RL refers to a set of goal-oriented algorithms inspired in how agents learn by interacting with the environment and repeating actions that result in a high reward, while avoiding actions that lead to punishments. RL algorithms have been known for decades; however, since the work in (V. Mnih et al., 2015) RL has gain attention in the scientific community. In (V. Mnih et al., 2015), RL agents were trained to play challenging Atari games by only receiving video frames as input. As a result, they were able to defeat human experts. The success of (V. Mnih et al., 2015) is due to the use of complementary techniques as experience replay (O'Neill et al., 2010) and a deep neural network (DNN) as a nonlinear approximator of the action-value function, despite it was thought that this was not feasible because of stability issues (Tsitsiklis & Van Roy, 1997).

Since the reported success in (V. Mnih et al., 2015), many other works combining RL algorithms with DNNs have appeared, giving rise to a new research area called Deep Reinforcement Learning (DRL). Most of these works focus on new techniques for training DRL agents for playing and solving Atari games (van Hasselt et al., 2015), board games (Silver et al., 2016) and robotic control tasks (Lillicrap et al., 2015). Despite the great success of DRL in these complex tasks, its adoption for solving real-world problems is still limited, mainly due to three major challenges: i) temporal credit assignment with long time horizons; ii) lack of diverse exploration; and iii) brittle convergence properties (Khadka & Tumer, 2018). The first may lead to the agent not being able to find an optimal solution, the second to the agent getting stuck in a local optima, and the last may produce the system to go unstable, which is not acceptable in real control applications. Recently, efforts from the RL community to address these challenges have been made, as (Puigdomènech Badia et al., 2020), that uses novel techniques like Never give up (Badia et al., 2020), and (Kamthe & Deisenroth, 2017) which also addressed optimality and stability.

An alternative effective approach to address these challenges (Khadka & Tumer, 2018) is the use of evolutionary algorithms (EA) (Spears et al., 1993), in particular NE, which

consists in training DNNs using EA for finding optimal policies directly in the subspace spanned by the weights of a DNN (Gomez, 2003). NE helps solving the aforementioned challenges by: i) using a fitness metric that consolidates returns across time, making EA indifferent to the sparsity of the reward (Salimans et al., 2017); ii) implementing a population-based approach that intrinsically allows diverse exploration, when a proper technique for maintaining diversity in the population is used (Lehman & Stanley, 2011); and iii) exploiting the redundancy in a population, which results in more robust and better convergence properties (Ahn & Ramakrishna, 2003) and potentially improves stability in a control application. Thanks to these favorable properties, NE algorithms have been applied successfully in different RL tasks (Salimans et al., 2017; Gangwani & Peng, 2017; Conti et al., 2017).

Nonetheless, there are still two major issues that have to be addressed when applying NE to real-world problems. First, NE algorithms do not solve the high sample complexity that DRL algorithms have, which means that multiple interactions with the environment are still needed to learn a good policy, which is not always possible in real world applications. Secondly, these algorithms need to maintain a large population to ensure diversity and avoid getting stuck in local optima, which is a potential disadvantage and represents a huge memory requirement since each individual represents a neural network (NN) with, possibly, thousands or millions of parameters.

In this work, we propose a control strategy that combines elements from MPC, DRL, and NE. The outputs of our main algorithm are optimized neural controllers that use a quadratic fitness function. To tackle existing implementation problems several contributions are made. First, to reduce sample complexity, we propose to use a neural model of the process, supported by the evidence that model-based RL substantially improves sample efficiency of DRL algorithms (Kaiser et al., 2019). Secondly, to maintain a large population of individuals without increasing computational load, we propose a novel compression algorithm for DNNs based on the ideas of (Petroski Such et al., 2017), improved with the use of a modification of the linear congruential generator algorithm (Bolte, 2011),

which allows representing a DNN of arbitrary size with only three parameters. Using this compression algorithm, we can maintain large populations of neural agents in a desktop computer, and by using the Map Elites evolutionary optimization algorithm (Mouret & Clune, 2015), we can ensure diverse populations.

Consequently, the main contributions of this chapter are three-fold: i) a model-based RL algorithm suitable for real industrial process control is proposed; ii) a novel compression algorithm for DNNs is developed; and iii) a modified version of the Map Elites optimization algorithm, suitable for real industrial process control applications is presented.

4.2. Background

4.2.1. Model-based Reinforcement learning

Model-based RL considers the use of a model to simulate the interactions between the environment and the agent. The idea behind using a model is to reduce the number of interactions with the real environment an agent needs to learn a good policy. If the model is reliable, model-based RL results in faster learning, since there is no need to wait for the environment to respond, instead, interactions are simulated.

A well-known model-based RL algorithm is Dyna (Sutton, 1991), which consists in a RL agent that interacts with the real environment and a model of it. At each iteration, the agent first interacts with the model several times to accumulate experience and improve its policy, and then faces the real environment using the learned policy. Experience from the real interaction is transferred indirectly by the periodic update of the model with this information. For Dyna to work properly, the sampling period of the real environment has to be much larger than the time each simulated interaction takes.

4.2.2. Value and Policy approximation methods

RL algorithms can be classified in two major families: value and policy approximation methods. The first family of algorithms focuses on finding the optimal policy by estimating the value function, which tabulates how good is for an agent to be in a given state or how good is to perform a given action in a given state (in action-value functions) (Sutton & Barto, 2014).

The value of a state s under a policy π is the expected discounted return when starting in s and following π thereafter. Mathematically,

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \tag{4.1}$$

where, $\mathbb{E}_\pi$ is the expected value operator given that the agent follows a policy π, γ stands for the discounted factor and R is the reward that the agent receives for being in state s. In practice, in value approximation methods, the action-value function is used, this function represents the value of taking an action a in a state s under a policy π and is given by,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \tag{4.2}$$

Q-learning (Sutton & Barto, 2014) is probably the most popular value approximation method, which tries to approximate $q(s, a)$ with a parametric function $\hat{q}_\theta(s, a)$ with learnable parameters. Q-learning has been used successfully in many applications (V. Mnih et al., 2015; van Hasselt et al., 2015), however, due to its cost, is limited to discrete action spaces, which limits its applicability in control-oriented tasks.

The second family of RL methods focuses on directly approximating the policy π by using a parametric function $\hat{\pi}_\theta$. The function $\hat{\pi}_\theta$ directly approximates the mapping between state and actions, which can be continuous. Hence, these methods can deal with continuous and stochastic action spaces, which is ideal for control purposes. However, when using gradient-based methods, like backpropagation, to optimize $\hat{\pi}_\theta$, it is necessary

to use a second approximator for $q(s, a)$ that allows to map gradients from rewards to actions and only then, gradients can be backpropagated to $\hat{\pi}_\theta$. Therefore, there is a trade-off between the advantages of policy approximation algorithms and the computational burden of maintaining and optimizing two function approximators. Actor-critic methods are an example of policy approximation algorithms (Lillicrap et al., 2015).

NE algorithms are a natural extension of policy approximation methods that exploit the fact that EA do not need to use gradients for optimization, which eliminates the necessity of a second approximator. However, maintaining a large population of policy approximators can be prohibitive in terms of memory.

4.2.3. Evolutionary Algorithms

EA are a set of black-box optimization methods used for solving optimization problems involving functions that are either non-differentiable or cannot be expressed in closed form (Mouret & Clune, 2015). EA have been used extensively in engineering systems due to its simplicity and effectiveness (Antonio & Coello, 2018; Choi et al., 2016).

In its basic form, these algorithms evolve a population P of N individuals. In case of NE, each individual represents a NN with parameter vector θ, often called genotypes. At each generation, each individual is evaluated and a fitness score, which is intended to be maximized, is associated to each individual. From these, a fraction E of the highest performing individuals passes to the next generation directly (called the elites). A fraction M suffers a process called mutation, that generally involves perturbing the parameters with noise. A fraction C suffers a cross-over, which is a permutation of parameters between two individuals called the parents. To complete the set passing to the next generation, individuals are chosen randomly with a probability proportional to their fitness score. This procedure is repeated until a stopping criterion is reached. This basic EA is presented in Algorithm 2.

Algorithm 2 Basic Evolutionary Algorithm

Given P: Population, N:Population size, E: Percentage of Elites, M: Percentage of mutated individuals, σ: Mutation level, C: Percentage of cross-over individuals F: fitness function and Δ termination criterion.
function EVOLUTIONARY ALGORITHM(N, E, M, C, σ, F, Δ)
    Initialize Population P of N individuals with parameters θ_i randomly.
    Initialize next generation list L as an empty list of individuals.
    while Δ is not reached **do**
        Empty list L
        for θ_i in P **do**
            $f_i = F(\theta_i)$
        end for
        Select $\%$ E of individuals and save them in L for the next generation
        Select $\%$ M of individuals, perturb its parameters with $\theta_i = \theta_i + \epsilon$, where ϵ $N(0, \sigma)$ and save them in L.
        Select $\%$ C pairs of individuals and randomly change some of its parameters $\theta_{i,r} = \theta_{j,r}$, $\theta_{j,r} = \theta_{i,r}$, where $\theta_{i,r}$ is the parameter at position r of individual i.
        while Length of $L < N$ **do**
            Select an individual θ_i with probability $p\ \alpha\ F(\theta_i)$ and save it in L
        end while
        $P = L$
    end while
    Return $\theta_{best} = argmax_\theta F(L)$
end function

4.2.4. Illumination algorithms and Map Elites

Classic EA are designed to return a small subset of high performing solutions that are optimal for the optimization objective, represented by the fitness function. In (Mouret & Clune, 2015) a more general type of algorithms, called illumination algorithms, are introduced. These algorithms are designed to return the highest-performing solution at each point in a feature space, thus illuminating the fitness potential of each region of this space. Here, each feature can be any measurable characteristic, related or not to the fitness function. In particular, in (Mouret & Clune, 2015), an illumination algorithm called Map Elites is proposed, which has as its major advantage over EA and other illumination algorithms that it is designed to facilitate the visualization of the optimization in the feature space. Map Elites allows viewing how fitness is distributed over a feature space that is low-dimensional and meaningful by design. With this information not only the best solution in terms of fitness can be chosen, but also a high performing solution with desired

characteristics depending on the application. In terms of process control, a more aggressive or mild controller can be selected without re-tuning the fitness function, facilitating controller design.

In Map Elites, as in other EAs, first, a fitness function $F(\theta_i)$ has to be chosen to evaluate each individual θ_i. Next, T dimensions of variation representing the selected features have to be chosen to define the feature space. Then, each dimension of variation is discretized. Given a particular discretization, Map Elites will search for the highest performing solution for each cell in the feature space grid. It should be noted that the search for solutions is done in the subspace containing the genotypes of each individual, that is, the parameters that represent each individual and then, by evaluating the selected features, each individual is positioned in the feature space.

The algorithm starts by generating G initial individuals, measuring the features and positioning each one in the corresponding cell on the grid. Then, the following steps are repeated until the termination criterion is reached: i) take a fraction $M + C$ of individuals from the population and produce a fraction of M new individuals by mutation and a fraction C by cross-over. ii) Measure the features of each new gene and place it in its corresponding cell if it is empty and replace the current occupant of the cell only if the new gene has better performance. Algorithm 3 presents the Map Elites algorithm.

As shown in (Mouret & Clune, 2015), Map Elites performs better than other EA in terms of performance, reliability, precision and coverage.

Algorithm 3 Map Elites Algorithm

Given G: initial number of genes, T: Dimensions of variation F: fitness function, B: features function, M: Fraction of individuals taken for mutation, σ: Mutation level, C: Fraction of new individuals created by Cross-over . Δ termination criterion.
function MAP ELITES ALGORITHM(G, T, F, B, n, m, σ, M, C,Δ)
    Create a T-dimensional map of elites E
    Create L as an empty list of genes
    Generate G initial genes θ_i and save them in L
    for θ_i in L **do**
        $b_i = B(\theta_i)$ ▷ Calculation of features
        $f_i = F(\theta_i)$ ▷ Calculation of fitness
        Place θ_i in position E_{b_i} according to its features b_i
    end for
    Empty L
    while Δ is not reached **do**
        Select a fraction M of individuals and create new genes by perturbing parameters with $\theta_i = \theta_i + \epsilon$, where ϵ $N(0, \sigma)$ and save them in L.
        Select a fraction C of individuals and create new genes by randomly changing some of its parents parameters $\theta_{i,r} = \theta_{j,r}$, $\theta_{j,r} = \theta_{i,r}$, where $\theta_{i,r}$ is the parameter at position r of individual i and save them in L
        for θ_i in L **do**
            $b_i = B(\theta_i)$ ▷ Calculation of features
            $f_i = F(\theta_i)$ ▷ Calculation of fitness
            if The position in the grid E_{b_i} is empty **then**
                Place θ_i in position E_{b_i}
            else if The position in the grid E_{b_i} is occupied by gene θ_j with fitness f_j **then**
                if $f_i > f_j$ **then**
                    Place θ_i in position E_{b_i}
                else if $f_i \leq f_j$ **then**
                    Retain θ_j in position E_{b_i} and discard gene θ_i
                end if
            end if
        end for
    end while
return E
end function

4.3. Neuroevolution for process control

4.3.1. Problem statement

Consider a generic non-linear time-invariant system Σ:

$$\Sigma : \begin{cases} \dot{\mathbf{z}} = f(\mathbf{z}, \mathbf{u}), & \mathbf{z} \in Z \subseteq \mathbb{R}^K, \mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^M \\ \mathbf{y} = h(\mathbf{z}, \mathbf{u}), & \mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^N \end{cases}, \tag{4.3}$$

where f and h are smooth non-linear mappings. Assume a sampling process exists where inputs $\mathbf{u}$ and outputs $\mathbf{y}$ are sampled at constant period τ to generate the M-dimensional input sequence $\mathbf{u}$ and the N-dimensional output sequence $\mathbf{y}$.

We aim to find the parameters of a controller θ_t that produces an M-dimensional output sequence $\hat{\mathbf{u}}_\theta$ given by

$$\hat{\mathbf{u}}_\theta(t) = \theta_t(\mathbf{U}(t-1), \mathbf{Y}(t)), \tag{4.4}$$

where $\mathbf{U}$ and $\mathbf{Y}$ are N_u-depth and N_y-depth windows of $\mathbf{u}$ and $\mathbf{y}$ respectively, so that when applying $\hat{\mathbf{u}}_\theta$ to Σ, the controlled variables $\mathbf{y}$ are headed towards a reference sequence.

Assuming that an accurate enough model

$$\hat{\mathbf{y}}(t+1) = M_t(\mathbf{U}(t), \mathbf{Y}(t)) \tag{4.5}$$

of the real system Σ exists, so that $\hat{\mathbf{y}}(t+1)$ effectively approximates $\mathbf{y}(t+1)$, we will optimize the controller parameters using the model and a modified version of Map Elites.

4.3.2. Map Elites for control of dynamical systems

When applying Map Elites for controlling dynamical systems, some modifications to the original formulation have to be done, mainly because of the involvement of time.

Consider a grid G, that for simplicity and without loss of generality will be taken as a two-dimensional grid $I \times J$, and a population $P(t)$ of individuals at sampling time t, in

this case NNs, represented by θ^k, where $k \in \mathbb{Z}_{\geq 0}$ represents the number of mutations the original θ^1 network has suffered. The output of θ^k at time t is given by

$$\hat{\mathbf{u}}_{\theta^k}(t) = \theta^k(\mathbf{U}(t-1), \mathbf{Y}(t)). \tag{4.6}$$

Then, the prediction of model M_t using $\hat{\mathbf{u}}_{\theta^k}(t)$ as input will be denoted as

$$\hat{\mathbf{y}}_{\theta^k}(t+1) = M_t(\hat{\mathbf{U}}_{\theta^k}(t), \mathbf{Y}(t)), \tag{4.7}$$

where

$$\hat{\mathbf{U}}_{\theta^k}(t) = [\mathbf{u}(t-N_u+1), \cdots, \mathbf{u}(t-1), \hat{\mathbf{u}}_{\theta^k}(t)]. \tag{4.8}$$

We can then define

$$\hat{\mathbf{Y}}_{\theta^k}(t+1) = [\mathbf{y}(t-N_y+2), \cdots, \mathbf{y}(t), \hat{\mathbf{y}}_{\theta^k}(t+1)]. \tag{4.9}$$

And then give these matrices as input to θ^k in the following step. Repeating this process for arbitrary horizons, which for simplicity and without loss of generality will be chosen in the following as N_y and N_u for $\hat{\mathbf{Y}}_{\theta^k}$ and $\hat{\mathbf{U}}_{\theta^k}$, will generate

$$\hat{\mathbf{Y}}_{\theta^k}(t+N_y) = [\hat{\mathbf{y}}_{\theta^k}(t+1), \cdots, \hat{\mathbf{y}}_{\theta^k}(t+N_y)], \tag{4.10}$$

and

$$\hat{\mathbf{U}}_{\theta^k}(t+N_u-1) = [\hat{\mathbf{u}}_{\theta^k}(t), \cdots, \hat{\mathbf{u}}_{\theta^k}(t+N_u-1)]. \tag{4.11}$$

It should be noted that necessarily $N_y \geq N_u$; therefore, to produce the remaining elements of $\hat{\mathbf{Y}}_{\theta^k}(t+N_y)$, the last element of $\hat{\mathbf{U}}_{\theta^k}(t+N_u-1)$ must be repeated $N_y - N_u$ times.

These two matrices will be used to calculate the fitness and the features that represent network θ^k in the Map Elites algorithm. The fitness is calculated as

$$f_{\theta^k} = V(\hat{\mathbf{U}}_{\theta^k}(t+N_u-1), \hat{\mathbf{Y}}_{\theta^k}(t+N_y)), \tag{4.12}$$

where $V : \mathbb{R}^{M \times N_u} \times \mathbb{R}^{N \times N_y} \to \mathbb{R}$ is an arbitrary mapping. Similarly, features are calculated as

$$b_{\theta^k} = B(\hat{\mathbf{U}}_{\theta^k}(t+N_u-1), \hat{\mathbf{Y}}_{\theta^k}(t+N_y)), \tag{4.13}$$

where $B : \mathbb{R}^{M \times N_u} \times \mathbb{R}^{N \times N_y} \rightarrow \mathbb{R}^2$ is an arbitrary mapping, not necessarily related to mapping V.

Given the features b_{θ^k} representing θ^k, a function $W : \mathbb{R}^2 \rightarrow I \times J$ maps each network to its corresponding position (i, j) in the grid G. A network after being assigned a position in the grid will be denoted by

$$\theta^k_{(i,j)} = W(b_{\theta^k}), \tag{4.14}$$

and its fitness and features by $f_{\theta^k_{(i,j)}}$ and $b_{\theta^k_{(i,j)}}$.

4.3.3. Compression and Mutation

EAs that use NNs generally store all the parameters of each network in the population. This causes a huge memory demand, since each network may have thousands of parameters, and also the population has to be large enough to ensure diversity. To address this issue, we propose a new compression algorithm for NNs that achieves state of the art performance. However, it can compress only networks evolved with EA.

Our compression algorithm is inspired by the ideas in (Petroski Such et al., 2017), where each network is compressed by storing only the random seeds used to generate its parameters. Then, when a mutation takes place, a recursive strategy is used:

$$\theta^k = \psi(\theta^{k-1}, \tau^k_\theta) = \theta^{k-1} + \sigma\epsilon(\tau^k_\theta), \tag{4.15}$$

where $\epsilon \sim N(0, I)$ and τ^k_θ is the random seed for the mutation operation k. In this algorithm, for compressing θ^k, k seeds must be stored and then, to recover the network parameters, (4.15) has to be applied forwards.

Our compression algorithm instead of storing each seed for each mutation operation, uses the linear congruential generation algorithm (Bolte, 2011) to generate the seeds for each mutation. Therefore, each seed is generated as

$$\tau^k_\theta = (a_\theta \tau^{k-1}_\theta + c_\theta) \; mod \; m, \tag{4.16}$$

where a_θ and c_θ are random numbers generated only once when $k = 1$ and m is a big number typically set as 2^{32}. Then, with the generated τ_θ^k, (4.15) is applied forwards as in (Petroski Such et al., 2017). Hence, instead of storing k random seeds for the k-th mutation, we just have to store three parameters for each network θ^k: a_θ, c_θ and k.

4.3.4. Fitness Function

The fitness function takes the form of a generic quadratic objective function as typically used in MPC. Hence, instead of maximizing the fitness function, as usually done in RL, our objective is to minimize it. Specifically, the fitness function is given by:

$$V(\hat{\mathbf{U}}_{\theta^k}, \hat{\mathbf{Y}}_{\theta^k}) := \sum_{i=1}^{N_{sp}} V_{sp_i}(\hat{\mathbf{Y}}^i_{sp\theta^k}) + \sum_{i=1}^{N_{st}} V_{st_i}(\hat{\mathbf{Y}}^i_{st\theta^k}) + \sum_{i=1}^{M} \sum_{j=0}^{N_u-1} \frac{R_k}{s_k} (\Delta \hat{\mathbf{U}}^i_{\theta^k_j})^2, \tag{4.17}$$

where we are omitting the explicit temporal dependence of $\hat{\mathbf{Y}}_{\theta^k}$ and $\hat{\mathbf{U}}_{\theta^k}$ to avoid overloading the notation. Here, N_{sp} is the number of controlled variables used for set-point tracking, N_{st} is the number of controlled variables to be stabilized ($N_{sp} + N_{st} = N$), $\Delta \hat{\mathbf{U}}^i_{\theta^k_j} = \hat{\mathbf{U}}^i_{\theta^k_{j+1}} - \hat{\mathbf{U}}^i_{\theta^k_j}$, M is the number of manipulated variables and $\hat{\mathbf{Y}}_{sp\theta^k}$ and $\hat{\mathbf{Y}}_{st\theta^k}$ are submatrices formed by extracting the entries corresponding to controlled variables used for set-point tracking and stabilization, respectively. Moreover,

$$V_{sp_i}(\hat{\mathbf{Y}}^i_{sp\theta^k}) = \sum_{j=1}^{N_y-1} \frac{Q_i}{l_i} (\hat{\mathbf{e}}^i_{sp_j})^2 + \sum_{j=1}^{N_y} \frac{\epsilon_{ij}}{l_i} \Lambda_i + \beta_i \left(\hat{\mathbf{Y}}^i_{sp\theta^k_{N_y}} - \mathbf{y}^i_{sp_{ss}} \right)^2. \tag{4.18}$$

On the other hand $V_{st_i}(\hat{\mathbf{Y}}^i_{st\theta^k})$, is equal to

$$V_{st_i}(\hat{\mathbf{Y}}^i_{st\theta^k}) = \sum_{j=1}^{N_y-1} \frac{Q_i}{l_i}(\Delta\hat{\mathbf{Y}}^i_{st\theta^k_j})^2 + \sum_{j=1}^{N_y} \frac{\epsilon_{ij}}{l_i}\Lambda_i \tag{4.19}$$

- N_y and N_u are the prediction and control horizon, respectively.
- $\hat{\mathbf{e}}^i_{sp_j} := w^i_{sp}(j) - \hat{\mathbf{Y}}^i_{sp\theta^k_j}$ is the predicted error for the controlled variable i, with respect to the respective reference sub-sequence w^i_{sp}.
- $\mathbf{y}^i_{sp_{ss}}$ is the steady-state target for controlled variable i.
- ϵ_{ij} is a binary *on-off* variable that takes into account the violation of a constraint for the controlled variable i at predicted step j.
- $Q_i, R_k, \beta_i, \Lambda_i \geq 0$ are positive weights.
- l_i and s_k are normalization coefficients.
- $\Delta\hat{\mathbf{Y}}^i_{st\theta^k_j} = \hat{\mathbf{Y}}^i_{st\theta^k_{j+1}} - \hat{\mathbf{Y}}^i_{st\theta^k_j}$ are incremental variations to be minimized when stabilizing.

Expression (4.17) considers incremental MV values $\Delta\hat{\mathbf{U}}_{\theta^k}$ as usual in most predictive control applications (Rawlings & Mayne, 2009).

Finally, constraints are included involving both controlled and manipulated variables.

- Controlled Variables Constraints: Outputs must remain bounded by process limits:

$$\underline{\mathbf{y}}^i \leq \hat{\mathbf{Y}}^i_{\theta^k_j} \leq \overline{\mathbf{y}}^i \;\; \forall j, \; \forall i = 1, ..., N \tag{4.20}$$

  These constraints will be *softened* through their inclusion in the objective function as expressed in (18).

- Manipulated Variables Constraints: Actuator limits were established. Also, rate constraints where considered for the MVs.

$$\underline{\mathbf{u}}^i \leq \hat{\mathbf{U}}^i_{\theta^k_j} \leq \overline{\mathbf{u}}^i \;\; \forall j, \; \forall i = 1, ..., M \tag{4.21}$$

$$\left| \Delta \hat{\mathbf{U}}^i_{\theta^k_j} \right| \leq \delta^i_u \;\; \forall j, \; \forall i = 1, ..., M \tag{4.22}$$

Typically, the optimization problem is solved at every time-step, giving a NN whose outputs minimize the objective function. This NN is selected as the controller for that time-step and only its first output is applied to the system, following the receding horizon principle. However, since NE gives a family of well trained NNs for controlling the system, their outputs could be used in multiple time-steps. Hence considerably reducing the computational burden of online optimization.

4.3.5. Handling of hard constraints

In case a particular application demands the addition of a hard constraint during the optimization process, there are several methods proposed in the literature for EA (Chehouri et al., 2016; Ponsich et al., 2008) that could be implemented in our controller. However, the simple elimination of infeasible individuals, also called the *death penalty method*, works well in practice and is used here when necessary.

4.3.6. Replacement

In the original Map Elites algorithm, if an individual $\phi^r_{(i,j)}$ is assigned to the position in the grid that $\theta^k_{(i,j)}$ occupies, then their respective fitness $f_{\phi^r_{(i,j)}}$ and $f_{\theta^k_{(i,j)}}$ are compared and the individual with the higher fitness takes the position in the grid, while the other is discarded. However, in this case, each individual represents a NN that we want to generalize as much as possible, to control the system in different operational points. Similar to the situation when Stochastic Gradient Descent (SGD) is used to train NNs, in NE, because of its random nature, the updates of the weights can be erratic producing that a network with good performance at an specific input may present poor results at others, especially when

the fitness function is evaluated at each time-step. To tackle this problem, instead of using the fitness function at a particular point in time, as presented above, we use an exponential moving average (EMA) (Alexander et al., 2016) with parameter α to take into account previous operational states and to force the population to have good performance in a broad operating range. This, in the same spirit of the momentum technique (Sutskever et al., 2013) used in SGD. Therefore, instead of comparing $f_{\phi^r_{(i,j)}}$ and $f_{\theta^k_{(i,j)}}$ when optimizing, we compare their EMA versions $\bar{f}_{\phi^r_{(i,j)}}$ and $\bar{f}_{\theta^k_{(i,j)}}$, where $\bar{f}_{\phi^r_{(i,j)}} = \alpha f_{\phi^r_{(i,j)}} + (1-\alpha)\bar{f}_{\phi^r_{(i,j)}}$.

4.4. Case Studies

4.4.1. Baselines for performance evaluation

4.4.1.1. Deep deterministic policy gradients

DDPG (Lillicrap et al., 2015) is a policy gradients controller consisting in two networks, an actor, that given the state of the system calculates an action, and a critic, which evaluates the actions taken by the actor by estimating the Q-value function. DDPG was designed for tasks with continuous action spaces and has excelled in control tasks (Qiu et al., 2019).

4.4.1.2. Evolutionary algorithm

We consider the algorithm presented in (Petroski Such et al., 2017), which is similar to ours but uses a standard genetic algorithm to optimize the population.

4.4.1.3. Linear MPC

A linear MPC is considered since this type of controller is typically used in real industrial control applications including pH neutralization processes (Hermansson & Syafiie, 2015). Linear MPCs have strong performance when the state of the system is around the equilibrium point.

4.4.1.4. Multivariable PID

As MPC controllers, PID controllers are typically used in industrial applications. In this case, a multivariable PID controller tuned using Particle Swarm Optimization (Mezura-Montes & Coello, 2011) is considered as baseline.

4.4.2. Evaluation metrics

To evaluate performance of the different controllers, two commonly used metrics are used, the integral squared error (ISE) and the total energy (TE), defined as

$$ISE = \sum_{j=0}^{N_y-1} (w_j^i - \hat{\mathbf{Y}}_{\theta_j^k}^i)^2,\ TE = \sum_{i=1}^{M}\sum_{j=0}^{N_u} (\Delta\hat{\mathbf{U}}_{\theta_j^k}^i)^2 \tag{4.23}$$

4.4.3. Neuroevolutive control of a pH neutralization process

For a first set of experiments, the multivariable pH neutralization process presented in (Hall & Seborg, 1989) is used. This system consists in two stirred tank reactors with an acid-base neutralization reaction taking place in each reactor, as shown in Fig. 4.1. The system has four controlled variables: the level and pH of each stirred tank reactor (inferior tanks); four manipulated variables: acid and base flowrates; and two flowrates that act as disturbances. This process is highly nonlinear, with long delays and time varying characteristics due to the inherit nonlinearity associated with pH and the shifts in the titration curve (Hall & Seborg, 1989).

This system has eight state variables: the four water levels and the two reaction invariants ($W_{a_{T_i}}$, $W_{b_{T_i}}$) as defined in (Hall & Seborg, 1989) for each reactor. Then, to obtain the pH value from the state variables, the following nonlinear mapping is used:

$$W_{a_{T_i}} = [H^+] - \frac{K_w}{[H^+]} - W_{b_{T_i}} \frac{K_{a1}/[H^+] + 2K_{a1}K_{a2}/[H^+]^2}{1 + K_{a1}/[H^+] + K_{a1}K_{a2}/[H^+]^2}, \tag{4.24}$$

where K_w, K_{a1} and K_{a2} are the equilibrium constants of the reactions and $[H^+]$ is the hydrogen ion concentration, which is the variable we want to obtain from the equation.

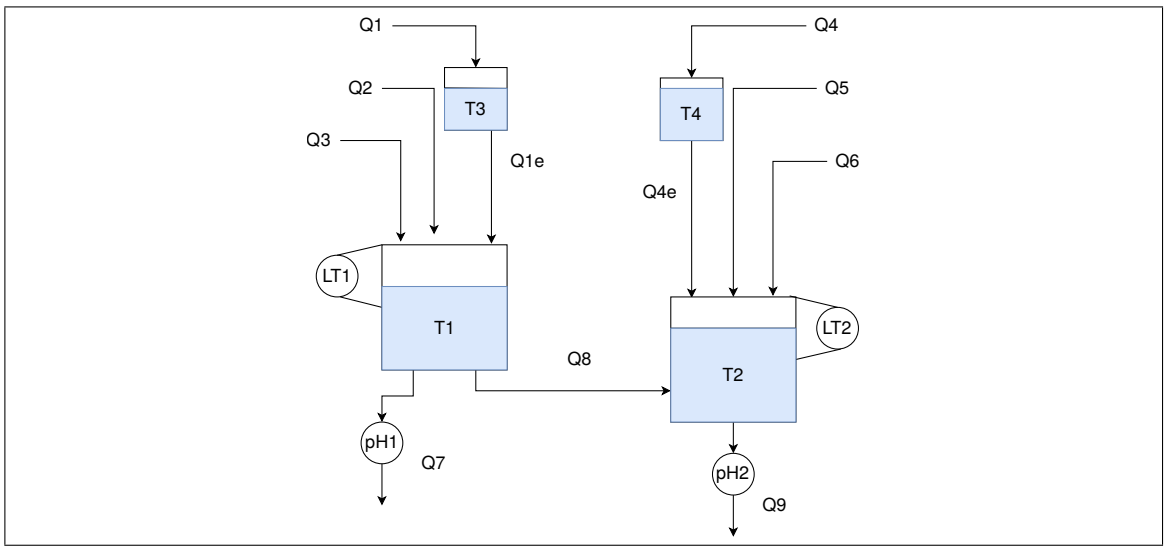


Figure 4.1. Multivariable pH neutralization process.

Table 4.1. Parameters of the simulator. W_{ai}, W_{bi} are the initial value of the reaction invariants. K_i are the equilibrium constants of the reactions and H_{max_s}, H_{max_I} are the maximum height of the superior and inferior tanks respectively.

Parameters of the simulator		
$W_{a1} = 3 \times 10^{-3}\ M$	$W_{a4} = 4 \times 10^{-3}\ M$	$K_{a1} = 4.47 \times 10^{-7}$
$W_{b1} = 0\ M$	$W_{b4} = 2 \times 10^{-6}\ M$	$K_{a2} = 5.62 \times 10^{-11}$
$W_{a2} = -0.02\ M$	$W_{a5} = -0.01\ M$	$K_w = 1 \times 10^{-14}$
$W_{b2} = 0.024\ M$	$W_{b5} = 0.01\ M$	$H_{max_s} = 20\ cm$
$W_{a3} = -0.028\ M$	$W_{a6} = -0.014\ M$	$H_{max_I} = 40\ cm$
$W_{b3} = 0.028\ M$	$W_{b6} = 0.014\ M$	

Finally, to obtain the pH, the following equation must be used:

$$pH = -log_{10}[H^+] \tag{4.25}$$

Simulations were conducted using the parameters presented in Table 4.1, on a 3.2GHz Intel Core i7-8700 machine with 32 GB of RAM and an NVIDIA RTX 2080-TI graphics card.

Table 4.2. Parameters of the encoder-decoder model.

Parameter	Value
N° of GRU units	20
N° of layers in encoder	2
N° of layers in decoder	2
Sequence length	20
Dropout prob	0.3
Learning Rate	1×10^{-4}

4.4.3.1. Model identification

As stated in Section III, for the proposed Map Elites algorithm an accurate model is needed. To generate a model, the simulator is excited at different operational points and an encoder-decoder recurrent NN with attention mechanism and GRU units is trained with these signals. This type of model was selected due to its good performance for multiple step-ahead predictions and its proven efficacy in industrial applications (Núñez et al., 2020). The network was trained to predict the controlled variables taking as inputs a sliding window of past manipulated variables and past controlled variables, leaving disturbances as unmeasured variables. Adam optimizer was used and MSE as loss function. Other parameters of the model are presented in Table 4.2 and 15 step-ahead predictions are presented in Fig. 4.2 with the corresponding RMSE.

Because of the time-varying nature and strong disturbances present in industrial systems, the weights of the model are updated at each time-step, following the Dyna algorithm. However, due to stability issues when training deep architectures, it is better to decorrelate the training signals at each update. Therefore, a buffer of past signals is maintained, similar to what is done in DRL applications with experience replay.

4.4.3.2. Controller design

The controller considers as agents recurrent encoder-decoder NNs with 2 GRU layers of 30 units at both encoder and decoder, yielding 18,484 parameters for each agent, which

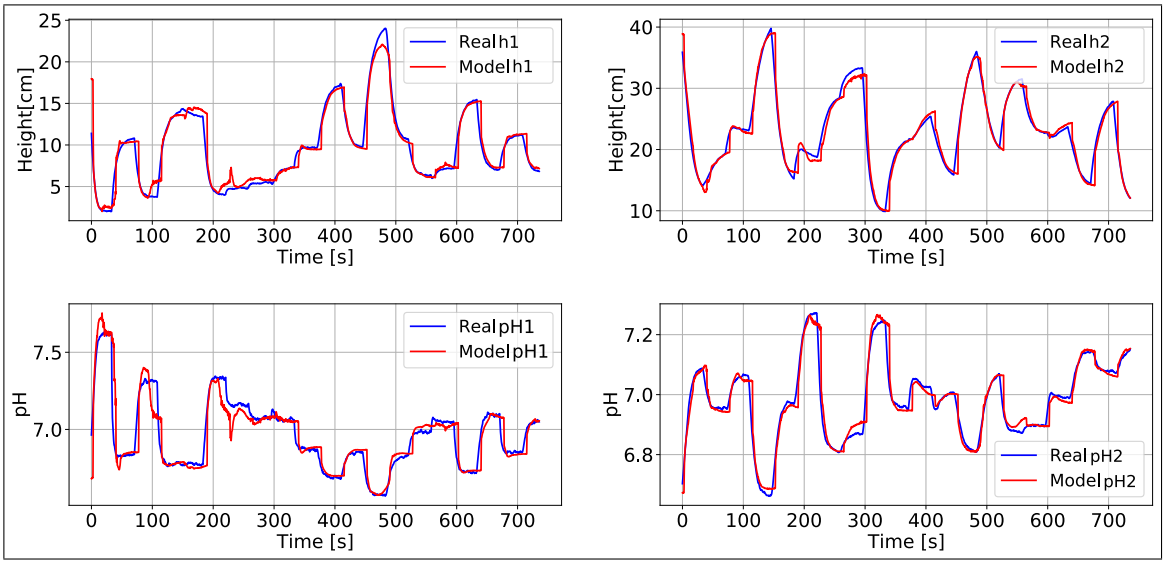


Figure 4.2. 15 steps-ahead predictions of the encoder-decoder with attention mechanism. The RMSE for each signal is $h1_{rmse} = 1.16\ cm$, $h2_{rmse} = 1.5\ cm$, $pH1_{rmse} = 0.088$, $pH2_{rmse} = 0.034$

are compressed using the algorithm proposed in Section III. To evaluate the fitness, parameters were set as: $N_y = 15$, $N_u = 10$. For evolving the population, the following were used, a mutation parameter $\epsilon = 0.2$, $M = 30$, $C = 0$ individuals taken for mutation and cross-over, and EMA parameter $\alpha = \frac{2}{t_{ema}+1}$ where $t_{ema} = 20$ is the length of the moving average window for fitness values. The population evolves through multiple generations until the termination criterion $\Delta = 5$ seconds, equal to the sampling time of the system, is reached. For simplicity, only three features, in this case related to the fitness function, were selected: set-point tracking error (sp), control effort (Δu), and constrains soft violation (Λ). Hence, a three-dimensional grid was used with, arbitrary, dimension $8 \times 8 \times 8$. It should be noted that while optimizing, in each iteration the grid is reset, $n_{new} = 100$ new individuals are created and are evaluated with the old ones using the arriving input signals.

4.4.3.3. Results for Set-point tracking and Stabilization

In this case, the control objective is primarily to track the pH set-point in the second stirred tank and, secondarily, to stabilize the rest of the controlled variables. To make the

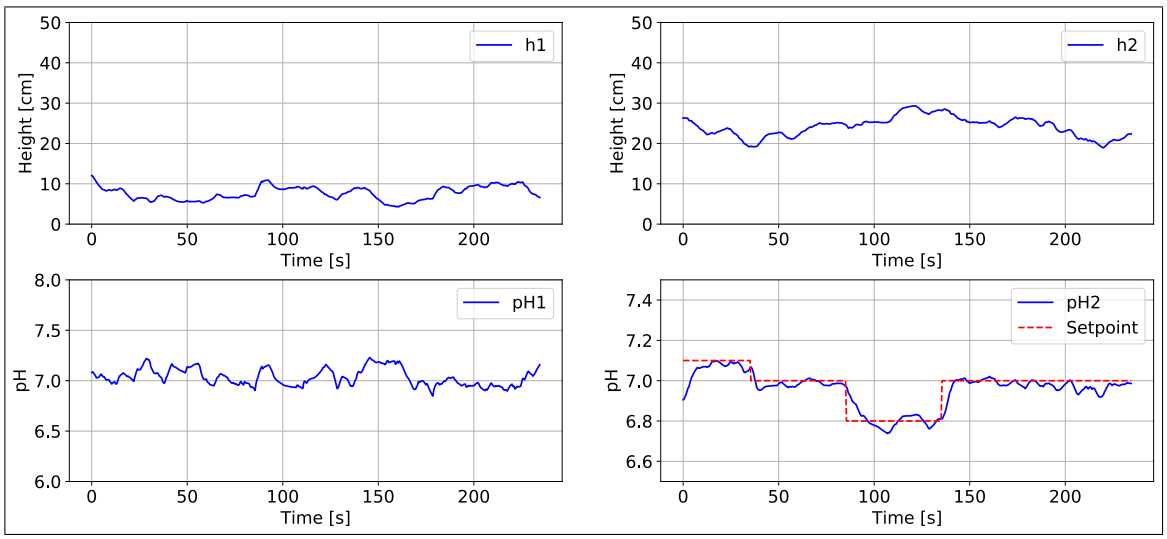


Figure 4.3. Set-point tracking for $pH2$ and Stabilization for the rest of the variables with the Map Elites controller.

simulation more realistic, noise was introduced to measured variables and unmeasured disturbances. However, for this test, the mean value of the disturbance signals remained constant.

Fig. 4.3 shows the performance of the Map Elites controller in set-point tracking and stabilization. It can be seen that the controller is able to drive pH_2 to the desired set-point but with some complications due to the unmeasured disturbances. Table 4.3 shows a comparison with the baselines in terms of TE and ISE. The Map Elites controller outperforms all the baselines.

4.4.3.4. Results for Disturbance Rejection

For this test, unmeasured disturbances were drastically changed to evaluate the ability of the controllers to maintain pH_2 at its set-point.

Fig. 4.4 shows both pH_2 and the disturbance signals when the Map Elites controller is used. It can be seen that the controller successfully rejects the strong disturbances. Table 4.3 indicates that the Map Elites controller is the best performer in terms of ISE, however the PID and MPC controllers have lower TE.

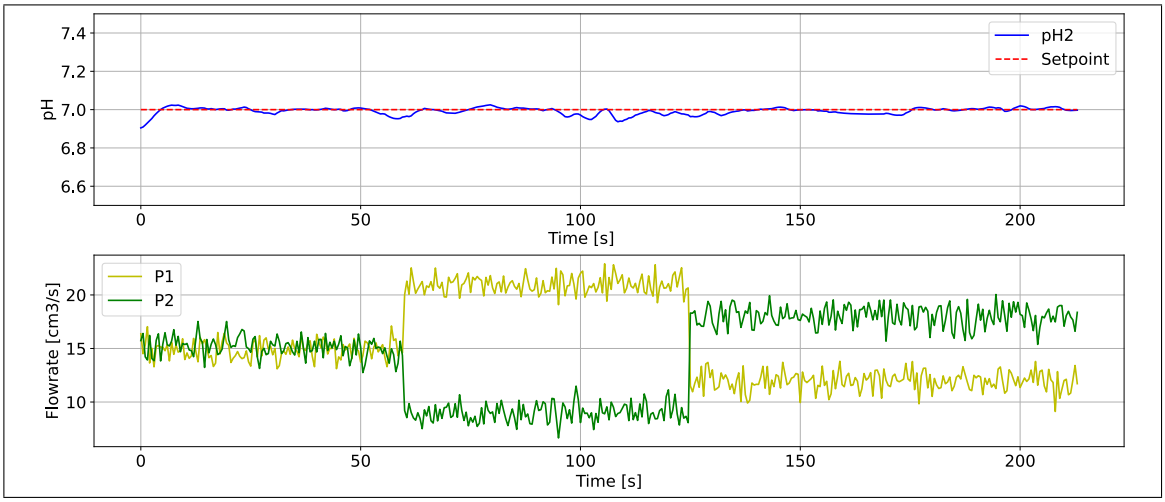


Figure 4.4. Disturbance Rejection for $pH2$ with two abrupt changes in the unmeasured disturbances

Table 4.3. Set-point tracking and Disturbance rejection performance for the different controllers in terms of TE and ISE

		Map Elites	EA controller	DDPG	Linear MPC	PID
S.T.	ISE	**0.7674**	2.073	6.25	4.004	4.335
	TE	30.447	51.881	102.284	16.343	**7.190**
D.R.	ISE	**0.1439**	0.819	8.027	0.456	0.222
	TE	22.285	14.915	7.878	**2.253**	5.459

4.4.3.5. Results changing the optimization frequency

One of the benefits of learning-based controllers is that the minimization of the objective function not only produces manipulated variables for driving the system to the set-point, but also the controller learns the dynamics of the system. This means that at some point the controller should be able to produce appropriate manipulated variables without optimizing at each step, thus, considerably reducing the computational load.

To test the learning ability of the learning-based controllers, after the first 50 steps of learning, we reduce the frequency of the optimization by different amounts. For the Map Elites and the EA controller, when no optimizing, the controller with best fitness among all was selected in each step. For DDPG, the actor produced manipulated variables at each step without optimizing its weights, and for the linear MPC, prediction and control

Table 4.4. Results for set-point tracking in terms of ISE when changing the frequency of the optimization after training each controller in the first 50 steps.

Optimizing each	20 steps	30 steps	50 steps
Map Elites	**1.354**	**1.358**	**2.0475**
DDPG	3.321	3.475	4.342
EA controller	2.196	4.8119	5.645
Linear MPC	3.497	2.106	4.246

horizons are set equal to the number of steps without optimizing and the entire control sequence is used.

Table 4.4 shows the results of this test. It can be seen that the Map Elites controller shows the best performance, followed by the EA controller, both with learning capabilities. On the other side, the DDPG and the linear MPC show the worst results. Even though the DDPG controller has learning capabilities, it learns too slow, in comparison to the other learning-based controllers. This reinforces what was pointed out in (Petroski Such et al., 2017), that EAs find good solutions much faster than RL techniques.

4.4.3.6. Using the features of the grid

To show how controllers from different cells of the grid produce different responses, without changing the fitness function, tests were conducted restricting the cell from which the controller is taken.

Table 4.5 shows the results. Here, the sp-axis was fixed and Δu-axis was varied all over the grid in different simulations. It can be seen that as Δu increases, the TE increases dramatically and the ISE remains relatively constant, showing that this has a similar effect as varying the relative weights of the error and energy terms in the fitness function. A similar approach can be taken with all the features used to construct the grid, which are not necessarily related with the fitness function.

A visualization of the feature space is shown in Fig. 4.5 from where it can be seen the different zones of the grid with high performing individuals that present different features.

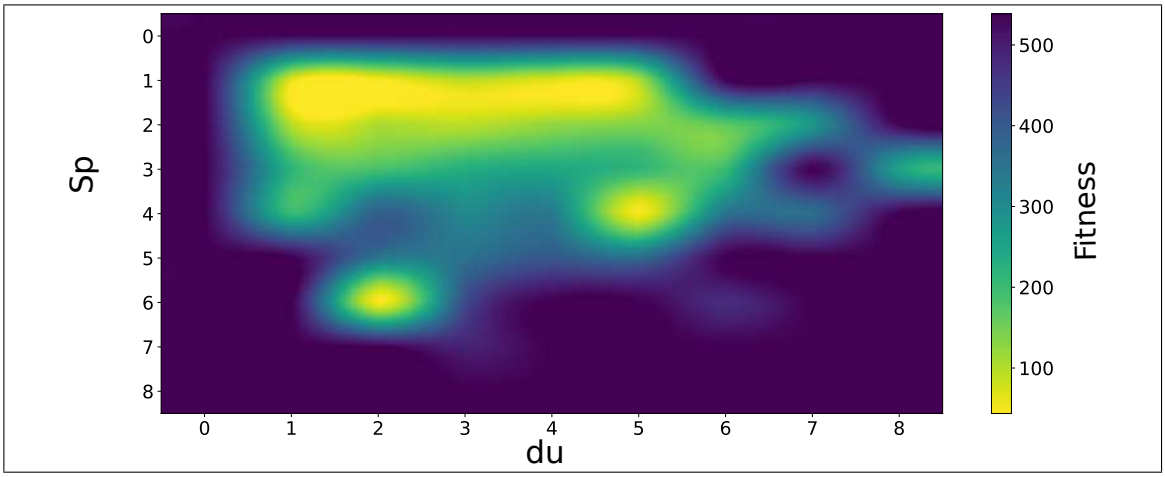


Figure 4.5. Feature space grid. Most of the high performing individuals (low fitness) are near $sp = 1$, $du = [1 : 5]$.

Table 4.5. Results in terms of ISE and TE when taking controllers from specific cells on the grid.

Δu	TE	ISE
$\Delta u_1 \vert sp_1$	20.561	1.0878
$\Delta u_3 \vert sp_1$	298.032	1.0462
$\Delta u_5 \vert sp_1$	316.938	1.6947
$\Delta u_7 \vert sp_1$	329.756	2.0747

4.4.4. Neuroevolutive control of an industrial paste thickener

For a second set of experiments, a pseudo-real setup involving an industrial paste thickener is used. Paste thickeners are the primary method for producing high density tailings in mineral processing. The thickener receives the tailings slurry along with a sedimentation-promoting polymer known as flocculant, which increases the sedimentation rate, and produces a high density material as underflow. Thickening is a slow process, with response times in the order of several hours, highly nonlinear, multi-input multi-output (MIMO), and subject to multiple disturbances(Núñez et al., 2020). The control objective is to drive the output solids concentration to a desired set-point, typically between 68% and 70%, while stabilizing the internal states and minimizing flocculant consumption.

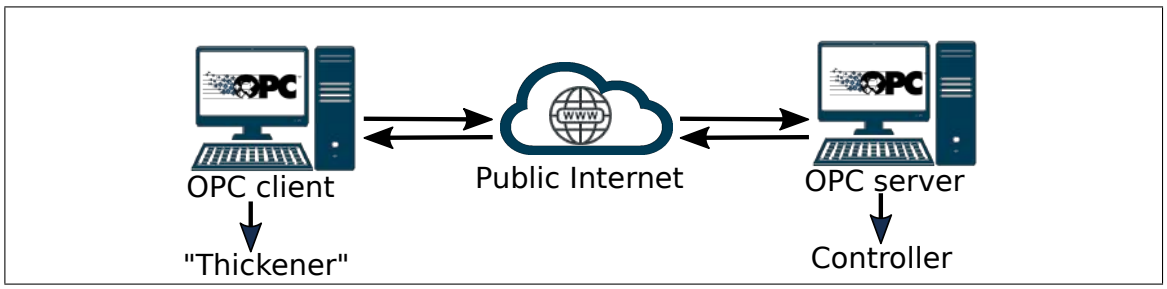


Figure 4.6. Experimental setup used for controlling the thickener over the internet using OPC-UA.

4.4.4.1. Experimental Setup

Inspired by the work in (Núñez et al., 2020), a pseudo-real environment was setup. In (Núñez et al., 2020) a model was obtained using real operational data, which closely represents the behavior of the real system. To evaluate the performance of the proposed controller, in this work we run the model from (Núñez et al., 2020) in a remote machine that communicates over the public Internet via OPC-UA to a local machine where the neuroevolutive controller is hosted. The local machine is the same used in the previous experiments, while for the remote machine we use a 2.7GHz Intel Core i7-7500U with 16 GB of RAM and an NVIDIA GeForce 940MX graphics card. The experimental setup is shown in Fig. 4.6. It should be noted that the unreliable nature of communications over the public Internet makes the experiments more meaningful since it mimics the situation in a real industrial facility.

4.4.4.2. Model identification

The model needed for NE, is generated using encoder-decoder recurrent neural networks, following the scheme presented in (Núñez et al., 2020). It should be noted that the model used for NE is simpler than the model hosted in the remote machine, which represents the thickener.

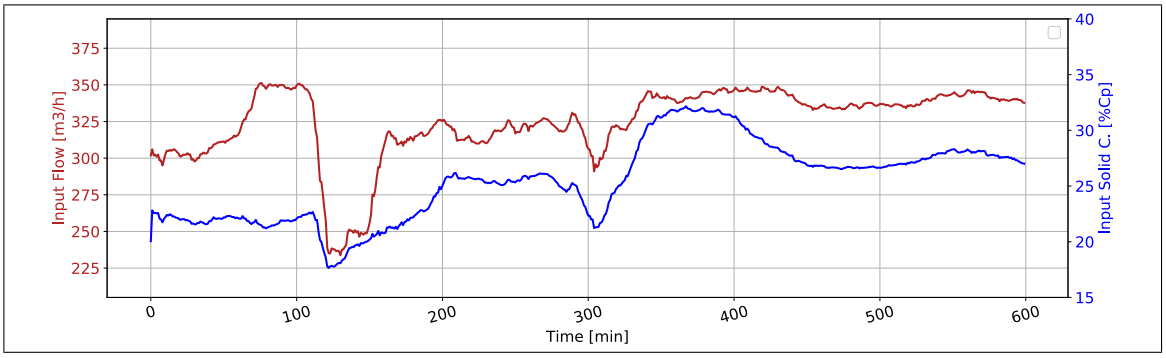


Figure 4.7. Real disturbances applied to the thickener simulator.

4.4.4.3. Controller design

The controller considers recurrent encoder-decoder NNs as agents and uses exactly the same parameters as in Section IV.C, except for the termination criterion Δ, which is set equal to the sampling time of the thickener: 1 minute.

4.4.4.4. Results for Set-point tracking and disturbance rejection

The objective is to regulate the output solids concentration to a desired set-point, while rejecting the strong disturbances (real data) at the input shown in Fig. 4.7. Results for the Map Elites controller are presented in Fig. 4.8. It can be seen that the controller is able to regulate the output to the desired set-point.

Table 4.6 presents the results for all the evaluated controllers. It can be seen that the Map Elites controller achieves the lowest ISE yet the highest TE_{u2}, which suggests that the controller uses a strategy based mainly on manipulating the output flow to control the output solids concentration and the flocculant to reject disturbances. As observed for human operators (Núñez et al., 2020).

4.4.5. Performance of compression algorithm

The compression algorithm is one of the highlights of the proposed controller. To demonstrate the advantages an analysis regarding memory savings and decompression

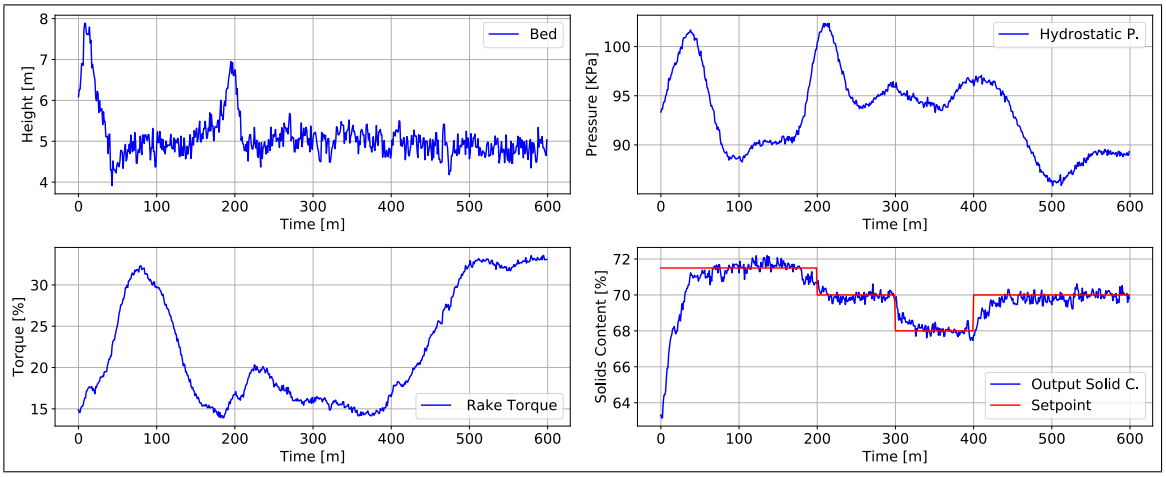


Figure 4.8. Set-point tracking and disturbance rejection for output solids concentration with the Map Elites controller.

Table 4.6. Set-point tracking and Disturbance rejection performance for the different controllers in terms of TE and ISE. TE_{u1} denotes flocculant flow and TE_{u2} output flow.

		Map Elites	EA controller	DDPG	Linear MPC	PID
	ISE	6033	7589	30534	20935	26001
S.T.	TE_{u1}	88.4	82.7	93.2	102.4	100.8
	TE_{u2}	278681	100422	75732	56251	96403

time is conducted. All the experiments were executed in the 3.2GHz Intel Core i7-8700 machine previously introduced.

4.4.5.1. Memory savings

The biggest advantage of the proposed compression algorithm is memory savings since NNs of arbitrary size can be compressed using only three parameters. Therefore, the required amount of memory only increases linearly with the population size. Fig. 4.9(a) shows the memory required to maintain different populations of encoder-decoder NNs, for different hidden sizes, using and not using the compression algorithm. For the sake of clarity, only the curve when compressing 300 NNs is shown in the figure.

For an experiment similar to the ones shown in the case studies (300 genes on average and a hidden size of 30) the memory usage when using the compression algorithm is only 2.536×10^{-3} MB, as opposed to 66.413 MB when storing all the networks parameters in a traditional manner. This yields a reduction factor of 26188, which increases exponentially with the hidden size of the networks.

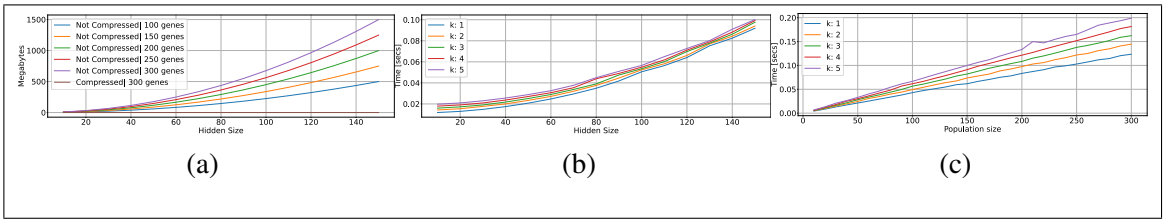
4.4.5.2. Time required for decompression

Compression algorithms present a trade-off between memory savings and decompression time. The user has to decide, based on the system requirements and the available hardware, which aspect should be prioritized. Since the proposed compression algorithm is recursive, decompression time not only depends on the hidden size of the networks and the population size, but also on the number of mutations k the original gene has suffered. Fig. 4.9 (b) and (c) show how decompression time varies as a function of the hidden and population size for different values of k. It can be seen that the decompression time for 30 networks of hidden size 30, is about 0.025 s, which represents about one tenth of the time required to run all the operations needed (see Algorithm 2) in each generation when optimizing.

This analysis also gives some insights on some simple modifications that can be done to the algorithm on the run, depending on the available computational resources at the moment in order to always meet the time constraints of a particular application. For example, number of new individuals n_{new} at each optimization step could be varied, a limit on the number of mutations k for each individual could be set, hidden size of new individuals could be limited, or the grid size and grid granularity could be modified on demand.

4.5. Discussion

In this chapter a learning-based data-driven controller with flexible computational workload and the capacity to easily enforce both soft and hard constraints is presented.



(a) (b) (c)

Figure 4.9. (a) Required memory for maintaining a population of NNs as a function of the hidden size of the individuals. (b) Decompression time as a function of the hidden size of the individuals. A constant population size of 30 is considered. (c) Decompression time as a function of the population size. A constant hidden size of 30 neurons is considered.

The controller is based on the map-elites optimization algorithm and with a novel compression algorithm is able to optimize several neural networks simultaneously in a desktop computer. Experiments showed the novel capacities of the map-elites controller and its superiority to other classical and learning-based approaches. Future work includes implementing the controller in a real process and analyzing the benefits of incorporating application-specific features in the map.

5. CONCLUSIONS AND FUTURE WORK

5.1. Concluding Remarks

In this thesis, three deep learning methods for iCPSs designed to deal with the inherent difficulties that real systems impose on data-driven learning techniques were designed, implemented and evaluated, namely, a contrastive blind denoising autoencoder for data-cleaning, an adaptative model for glucose prediction with a probabilistic extension, and a neuroevolutive controller with learning capabilities and flexible computational workload.

As for the data-cleaning method, experimental results show that the CBDAE outperforms classical data-driven and model-based denoising techniques typically used in the industry, as well as other state-of-the art neural networks. The use of the NCE regularization enables the CBDAE to capture high-level information, i.e., dynamical information, in the latent space of the network leaving out irrelevant information, as noise. Additionally, it was shown that NCE regularization induces a smooth and meaningful structure in the latent space, which can be eventually used for other downstream tasks as fault detection or control.

In the case of the adaptative model, a Meta-Learning based approach is proposed for producing personalized deep-learning models that can potentially be used in an MPC application for the artificial pancreas. It was shown that by using this approach, models require less data, fewer training iterations and have a lower risk of over-fitting when adapting to a particular patient if compared to other personalization approaches, like transfer-learning. Results show that the proposed model is superior to other strong baselines in standard accuracy metrics and task-specific metrics, difference that is accentuated for longer horizons and higher degrees of distributional shifts. Also, a probabilistic extension based on recurrent Kalman networks, to estimate the uncertainty produced by meal specification errors by the patient, is proposed. This probabilistic model demonstrates better

performance than other approaches in the task, and it is shown how it can correctly estimate the propagated uncertainty on the glucose signal. Additionally, the generalization capability of the two proposed models is tested with a novel task-specific out-of-distribution evaluation procedure.

Finally, a new learning-based neural controller that combines elements of neuroevolution, reinforcement learning and model predictive control is introduced along with an efficient compression algorithm for neural networks evolved with evolutionary algorithms that allows to reduce the computational burden of maintaining big populations of deep neural networks. Multiple tests against strong baselines in two complex multivariable nonlinear systems show that, in addition to good performance, the proposed controller can reduce the computational burden of optimizing at each time-step, as classical controllers do, thanks to its learning capabilities. The use of a population of trained neural networks as candidate controllers provides an intuitive method to tune the control loop without modifying the weights of the fitness function and a simple way to adapt the computational workload of the controller.

Fulfillment of the objectives of this thesis through the design, implementation and evaluation of the aforementioned methods in different CPSs applications, evidences how properly designed DL methods are not only superior to classical techniques across different CPSs application domains, but also endow CPSs with new capabilities as personaliozation, feature extraction, and learning. Thus, showing how CPSs can be highly benefited by the incorporation of DL methods at different levels of its architecture, and how these learning-based techniques can help to unleash the full potential of these complex systems.

It is expected that the results of this research will encourage the development of more intelligent data-driven methods for iCPSs.

5.2. Directions for Future Research

Even though, the results in this thesis are encouraging and show that DL-based methods are able to improve the performance of different iCPSs across different application domains, there are still several concrete actions drawn from this work that can be taken to continue unlocking the full potential of this type of systems, namely,

- Exploring the latent space of the CBDAE in order to use it for different applications, such as reconstruction, fault detection or even control, where a latent vector could be used as an informative feature of the state of the system.
- Implementing a probabilistic extension of the CBDAE, in order to account for the inherent uncertainties of the underlying physical process and the measurement process.
- Enhancing the proposed models with expert knowledge in the form of neural differential equations or penalty terms in the respective losses. This could help improving generalization, explainability, and reliance of the models.
- Implementing a probabilistic extension of the neuroevolutive controller in order to effectively incorporate a probabilistic model in its design so that uncertainty-aware decisions can be made.
- Performing a stability analysis of the neuroevolutive controller using Lyapunov functions or, similarly to other approaches, incorporating Lyapunov conditions in the construction of the controller to synthesize a provably stable controller.
- Integrating the three proposed methods in a unique iCPS and test it on a real process.

REFERENCES

Abooshahab, M. A., Alyaseen, M. M., Bitmead, R. R., & Hovd, M. (2022). Simultaneous input state estimation, singular filtering and stability. *Automatica*, *137*, 110017.

Ahn, C. W., & Ramakrishna, R. S. (2003, Aug). Elitism-based compact genetic algorithms. *IEEE Trans. Evol. Comput.*, *7*(4), 367-385.

Alexander, B., Ivan, T., & Denis, B. (2016, May). Analysis of noisy signal restoration quality with exponential moving average filter. In *2016 international siberian conference on control and communications (sibcon)* (p. 1-4).

Ansari, A., & Bernstein, D. S. (2019). Input estimation for nonminimum-phase systems with application to acceleration estimation for a maneuvering vehicle. *IEEE Trans. Control Syst. Technol.*, *27*(4), 1596-1607.

Antonio, L., & Coello, C. (2018, Dec). Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art. *IEEE Trans. Evol. Comput.*, *22*(6), 851-865.

Armandpour, M., Kidd, B., Du, Y., & Huang, J. Z. (2021, June). Deep Personalized Glucose Level Forecasting Using Attention-based Recurrent Neural Networks. *arXiv e-prints*, arXiv:2106.00884.

Arnold, S. M. R., Mahajan, P., Datta, D., Bunner, I., & Zarkias, K. S. (2020). learn2learn: A library for meta-learning research. *CoRR*, *abs/2008.12284*.

Arnold, S. M. R., & Sha, F. (2021, April). Embedding Adaptation is Still Needed for Few-Shot Learning. *arXiv e-prints*, arXiv:2104.07255.

Arsene, C. T., Hankins, R., & Yin, H. (2019). Deep learning models for denoising ecg signals. In *2019 27th european signal processing conference (eusipco)* (p. 1-5).

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., ... Blundell, C. (2020). Never give up: Learning directed exploration strategies. In *International conference on learning representations.*

Batson, J., & Royer, L. (2019). Noise2self: Blind denoising by self-supervision. *CoRR*, *abs/1901.11365*.

Becker, P., Pandya, H., Gebhardt, G. H. W., Zhao, C., Taylor, C. J., & Neumann, G. (2019). Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. *CoRR*, *abs/1905.07357*.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015, Jun). Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *arXiv e-prints*, arXiv:1506.03099.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *CoRR*, *abs/1206.5533*.

Berend, D., Xie, X., Ma, L., Zhou, L., Liu, Y., Xu, C., & Zhao, J. (2020). Cats are not fish: Deep learning testing calls for out-of-distribution awareness. In *2020 35th ieee/acm international conference on automated software engineering (ase)* (p. 1041-1052).

Bolte, J. (2011). Linear congruential generator. *Wolfram Demonstrations Project.*. Retrieved from `https://demonstrations.wolfram.com/LinearCongruentialGenerators/`

Brazeau, A. S., Mircescu, H., Desjardins, K., Leroux, C., Strychar, I., Ekoé, J. M., & Rabasa-Lhoret, R. (2012, November). Carbohydrate counting accuracy and blood glucose variability in adults with type 1 diabetes. *Diabetes Res Clin Pract*, *99*(1), 19–23.

Brown, S. A., Kovatchev, B. P., Raghinaru, D., Lum, J. W., Buckingham, B. A., Kudva, Y. C., ... Beck, R. W. (2019). Six-month randomized, multicenter trial of closed-loop control in type 1 diabetes. *New England Journal of Medicine*, *381*(18), 1707-1717.

Carlson, A. L., Sherr, J. L., Shulman, D. I., Garg, S. K., Pop-Busui, R., Bode, B. W., ... Vigersky, R. A. (2021, November). Safety and glycemic outcomes during the MiniMed™ advanced hybrid Closed-Loop system pivotal trial in adolescents and adults with type 1 diabetes. *Diabetes Technology Therapeutics*, *24*(3), 178–189.

Chaitanya, C. R. A., Kaplanyan, A. S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai,

D., & Aila, T. (2017, July). Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, *36*(4), 98:1–98:12.

Chehouri, A., Younes, R., Perron, J., & Ilinca, A. (2016). A constraint-handling technique for genetic algorithms using a violation factor. *CoRR*, *abs/1610.00976*.

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, February). A Simple Framework for Contrastive Learning of Visual Representations. *arXiv e-prints*, arXiv:2002.05709.

Cheng, Y., Zhang, Y., Ji, P., Xu, W., Zhou, Z., & Tao, F. (2018, Jul 01). Cyber-physical integration for moving digital factories forward towards smart manufacturing: a survey. *The International Journal of Advanced Manufacturing Technology*, *97*(1), 1209-1221.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014, October). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1724–1734).

Choi, K., Jang, D., Kang, S., Lee, J., Chung, T., & Kim, H. (2016, March). Hybrid algorithm combing genetic algorithm with evolution strategy for antenna design. *IEEE Trans. Magn.*, *52*(3), 1-4.

Clarke, W. L., Cox, D., Gonder-Frederick, L. A., Carter, W., & Pohl, S. L. (1987, 09). Evaluating Clinical Accuracy of Systems for Self-Monitoring of Blood Glucose. *Diabetes Care*, *10*(5), 622-628.

Clavera, I., Nagabandi, A., Fearing, R. S., Abbeel, P., Levine, S., & Finn, C. (2018). Learning to adapt: Meta-learning for model-based control. *CoRR*, *abs/1803.11347*.

Colombo, A. W., Karnouskos, S., Kaynak, O., Shi, Y., & Yin, S. (2017). Industrial cyber-physical systems: A backbone of the fourth industrial revolution. *IEEE Industrial Electronics Magazine*, *11*(1), 6-16.

Conti, E., Madhavan, V., Petroski Such, F., Lehman, J., Stanley, K. O., & Clune,

J. (2017, Dec). Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *arXiv e-prints*, arXiv:1712.06560.

Creswell, A., & Bharath, A. A. (2019, April). Denoising adversarial autoencoders. *IEEE Trans. Neural Netw. Learn. Syst.*, *30*(4), 968-984.

Daniels, J., Herrero, P., & Georgiou, P. (2022a, January). A deep learning framework for automatic meal detection and estimation in artificial pancreas systems. *Sensors (Basel)*, *22*(2).

Daniels, J., Herrero, P., & Georgiou, P. (2022b). A multitask learning approach to personalized blood glucose prediction. *IEEE Journal of Biomedical and Health Informatics*, *26*(1), 436-445.

Das Sharma, K., Chatterjee, A., Siarry, P., & Rakshit, A. (2021). A novel disturbance rejection factor based stable direct adaptive fuzzy control strategy for a class of nonlinear systems. *Expert Systems*, *38*(3), e12651.

De Bois, M., Yacoubi, M. A. E., & Ammi, M. (2022, Jan 01). Glyfe: review and benchmark of personalized glucose predictive models in type 1 diabetes. *Medical & Biological Engineering & Computing*, *60*(1), 1-17.

Donoho, D. L. (1995, May). De-noising by soft-thresholding. *IEEE Trans. on Inf. Theory*, *41*(3), 613-627.

Facchinetti, A., Sparacino, G., Trifoglio, E., & Cobelli, C. (2011, February). A new index to optimally design and compare continuous glucose monitoring glucose prediction algorithms. *Diabetes Technol Ther*, *13*(2), 111–119.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, *abs/1703.03400*.

Frusque, G., & Fink, O. (2022, June). Robust Time Series Denoising with Learnable Wavelet Packet Transform. *arXiv e-prints*, arXiv:2206.06126.

Gangwani, T., & Peng, J. (2017, Nov). Policy Optimization by Genetic Distillation. *arXiv e-prints*, arXiv:1711.01012.

Georga, E. I., Protopappas, V. C., Polyzos, D., & Fotiadis, D. I. (2012). A predictive

model of subcutaneous glucose concentration in type 1 diabetes based on random forests. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, *2012*, 2889–2892.

Gillijns, S., & De Moor, B. (2007). Unbiased minimum-variance input and state estimation for linear discrete-time systems. *Automatica*, *43*(1), 111-116.

Gneiting, T., Balabdaoui, F., & Raftery, A. E. (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *69*(2), 243-268.

Gomez, F. J. (2003). *Robust non-linear control through neuroevolution* (Unpublished doctoral dissertation). The University of Texas at Austin.

Gondhalekar, R., Dassau, E., & Doyle, F. J. (2018, May). Velocity-weighting & velocity-penalty MPC of an artificial pancreas: Improved safety & performance. *Automatica*, *91*, 105-117.

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th international conference on machine learning - volume 70* (p. 1321–1330). JMLR.org.

Guo, G., & Zhang, N. (2019). A survey on deep learning based face recognition. *Computer Vision and Image Understanding*, *189*, 102805.

Gutmann, M., & Hyvärinen, A. (2010, 13–15 May). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y. W. Teh & M. Titterington (Eds.), (Vol. 9, pp. 297–304). Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings.

Haidar, A. (2016). The artificial pancreas: How closed-loop control is revolutionizing diabetes. *IEEE Control Systems Magazine*, *36*(5), 28-47.

Hall, R. C., & Seborg, D. E. (1989, June). Modelling and self-tuning control of a multivariable ph neutralization process part I: Modelling and multiloop control. In *1989 american control conference* (p. 1822-1827).

Hermansson, A., & Syafiie, S. (2015). Model predictive control of ph neutralization processes: A review. *Control Engineering Practice*, *45*, 98-109. Retrieved

from https://www.sciencedirect.com/science/article/pii/S0967066115300162 doi: https://doi.org/10.1016/j.conengprac.2015.09.005

Herrero, P., Bondia, J., Adewuyi, O., Pesl, P., El-Sharkawy, M., Reddy, M., ... Georgiou, P. (2017, Jul). Enhancing automatic closed-loop glucose control in type 1 diabetes with an adaptive meal bolus calculator - in silico evaluation under intra-day variability. *Computer methods and programs in biomedicine*, *146*, 125-131.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hospedales, T. M., Antoniou, A., Micaelli, P., & Storkey, A. J. (2020). Meta-learning in neural networks: A survey. *CoRR*, *abs/2004.05439*.

Hyvarinen, A., & Morioka, H. (2016, May). Unsupervised Feature Extraction by Time-Contrastive Learning and Nonlinear ICA. *arXiv e-prints*, arXiv:1605.06336.

Jiang, C., Chen, Y., Chen, S., Bo, Y., Li, W., Tian, W., & Guo, J. (2019). A mixed deep recurrent neural network for mems gyroscope noise suppressing. *Electronics*, *8*(2).

Jiang, Y., Fan, J., Chai, T., Li, J., & Lewis, F. L. (2018). Data-driven flotation industrial process operational optimal control based on reinforcement learning. *IEEE Transactions on Industrial Informatics*, *14*(5), 1974-1989.

Johansson, K. H. (2000, May). The quadruple-tank process: a multivariable laboratory process with an adjustable zero. *IEEE Trans. Control Syst. Technol.*, *8*(3), 456-465.

Jose, C., Cisse, M., & Fleuret, F. (2017, May). Kronecker Recurrent Units. *arXiv e-prints*, arXiv:1705.10142.

Kahanovitz, L., Sluss, P. M., & Russell, S. J. (2017). Type 1 diabetes–a clinical perspective. *Point of care*, *16*(1), 37.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., ... Levine, S. (2019, Mar). Model-Based Reinforcement Learning for Atari. *arXiv e-prints*, arXiv:1903.00374.

Kamthe, S., & Deisenroth, M. P. (2017, June). Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control. *arXiv e-prints*, arXiv:1706.06491.

Khadka, S., & Tumer, K. (2018, May). Evolution-Guided Policy Gradient in Reinforcement Learning. *arXiv e-prints*, arXiv:1805.07917.

King, A. (2021). *What are cyber-physical systems?* Retrieved from `https://www.rmit.edu.au/news/c4de/what-are-cyber-physical-systems`

Kravets, A. G. (2022). *Cyber-physical systems: Intelligent models and algorithms* (1st ed.). Springer Nature.

Krull, A., Buchholz, T.-O., & Jug, F. (2018, November). Noise2Void - Learning Denoising from Single Noisy Images. *arXiv e-prints*, arXiv:1811.10980.

Laakom, F., Raitoharju, J., Iosifidis, A., & Gabbouj, M. (2022, February). Reducing Redundancy in the Bottleneck Representation of the Autoencoders. *arXiv e-prints*, arXiv:2202.04629.

Laguna Sanz, A. J., Doyle, F. J., & Dassau, E. (2017, May). An enhanced model predictive control for the artificial pancreas using a confidence index based on residual analysis of past predictions. *Journal of diabetes science and technology*, *11*(3), 537-544.

Langarica, S., & Núñez, F. (2021). Neuroevolutive control of industrial processes through mapping elites. *IEEE Trans. Ind. Informat.*, *17*(5), 3703-3713.

Langarica, S., & Núñez, F. (2023). Contrastive blind denoising autoencoder for real time denoising of industrial iot sensor data. *Engineering Applications of Artificial Intelligence*, *120*, 105838.

Langarica, S., Pizarro, G., Poblete, P. M., Radrigán, F., Pereda, J., Rodriguez, J., & Núñez, F. (2020). Denoising and voltage estimation in modular multilevel converters using deep neural-networks. *IEEE Access*, *8*, 207973-207981.

Langarica, S., Rodriguez, M., Núñez, F., & Doyle III, F. J. (2023). *A meta-learning approach to personalized blood glucose prediction in type 1 diabetes.* (Accepted in the Control Engineering Practice Journal)

Langarica, S., Rüffelmacher, C., & Núñez, F. (2020). An industrial internet application for real-time fault diagnosis in industrial motors. *IEEE Trans. Autom. Sci. Eng.*, *17*(1), 284-295.

Lecoq, J., Oliver, M., Siegle, J. H., Orlova, N., Ledochowitsch, P., & Koch, C. (2021,

Nov 01). Removing independent noise in systems neuroscience data using deepinterpolation. *Nature Methods*, *18*(11), 1401-1408.

Lee, E. A. (2015a). The past, present and future of cyber-physical systems: A focus on models. *Sensors*, *15*(3), 4837–4869.

Lee, E. A. (2015b). The past, present and future of cyber-physical systems: A focus on models. *Sensors*, *15*(3), 4837–4869.

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, *19*(2), 189-223.

Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., & Aila, T. (2018). Noise2noise: Learning image restoration without clean data. *CoRR*, *abs/1803.04189*.

Leitão, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., & Colombo, A. W. (2016). Smart agents in industrial cyber–physical systems. *Proceedings of the IEEE*, *104*(5), 1086-1101.

Li, K., Daniels, J., Liu, C., Herrero, P., & Georgiou, P. (2020). Convolutional recurrent neural networks for glucose prediction. *IEEE Journal of Biomedical and Health Informatics*, *24*(2), 603-613.

Lillicrap, P., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015, Sep). Continuous control with deep reinforcement learning. *arXiv e-prints*, arXiv:1509.02971.

Liu, X., Zhang, F., Hou, Z., Wang, Z., Mian, L., Zhang, J., & Tang, J. (2020, June). Self-supervised Learning: Generative or Contrastive. *arXiv e-prints*, arXiv:2006.08218.

Liu, X., Zhou, Q., Zhao, J., Shen, H., & Xiong, X. (2019). Fault diagnosis of rotating machinery under noisy environment conditions based on a 1-d convolutional autoencoder and 1-d convolutional neural network. *Sensors*, *19*(4).

Liu, Y., Chen, Y., Li, M., & Wan, Z. (2020). Mpc for the cyber-physical system with deception attacks. In *2020 chinese control and decision conference (ccdc)* (p. 3847-3852).

Lucas, J., Tucker, G., Grosse, R., & Norouzi, M. (2019). *Understanding posterior collapse*

in generative latent variable models. Retrieved from https://openreview.net/forum?id=r1xaVLUYuE

Lundervold, A. S., & Lundervold, A. (2019). An overview of deep learning in medical imaging focusing on mri. *Zeitschrift für Medizinische Physik*, *29*(2), 102-127. (Special Issue: Deep Learning in Medical Physics)

Majumdar, A. (2019). Blind denoising autoencoder. *IEEE Trans. Neural Netw. Learn. Syst.*, *30*(1), 312-317.

Man, C. D., Micheletto, F., Lv, D., Breton, M., Kovatchev, B., & Cobelli, C. (2014, Jan). The UVA/PADOVA type 1 diabetes simulator: New features. *Journal of diabetes science and technology*, *8*(1), 26-34.

Meade, L. T., & Rushton, W. E. (2016, July). Accuracy of carbohydrate counting in adults. *Clin Diabetes*, *34*(3), 142–147.

Mezura-Montes, E., & Coello, C. A. (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, *1*(4), 173 - 194.

Mnih, A., & Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 2265–2273). Curran Associates, Inc.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018). Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys Tutorials*, *20*(4), 2923-2960.

Mohebbi, A., Johansen, A. R., Hansen, N., Christensen, P. E., Tarp, J. M., Jensen, M. L., ... Mørup, M. (2020, February). Short Term Blood Glucose Prediction based on Continuous Glucose Monitoring Data. *arXiv e-prints*, arXiv:2002.02805.

Mouret, J.-B., & Clune, J. (2015, Apr). Illuminating search spaces by mapping elites.

arXiv e-prints, arXiv:1504.04909.

Munir, M. T., Udugama, I. A., Boiarkina, I. A., Yu, W., & Young, B. R. (2017). Beyond the theory — how can academia contribute to the advanced process control of industrial processes? In *2017 6th international symposium on advanced control of industrial processes (adconip)* (p. 541-546). doi: 10.1109/ADCONIP.2017.7983838

Nezamoddini-Kachouie, N., & Fieguth, P. (2005, May). A gabor based technique for image denoising. In *Canadian conference on electrical and computer engineering, 2005.* (p. 980-983).

NSF. (2022). *Cyber-physical systems (cps).* National Science Foundation. Retrieved from `https://beta.nsf.gov/funding/opportunities/cyber-physical-systems-cps`

Núñez, F., Langarica, S., Díaz, P., Torres, M., & Salas, J. C. (2020). Neural network-based model predictive control of a paste thickener over an industrial internet platform. *IEEE Trans. Ind. Informat.*, *16*(4), 2859-2867.

Oks, S. J., Fritzsche, A., & Möslein, K. M. (2017). An application map for industrial cyber-physical systems. In S. Jeschke, C. Brecher, H. Song, & D. B. Rawat (Eds.), *Industrial internet of things: Cybermanufacturing systems* (pp. 21–46). Cham: Springer International Publishing.

O'Neill, J., Pleydell-Bouverie, B., D.Dupret, & Csicsvari, J. (2010). Play it again: reactivation of waking experience and memory. *Trends in neurosciences*, *33*(5), 220 - 229.

Oviedo, S., Vehí, J., Calm, R., & Armengol, J. (2016, October). A review of personalized blood glucose prediction strategies for T1DM patients. *Int J Numer Method Biomed Eng*, *33*(6).

Petroski Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017, Dec). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv e-prints*, arXiv:1712.06567.

Pinsker, J. E., Dassau, E., Deshpande, S., Raghinaru, D., Buckingham, B. A., Kudva,

Y. C., ... Eggerman, T. (2022). Outpatient randomized crossover comparison of zone model predictive control automated insulin delivery with weekly data driven adaptation versus sensor-augmented pump: Results from the international diabetes closed-loop trial 4. *Diabetes Technology & Therapeutics (Ahead of print)*. (PMID: 35549708)

Pirani, M., Dragoni, A. F., & Longhi, S. (2021). Towards sustainable models of computation for artificial intelligence in cyber-physical systems. In *Iecon 2021 – 47th annual conference of the ieee industrial electronics society* (p. 1-8).

Ponsich, A., Azzaro-Pantel, C., Domenech, S., & Pibouleau, L. (2008). Constraint handling strategies in genetic algorithms application to optimal batch plant design. *Chemical Engineering and Processing: Process Intensification*, *47*(3), 420-434. (10th French Congress on Chemical Engineering)

Porter, J. W. (2020). *Sleep, exercise, and insulin sensitivity* (Thesis). University of Missouri–Columbia.

Precup, R.-E., Roman, R.-C., & Safaei, A. (2022). *Data-driven model-free controllers*. CRC Press.

Puigdomènech Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., & Blundell, C. (2020, March). Agent57: Outperforming the Atari Human Benchmark. *arXiv e-prints*, arXiv:2003.13350.

Pérez-Gandía, C., Facchinetti, A., Sparacino, G., Cobelli, C., Gómez, E., Rigla, M., ... Hernando, M. (2010). Artificial neural network algorithm for online glucose prediction from continuous glucose monitoring. *Diabetes Technology & Therapeutics*, *12*(1), 81-88.

Qin, S., & Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control Eng. Pract.*, *11*(7), 733 - 764.

Qiu, C., Hu, Y., Chen, Y., & Zeng, B. (2019, Oct). Deep deterministic policy gradient (DDPG)-based energy harvesting wireless communications. *IEEE Internet Things J.*, *6*(5), 8577-8588.

Raghu, A., Raghu, M., Bengio, S., & Vinyals, O. (2019). Rapid learning or feature reuse?

towards understanding the effectiveness of MAML. *CoRR*, *abs/1909.09157*.

Rawlings, J. B., & Mayne, D. Q. (2009). *Model predictive control: Theory and design*. Nob Hill Pub. Madison, Wisconsin.

Reymann, M. P., Dorschky, E., Groh, B. H., Martindale, C., Blank, P., & Eskofier, B. M. (2016, August). Blood glucose level prediction based on support vector regression using mobile platforms. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, *2016*, 2990–2993.

Robinson, S., Newson, R. S., Liao, B., Kennedy-Martin, T., & Battelino, T. (2021, October). Missed and mistimed insulin doses in people with diabetes: A systematic literature review. *Diabetes Technol Ther*, *23*(12), 844–856.

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017, Mar). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv e-prints*, arXiv:1703.03864.

Schwenzer, M., Ay, M., Bergs, T., & Abel, D. (2021, Nov 01). Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, *117*(5), 1327-1349.

Seo, W., Park, S.-W., Kim, N., Jin, S.-M., & Park, S.-M. (2021). A personalized blood glucose level prediction model with a fine-tuning strategy: A proof-of-concept study. *Computer Methods and Programs in Biomedicine*, *211*, 106424.

Serpanos, D. (2018). The cyber-physical systems revolution. *Computer*, *51*(3), 70-73.

Setlur, A., Li, O., & Smith, V. (2021). Two sides of meta-learning evaluation: In vs. out of distribution. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (Vol. 34, pp. 3770–3783). Curran Associates, Inc.

Shaj, V., Becker, P., Buchler, D., Pandya, H., van Duijkeren, N., Taylor, C. J., ... Neumann, G. (2020). Action-conditional recurrent kalman networks for forward and inverse dynamics learning. *CoRR*, *abs/2010.10201*.

Shaj, V., Büchler, D., Sonker, R., Becker, P., & Neumann, G. (2022). Hidden parameter recurrent state space models for changing dynamics scenarios. In *International*

conference on learning representations.

Shao, L., Yan, R., Li, X., & Liu, Y. (2014, July). From heuristic optimization to dictionary learning: A review and comprehensive comparison of image denoising algorithms. *IEEE Trans. Cybern.*, *44*(7), 1001-1013.

Shi, D., Dassau, E., & Doyle, F. J. (2019). Adaptive zone model predictive control of artificial pancreas based on glucose- and velocity-dependent control penalties. *IEEE Transactions on Biomedical Engineering*, *66*(4), 1045-1054.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... Hassabis, D. (2016, Jan 27). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484-489.

Singh, S. P., Kumar, A., Darbari, H., Singh, L., Rastogi, A., & Jain, S. (2017). Machine translation using deep learning: An overview. In *2017 international conference on computer, communications and electronics (comptelix)* (p. 162-167).

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29* (pp. 1857–1865). Curran Associates, Inc.

Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., & de Garis, H. (1993). An overview of evolutionary computation. In P. B. Brazdil (Ed.), *Machine learning: Ecml-93* (pp. 442–459). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017, July). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. *arXiv e-prints*, arXiv:1707.02968.

Sun, Q., Jankovic, M. V., Bally, L., & Mougiakakou, S. G. (2018). Predicting blood glucose with an LSTM and Bi-LSTM based deep neural network. *CoRR*, *abs/1809.03817*.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, 17–19 Jun). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 1139–1147). PMLR.

Sutton, R. S. (1991, July). Dyna, an integrated architecture for learning, planning, and

reacting. *SIGART Bull.*, *2*(4), 160–163.

Sutton, R. S., & Barto, A. G. (2014). *Reinforcement learning: An introduction*. The MIT Press.

Teney, D., Abbasnejad, E., Kafle, K., Shrestha, R., Kanan, C., & van den Hengel, A. (2020). On the value of out-of-distribution testing: An example of goodhart's law. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 407–417). Curran Associates, Inc.

Thanh, D. N. H., Thanh, L. T., Hien, N. N., & Prasath, S. (2020). Adaptive total variation l1 regularization for salt and pepper image denoising. *Optik*, *208*, 163677.

Toffanin, C., Zisser, H., Doyle, F. J., & Dassau, E. (2013, Jul 01). Dynamic insulin on board: incorporation of circadian insulin sensitivity variation. *Journal of diabetes science and technology*, *7*(4), 928-940.

Tsitsiklis, J., & Van Roy, B. (1997, May). An analysis of temporal-difference learning with function approximation. *IEEE Trans. Autom. Control*, *42*(5), 674-690.

Turksoy, K., Bayrak, E. S., Quinn, L., Littlejohn, E., & Cinar, A. (2013). Adaptive multivariable closed-loop control of blood glucose concentration in patients with type 1 diabetes. In *2013 american control conference* (p. 2905-2910).

van den Oord, A., Li, Y., & Vinyals, O. (2018, July). Representation Learning with Contrastive Predictive Coding. *arXiv e-prints*, arXiv:1807.03748.

van Hasselt, H., Guez, A., & Silver, D. (2015, Sep). Deep Reinforcement Learning with Double Q-learning. *arXiv e-prints*, arXiv:1509.06461.

van Heusden, K., Dassau, E., Zisser, H. C., Seborg, D. E., & Doyle III, F. J. (2012). Control-relevant models for glucose control using a priori patient characteristics. *IEEE Transactions on Biomedical Engineering*, *59*(7), 1839-1849.

Vergari, E., Knudsen, J. G., Ramracheya, R., Salehi, A., Zhang, Q., Adam, J., ... others (2019). Insulin inhibits glucagon release by sglt2-induced stimulation of somatostatin secretion. *Nature communications*, *10*(1), 1-11.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Volpp, M., Flürenbrock, F., Grossberger, L., Daniel, C., & Neumann, G. (2021). Bayesian context aggregation for neural processes. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=ufZN2-aehFa`

Wang, X., & Gupta, A. (2015). Unsupervised learning of visual representations using videos. In *2015 ieee international conference on computer vision (iccv)* (p. 2794-2802).

Woldaregay, A. Z., Årsand, E., Walderhaug, S., Albers, D., Mamykina, L., Botsis, T., & Hartvigsen, G. (2019, July). Data-driven modeling and prediction of blood glucose dynamics: Machine learning applications in type 1 diabetes. *Artif Intell Med*, *98*, 109–134.

Wu, M., Zhuang, C., Mosse, M., Yamins, D., & Goodman, N. (2020). *On mutual information in contrastive learning for visual representations.*

Xie, J., & Wang, Q. (2020). Benchmarking machine learning algorithms on blood glucose prediction for type I diabetes in comparison with classical time-series models. *IEEE Transactions on Biomedical Engineering*, *67*(11), 3101-3124.

Xu, M., Peng, J., Gupta, B. B., Kang, J., Xiong, Z., Li, Z., & El-Latif, A. A. A. (2021). Multi-agent federated reinforcement learning for secure incentive mechanism in intelligent cyber-physical systems. *IEEE Internet of Things Journal*, 1-1.

Xu, S., Lu, B., Baldea, M., Edgar, T. F., Wojsznis, W., Blevins, T., & Nixon, M. (2015). Data cleaning in the process industries. *Reviews in Chemical Engineering*, *31*(5), 453 - 490.

Yao, S., Zhao, Y., Zhang, A., Hu, S., Shao, H., Zhang, C., ... Abdelzaher, T. (2018). Deep learning for the internet of things. *Computer*, *51*(5), 32-41.

Yin, S., Ding, S. X., Xie, X., & Luo, H. (2014). A review on basic data-driven approaches for industrial process monitoring. *IEEE Transactions on Industrial Electronics*, *61*(11), 6418-6428.

Yu, W., & Zhao, C. (2020). Robust monitoring and fault isolation of nonlinear industrial processes using denoising autoencoder and elastic net. *IEEE Trans. Control Syst. Technol.*, *28*(3), 1083-1091.

Zamfirache, I. A., Precup, R.-E., Roman, R.-C., & Petriu, E. M. (2022a). Policy iteration reinforcement learning-based control using a grey wolf optimizer algorithm. *Information Sciences*, *585*, 162-175.

Zamfirache, I. A., Precup, R.-E., Roman, R.-C., & Petriu, E. M. (2022b). Reinforcement learning-based control using q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system. *Information Sciences*, *583*, 99-120.

Zhu, T., Li, K., Herrero, P., & Georgiou, P. (2021). Deep learning for diabetes: A systematic review. *IEEE Journal of Biomedical and Health Informatics*, *25*(7), 2744-2757.