NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

VASCO MIGUEL PESSOA BASTOS

Bachelor's degree in Electrical and Computer Engineering

MODELING OF COMPLEX NONLINEAR DYNAMIC SYSTEMS USING TEMPORAL CONVOLUTION NEURAL NETWORKS

AN HYBRID SYSTEM APPROACH

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2022

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

MODELING OF COMPLEX NONLINEAR DYNAMIC SYSTEMS USING TEMPORAL CONVOLUTION NEURAL NETWORKS

AN HYBRID SYSTEM APPROACH

VASCO MIGUEL PESSOA BASTOS

Bachelor's degree in Electrical and Computer Engineering

Adviser: Paulo José Carrilho de Sousa Gil
Tenure Assistant Professor, NOVA University Lisbon

Co-adviser: Luís Filipe Figueira Brito Palma
Tenure Assistant Professor, NOVA University of Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March, 2022

Modeling of complex nonlinear dynamic systems using temporal convolution neural networks

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.10) [30].

In memory of my grandfather, Manuel Bastos.
This dissertation is completely dedicated to my respectful parents, Marília e Pedro Bastos, my sister Inês, my grandmother Idalina Bastos and my grandparents Conceição e Manuel Pessoa. They always inspire me. I extend my gratitude to rest of my family for motivating me throughout my journey.

Acknowledgements

I would like to express my sincere gratitude, throughout the writing of this dissertation, to everyone who contributed to support, think and discuss with me.

I would first like to thank my supervisor, Professor Paulo Gil, whose expertise was must in formulating research questions, discuss results and exploring new methodologies. Your insightful feedback pushed me to sharpen my thinking and made me aim to challenger objectives.

Following, I would like to extend my thanks to Professor Luís Brito Palma for their valuable guidance throughout my studies. You provided me with the tools that I needed to choose the right direction and successfully complete my dissertation.

I am deeply grateful to my parents, whom without this would have not been possible. You did went above and beyond your responsibilities to provide me all the tools and well being to pursue my objectives. I also appreciate all the support I received from the rest of my family without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

In addition, I would like to thank my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

“I’m going to use all my tools, my God-given ability, and make the best life I can with it. ”
(LeBron James)

Abstract

An increasingly important class of nonlinear systems includes the nonaffine hybrid systems, in particular those in which the underlying dynamics explicitly depends on a switching signal. When the inherent complexity is treatable and the phenomena governing the system dynamics are known an implicit model can be derived to describe its behaviour over time. Conversely, when these assumptions are not met the system dynamics can still be approximated by regression-based techniques, provided a dataset comprising inputs and outputs collected from the system is available. One approach to deal with data driven modelling relies on computational intelligent frameworks, in which artificial neural networks stand out as a prominent class of universal approximation black box models. This work aims to explore 1D Convolutional Neural Networks capabilities, in which the inputs are represented by regressors and structural configuration parameters, to modelling nonlinear hybrid dynamic systems. Moreover, in order evaluate the intrinsic ability to transparently approximate hybrid dynamics, this deep neural network architecture is compared to a shallow multilayer layer perceptron framework, in which each structural configuration is independently approximated.

Keywords: Nonlinear hybrid systems, switching systems, data driven modelling, convolutional neural network, multilayer perceptron

Resumo

Uma classe de sistemas não lineares que tem vindo a ganhar cada vez mais importância é a dos sistemas híbridos não-afins, em particular aqueles em que a dinâmica subjacente depende explicitamente de um sinal de comutação. Quando a complexidade inerente é tratável e os fenómenos que controlam a dinâmica do sistema são conhecidos, é possível obter-se um modelo implícito para descrever seu comportamento ao longo do tempo. Por outro lado, quando essas suposições não são cumpridas, a dinâmica do sistema pode ainda ser aproximada por técnicas baseadas em regressão, desde que um conjunto de dados contendo as entradas e as saídas do sistema esteja disponível. Uma abordagem para lidar com o problema de modelação experimental recorrendo a técnicas de inteligência computacional, na quais as redes neuronais artificiais se destacam como uma das classes proeminentes de aproximadores universais. Este trabalho tem como objetivo explorar as capacidades de redes neuronais convolutivas 1D, onde as entradas são representadas por regressores e parâmetros de configuração estrutural. Além disso, para avaliar a capacidade intrínseca para a aproximação de dinâmicas híbridas, esta arquitetura de rede neuronal profunda é comparada a uma estrutura neuronal proactivas multicamada, na qual cada configuração estrutural é independentemente aproximada.

Palavras-chave: Sistemas híbridos não lineares, sistemas de comutação, modelagem orientada por dados, rede neural convolutiva, *perceptron* multicamada

Contents

List of Figures

List of Tables

Acronyms

This document is incomplete. The external file associated with the glossary 'acronym' (which should be called `output.acr`) hasn't been created.

Check the contents of the file `output.acn`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "output"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "output"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

Symbols

This document is incomplete. The external file associated with the glossary 'symbols' (which should be called `output.sls`) hasn't been created.

Check the contents of the file `output.slo`. If it's empty, that means you haven't indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can't be generated. If the file isn't empty, the document build process hasn't been completed.

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

  ```
  \usepackage[automake]{glossaries-extra}
  ```

- Run the external (Lua) application:

  ```
  makeglossaries-lite.lua "output"
  ```

- Run the external (Perl) application:

  ```
  makeglossaries "output"
  ```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

1 Introduction

The technological evolution had always the need to model dynamical interactions from collected data. If we take a look at the real world we can observe that systems are inherently nonlinear in natures constitution.

"Constructing models from observed data is a fundamental element in science [46]".

If we take a look at the definition of a nonlinear system, it is a system where the input variation does not affect the output in a proportional way. Nonlinear dynamic systems are a point of interest to many scientists and engineers, describing alterations on variables over time, which may seem unpredictable or counter-intuitive.

Physical system modeling is almost a requirement for turning novel ideas into prototypes and, eventually, products. Low-order models are essential to comprehend the interacting behaviors of complex systems at the conceptual phase, and as project continues into prototyping and testing, more advanced models may be necessary to achieve good performance in the assigned tasks[33].

Is it possible to model a system based only on observed data with any information about its structure? The answer is yes, modeling systems that use only input and output data collected from a system, termed black-box identification will be detailed in the next chapter.

One of the technologies that is coming to a new era due to the fast development of computer technology is neural networks. Their popularity and applications are almost infinite and can achieve very satisfying results. Applying this tool to modeling nonlinear dynamical systems may be an interesting path to explore.

Some recent research papers show that convolutional architectures can achieve state-of-the-art accuracy in fields such as audio modeling, machine translation and word-level language modeling [7], [23], [45]. This let us raise the question on how well can a convolutional architecture perform on a complex modeling task.

1.1 Motivation

In the last few years, technology has evolved at an impressive rate. One that had an important role was the computer, today is possible to simulate reality, it is possible to recreate behaviors from real systems. Normally a computer follows a set of rules and if well programmed does the job.

The problem comes with the need to have more subjective rules to be able to represent more complex problems. Many have patterns that are complex, almost unpredictable and its is impossible to define rules for every case. Nonlinear systems are one of this complex problems that need methodologies to help us capture their complex dynamic.

In 2020, *Forbes* published an article saying the following:

"A look at the evolution of the data landscape, how technology is helping businesses solve for now, and how the data analysis that's possible today can provide us with a new understanding of how we, as a society, can perform better in the future[41]."

In a data-driven world that we live in, the urge to pick up data from any system and being able to extract or understand their way of operation is tremendous. Said that, modeling is a task that is more and more necessary.

Either in science as in engineering, data-driven modeling combined with scientific progress creates a environment of addressing many problems with a different approach. Various scientific fields have a certain inherent difficulty in describing phenomena and systems with mathematical equations or laws, so the have been turning to artificial intelligence as a powerful approach in data-driven modeling[35].

Machine learning is a branch of AI that has gained popularity over the years. In the present research environment, we see applications from image processing to many other data analyses. Especially deep learning architectures have an huge importance due to using graph technologies side by side with neuron transformation to get multilayer learning model that easily learns through the data[40].

Inspired on human neurons and in the way the learn and react to different stimulus, the application of artificial neural networks could be interesting way to deal with this problem. These architectures have shown real capabilities in handling nonlinear problems.

The main motivation to develop this thesis is to be able to discover more about the integration of machine learning techniques with the ability to capture and predict an hybrid nonlinear dynamic behavior. The focus will be on temporal convolutional neural network models and their strength deal with temporal data to test their ability to provide a good system approximator.

1.2 Problem Context

Nowadays in fields such as the industry 4.0, science and engineering we are observing an environment of technological evolution and digital transformation. Artificial intelligence is being applied to almost every object we know. Said that, the control field makes the connection between the software adaptability and the real world.

A point that can have a huge impact on industry, the accuracy of modeling and prediction of systems behaviors as in many fields, is the ability to incorporate structural or model reconfigurations into the description of nonlinear systems.

The urge to have models capable to understand and capture dynamics that many times have a complex nonlinear behavior is leading many fields to pursue the search for better tools.

One of the methodologies to explore is artificial neural networks. They are gaining a lot of popularity again because of their good results and due to the evolution in computer hardware, which allows heavier computing power.

Using convolutional neural networks to capture a real system dynamics and be able to represent it could lead to cost reductions, increasing effectiveness and time saving. Moreover, the ability to be able to represent a nonlinear process is a key point in optimization for numerous fields.

1.3 Objectives

One of the approaches, which is around for decades ([22, 46]), relies on data collected from the system and on regression techniques, associated with black-box structures. Among possible black-box nonlinear structures neural networks have emerged as an important class of universal approximators [3]. For instance in [18]. showed that shallow multilayer feedforward neural networks (MultiLayer Perceptron (MLP)s) with one hidden layer, using arbitrary squashing functions, are able to approximate any Borel measurable function from one finite dimensional space to another, with arbitrary degree of accuracy.

In the last few years a new type of neural network topologies has emerged, the so called deep architectures. They are composed of a multitude of hidden layers, each containing possibly a huge number of processing units. A special case of these topologies is the convolutional neural network (Convolutional Neural Network (CNN)). These topologies are essentially multi-layer neural networks, in which each layer is composed of a number of two-dimensional planes with independent neurons, being a sparse connection used between layers ([50]). Recently, it has been proven that, like MLPs, CNNs are also universal approximators [52].

The objective of this dissertation relies on the development of a convolutional neural network architecture capable of modeling a complex nonlinear dynamic system. The architecture should handle the spatial-temporal data as well as the structural data.

Also, a Multilayer Perceptron architecture will be implemented to compare and analyze the results from the Convolutional architecture. The MLP architecture is composed by n MultiLayer Perceptron neural networks for each structural configuration while the convolutional architecture handles all configurations.

Taking all of this into account, the present work aims to investigate the performance of CNNs in the context of nonlinear hybrid system identification, by proposing a particular CNN topology and validating the approach on a benchmark system. Moreover, in the proposed framework the asynchronous event-driven switching signal is transparently provided to the network as an additional input, while each subsystem dynamics is internally approximated considering spatio-temporal information, under the form of regressors and the underlying outputs. The proposed identification framework is compared against a shallow MLP topology, in which the switching between structural configurations is externally managed by assigning the current configuration to the corresponding MLP approximator.

To accomplish this objectives, it was developed a convolutional neural network architecture in *Python 3.9.7* using libraries such as *TensorFlow 2.8.0*, *Keras 2.8.0* with *Keras Preprocessing 1.1.2* and *Scikit-learn 1.0.2*.

1.4 Contributions

The present work presents a few contributions:

- Ability to be able to represent a nonlinear process is a key point in optimization for numerous fields,
- Supports systems that have structural reconfiguration,
- Could be a tool that is very effective in testing industrial processes in simulation, which translate in cost reduction,
- Development and implementation a CNN topology oriented to the context of nonlinear hybrid system identification. In the proposed framework the asynchronous event-driven switching signal is transparently provided to the network as an additional input, while each subsystem dynamics is internally approximated considering spatio-temporal information,
- Development and implementation a shallow MLP topology oriented to the context of nonlinear hybrid system identification where the structural configurations is externally managed by assigning the current configuration to the corresponding MLP approximator,

1.5 Document Structure

This dissertation is organized in the following way:

- This introductory chapter (Chapter 1),
- Initially some literature review has been made in order to collect information about the dissertation subject and which approaches could we use to reach our goal in the second chapter, State of the Art (Chapter 2),
- Then, we have the third chapter (Chapter 3) related to hybrid system, their identification, and proposed architecture.
- The fourth chapter (Chapter 4) is purely dedicated to the Case Study approach, where we apply the proposed architecture to the three-tank system. It shows how the system can be described and processed by the neural networks,
- Finally, the last chapter (Chapter 5) where we make some final considerations about the results and where the conclusions are drawn. Also, some prospective lines of research are presented.

2 State of the Art

2.1 Nonlinear Black-box Modeling in System Identification

2.1.1 Nonlinear System Identification: Black-box Modeling

The search for a desirable model structure where we can build a good model is one of the main problems in system identification. In most cases, it is easier to fit a model in given structure / parameter estimation than searching for a model structure.

"A basic rule in estimation is not to estimate what you already know. In other words, one should utilize prior knowledge and physical insight about the system when selecting the model structure"[46].

To make the distinction between the type of model structure, it is usually color encoded in levels of prior knowledge2.1:

- White-box
- Grey-box
- Black-box

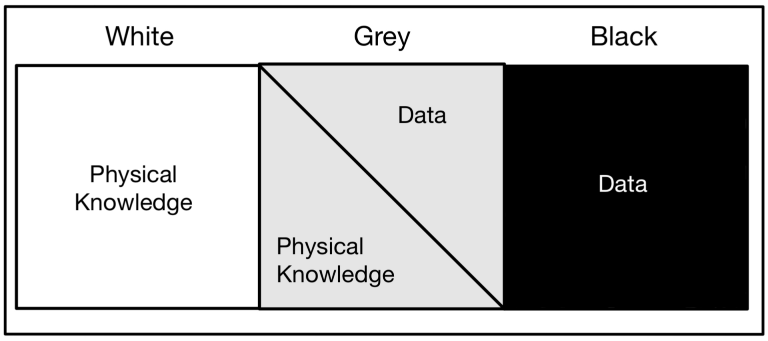


Figure 2.1: Illustration of the white-box, grey-box and black-box concepts (adapted from [9]).

2.1.1.1 White-box Models

The white-box models are mainly based system knowledge, the model is completely known and its construction only requires prior knowledge and physical grounds. For example, deterministic equations, detailed sub models and physical knowledge.

2.1.1.2 Grey-box Models

Grey-box models could be viewed as a white-box and black-box combination. Sometimes all the physical insights are not available, therefore it is necessary to estimate a-few parameters bond on collected data. Mainly there are two typical sub cases:

- Physical modeling $\Rightarrow$ Based on the laws of physics, but with some parameters which require to be found on data.
- Semi physical modeling $\Rightarrow$ Using physical perceptiveness, nonlinear combinations are build from measure data signals.

2.1.1.3 Black-box Models

Black-box models can be used in many structures based on, for example neural networks, gradient boosting models, fuzzy models and others [46]. Most of them very often provide great accuracy. No physical insight is necessary.

One simple way to divide black-box models is break them up in two classes, linear models and nonlinear models.

On one hand, we have the linear black-box models, where time series models and transfer functions prevail. To find the parameters of the black-box model, we just need data and various techniques used to find linear parameters. On the other hand, nonlinear models are much more complex compared to the linear ones. The explanation behind this situation is that all is considered, resulting in wide spectrum of possible model descriptions. In this category, time-series features are predominant and often they are combined with neural network models. It is important to note the increasing use of neural networks in building models due to the accessibility of computer power and the evolving technologies.

2.1.2 Nonlinear black-box - structure

When working with nonlinear black-box models, it may be assumed that a system identification problem starts with the series inputs such as $u(t)$ and $y(t)$ observed from the system dynamic [46].

$$u^t = \begin{bmatrix} u(1) & u(2) & u(3) & \dots & u(t) \end{bmatrix} \tag{2.1}$$

$$y^t = \begin{bmatrix} y(1) & y(2) & y(3) & \dots & y(t) \end{bmatrix} \tag{2.2}$$

Analysing past observations $[u^{t-1}, y^{t-1}]$ and the outputs $y(t)$ yet to be known, the goal is to find the underlying correlation [46]:

$$y(t) = g(u^{t-1}, y^{t-1}) + v(t) \tag{2.3}$$

From equation 2.3, it is easily noticed a new term $v(t)$. As expected the output $y(t)$ will not be exactly like past observations, it may have a few changes, so this addiction comes in compensation. This value should be as small as possible, to be sure that $g(u^{t-1}, y^{t-1})$ is a good representation of past data and consequently a good prediction.

The other important factor is the function $g(u^{t-1}, y^{t-1})$, and our next step is to use a finite dimensional parameter to parameterize this function, mind that this is an approximation [46].

$$g(u^{t-1}, y^{t-1}, \theta) \tag{2.4}$$

To assess θ, it should be formulated the optimization problem. Using means to fit the recorded data with the model, it obtains the quality of θ[46]. Said that the optimization problem can be defined by:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} V(t) \tag{2.5}$$

$$V(t) = \sum_{t=1}^{N} \left\| y(t) - g\left(u^{t-1}, y^{t-1}, \theta\right) \right\|^2 \tag{2.6}$$

Due to the generality of the model structure, it could be good to rewrite the function g as a combination between two mapping: one that takes past entries u', y' transform them and maps a finite dimensional vector φ and another one which uses the vector to map the outputs [46]:

$$g(u^{t-1}, y^{t-1}, \theta) = g(\varphi(t), \theta) \tag{2.7}$$

where the variable φ is as follows [46],

$$\varphi(t) = \varphi(u^{t-1}, y^{t-1}, \eta) \tag{2.8}$$

This equation is a parameterized version of the regression vector. Therefore, the new short form of φ (used in (2.7)) is $\varphi(t,\eta)$.

Now we will address the two decomposed problems regarding the nonlinear mapping in (2.4):

- $\varphi(t) \Rightarrow$ Regression vector from past I/O.
- $g(\varphi) \Rightarrow$ Nonlinear mapping of the regressor on to the space of outputs.

2.1.3 Nonlinear black-box modeling: Regressors

Lets consider the following structure for the nonlinear black-box modeling [46]:

$$y(t|\theta) = g(\varphi(t), \theta) \tag{2.9}$$

Note that in the equation 2.9 $g(\varphi(t),\theta)$ refer to the parameterized function.

The regression vector $\varphi(t)$ is characterized by [46]:

$$\varphi(t) = \begin{bmatrix} u(t-1) & u(t-2) & u(t-3) & \dots & u(t-k) \end{bmatrix} \tag{2.10}$$

Nonlinear models				
Model	Regressors			
NFIR	$u(t-k)$			
NARX	$u(t-k)$	$y(t-k)$		
NOE	$u(t-k)$	$\hat{y}_u(t-k\|\theta)$		
NARMAX	$u(t-k)$	$y(t-k)$	$\varepsilon(t-k\|\theta)$	
NBJ	$u(t-k)$	$\hat{y}(t-k\|\theta)$	$\varepsilon(t-k\|\theta)$	$\varepsilon_u(t-k\|\theta)$

Table 2.1: A few nonlinear models and possible regressors [46].

The previous models abbreviations are reffering to Nonlinear Finite Impulse Response model (NFIR), Nonlinear autoregressive exogenous model (NARX), Nonlinear Output Error model (NOE),Nonlinear Autoregressive Moving Average with eXogenous inputs model (NARMAX) and Nonlinear Box–Jenkins model (NBJ).

Additional notes to the previous table:

- In the NOE models, the output is $\hat{y}(t|\theta)$.
- In the NBJ models, using the equation (2.10), we get the simulated output $\hat{y}_u$ by exchanging ε and ε_u with a regression vector $\varphi(t,\theta)$ with zeros.

Following Ljung[29] perspective on system identification and Sjöberg[46], common model can be generalized by:

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t) \tag{2.11}$$

Said that, in table 2.1:

- $u(t-k)$ is related with polynomial B,
- $y(t-k)$ is related with polynomial A,
- $\hat{y_u}(t-k)$ is related with polynomial F, predicted output from past u,
- $\epsilon(t-k) = y(t-k) - \hat{y}(t-k|\theta)$ is related with polynomial C, which is the prediction errors,
- $\epsilon_u(t-k) = y(t-k) - \hat{y_u}(t-k|\theta)$ is related with polynomial D, which is the prediction errors,

2.1.4 Nonlinear mapping

For the nonlinear mapping the objective here is for any θ it goes from $\mathbb{R}^d$ to $\mathbb{R}^p$. Independently of the chosen regression vector $\varphi = \begin{bmatrix} \varphi_1 & \dots & \varphi_d \end{bmatrix}^T$, this will be a vector in $\mathbb{R}^d$[46].

$$g(\varphi,\theta) \tag{2.12}$$

The parameterized function extension is denotated by[46]:

$$g(\varphi,\theta) = \sum \alpha_k g_k(\varphi) \tag{2.13}$$

where g_k is a basis function.

The expansion above with different basis function and various choices of regressors, will act like an unified framework for working on many nonlinear black-box structures.

2.2 Artificial Neural Networks

The initial idea behind artificial neural networks (ANN) began as a machine learning model inspired by the networks of biological neurons found in our brains. The first neuron model was developed by McCulloch and Pitts in 1943[34]. This model was composed of an artificial neuron that did a weighted sum of the inputs and, according to the value of that sum, the neuron is activated or not. Since 1943, Artificial Neural Network (ANN)s have evolved a lot and this section describes neural network models and their applications.

In this kind of technology, a network that has weights on it that can be adjusted during the training process to achieve the greatest results, has real capabilities. ANNs are currently being applied to a vast number of fields, such as in image recognition, time-series, computer vision, in control applications and many others[32].

"The success of ANNs arises from their ability to effectively learn static representations from complex data and in building relationships between features and outputs"[36].

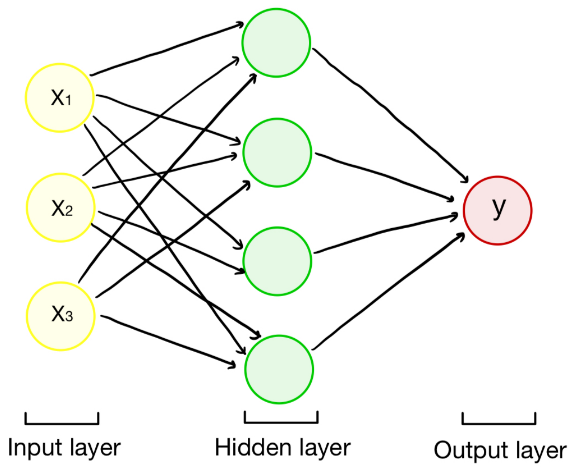


Figure 2.2: Illustration of simple neural network (adapted from [12]).

One of the simplest ANN architectures is the perceptron (check Fig. 2.2 and Fig. 2.3). The inputs neurons and other neurons have connections between them with a weight associated. Then the perceptron makes a weighted sum with the inputs [12].

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{X}^T W \tag{2.14}$$

after that, it uses a step function and outputs the outcome [12]:

$$h_w(\mathbf{X}) = step(z) \tag{2.15}$$

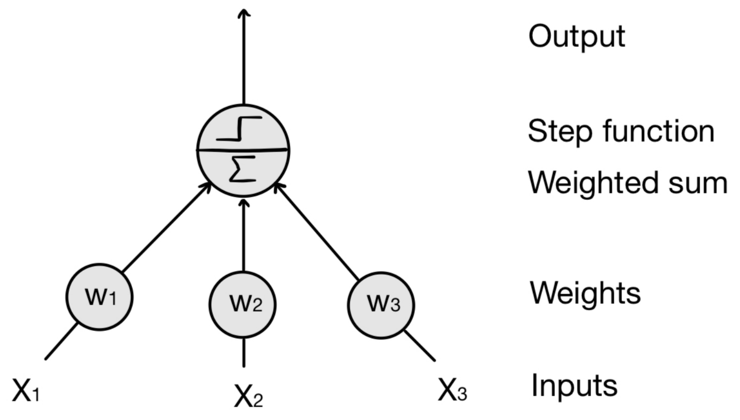


Figure 2.3: Architecture of a perceptron with a single neuron.

Usually the common step functions are heaviside step function and the sign function (shown below)(assuming threshold = 0)[12].

$$heaviside(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \qquad sgn(z) = \begin{cases} -1, & \text{if } z < 0 \\ 0, & \text{if } z = 0 \\ 1, & \text{if } z > 0 \end{cases}$$

Lets look at a case of a perceptron neural network built to classify the inputs into three binary classes (which is a classifier multi-output) in Fig. 2.4:

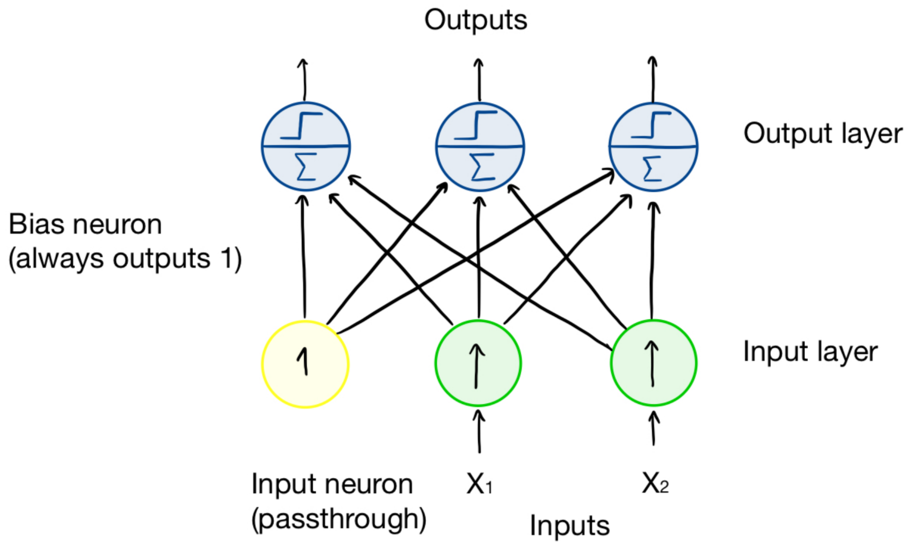


Figure 2.4: Perceptron architecture composed with two input neurons, one bias neuron and three output neurons (weights omitted) (adapted from [12]).

In order to get the output of a fully connected layer of neurons for multiple instances in perceptron architecture, we have [12]:

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{W}\mathbf{X} + \mathbf{b}) \tag{2.16}$$

X is the matrix of inputs. With one column per feature and one row per instance. **W** is naturally a matrix with the respective connection weights except the ones referring to the bias neuron. Regarding the bias vector **b**, it has all weights associated with connections from the bias neuron to other artificial neurons.

The inspiration for the perceptron training algorithm comes from *Hebb's Rule* and it was introduced by Rosenblatt in 1958.

"**The Hebb's Rule** is a learning rule that describes how the neuronal activities influence the connection between neurons, i.e., the synaptic strength. It provides an algorithm to update weights of neuronal connections within neural network."[12]

Later summarized in the catchy phrase "Cells that fire together, wire together", that is, when two neurons fire up at the same time their connection weights are likely to increase. Based on this rule the perceptrons are trained using a similar method, the rule reinforces the connections that help reduce the error [12].

The perceptron learning rule is formulated as [12]:

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta \left(y_j - \hat{y}_j \right) x_i \tag{2.17}$$

To better understand the previous equation lets set a ground on the nomenclature, the i^{th} refers to the th input neuron same for j^{th} output neuron. So x_i is the value of the current input of the training instance, while $\hat{y}_j$ is the value of current output of the training instance. The other y_j is the target value to be achieve on that training instance. The learning rate is given by η.

Taking into account that the boundaries of each output neuron are linear, the perceptrons are unable to learn more complex designs (for example classifiers like Logistic Regressors). Its important to note that perceptrons do not output classes probabilities, they make decisions based on a threshold.

One solution to the perceptrons limitations is to stack multiple perceptrons. This result in a kind of ANN is called Multilayer Perceptron (MLP) [12].

It will be addressed three particular artificial neural networks:

1. MLP
2. Recurrent Neural Network (RNN)
3. CNN

2.2.1 Multilayer Perceptron (MLP)

The first kind that we are going to look is the Multilayer Perceptron neural network (MLP). A MLP has in its constitution one (passthrough) input layer, can have a few hidden layers and one last layer named output layer. Excluding the output layer (mind a few particular cases), every layer will have a bias neuron even if not shown and is fully connected to next in line(see Fig. 2.5).

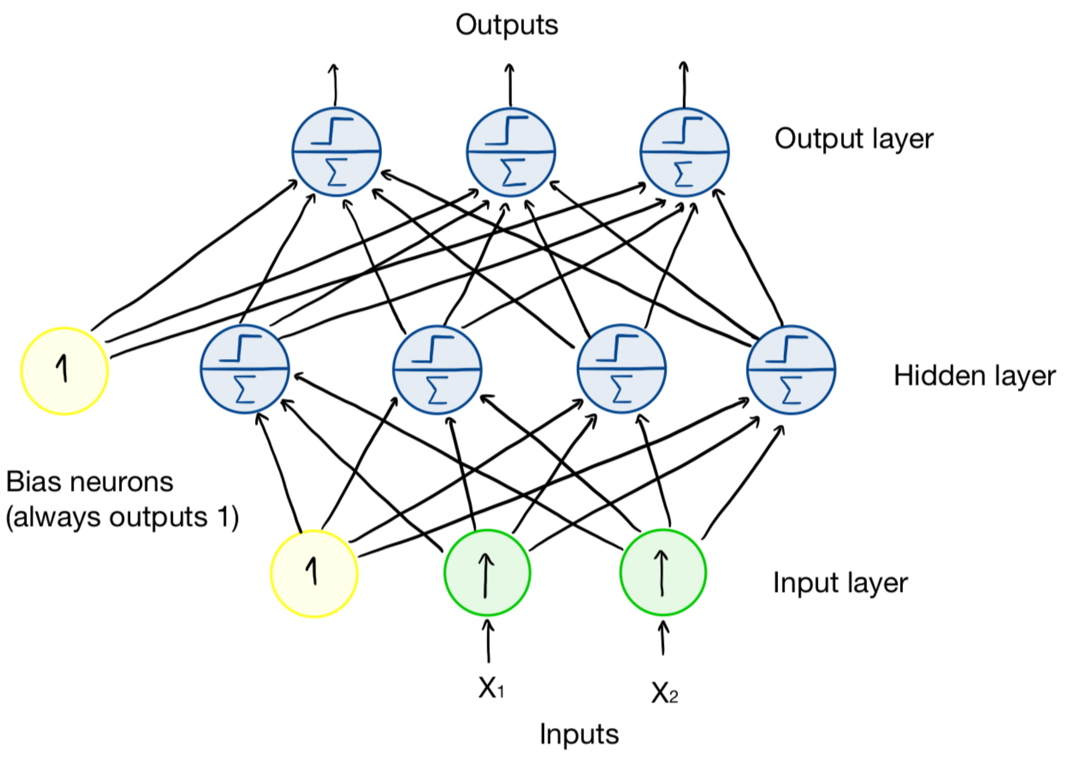


Figure 2.5: MLP Architecture composed with a two input neurons, two bias neurons, one hidden layer and three output neurons.

2.2.1.1 Training MPLs

For a several years, researchers had struggled in the search for a training method for MLPs, without success. In 1986, a paper that was cutting edge to the time, introduced a new learning method, the back-propagation, for networks with neuron units. This method repeatedly try to adjust the weights of connections to minimize a measure of the difference between the target output and the result that we are getting [39].

Let's take a closer look on the back-propagation algorithm:

- The algorithm goes through the training set various times (epochs) handing a batch at a time.
- In each epoch, this batch is fed to the input layer which passes it through to the first hidden layer. Then each neuron computes each output and forwards to the layer in line until it reaches the output layer.
- The next step is to calculate the predicted output error (desired output vs actual output).
- Now begin the back-propagation part, the algorithm computes the contribution of each output to the error.
- Then it computes the error contributions of each connection to layer below. Working backward until it reaches the input layer. The reverse pass calculates the error gradient of the network.
- Finally, in order to tweak the weights of the connections, it performs a gradient descent step with all gradient calculated.

That summarizes the back-propagation algorithm. To make this algorithm work properly, it is not possible to use step function because they are flat segments, so the gradient descent will not move on flat area. To solve this, replace the activation function with a function with a well-defined non-zero derivative (see Fig. 2.6).

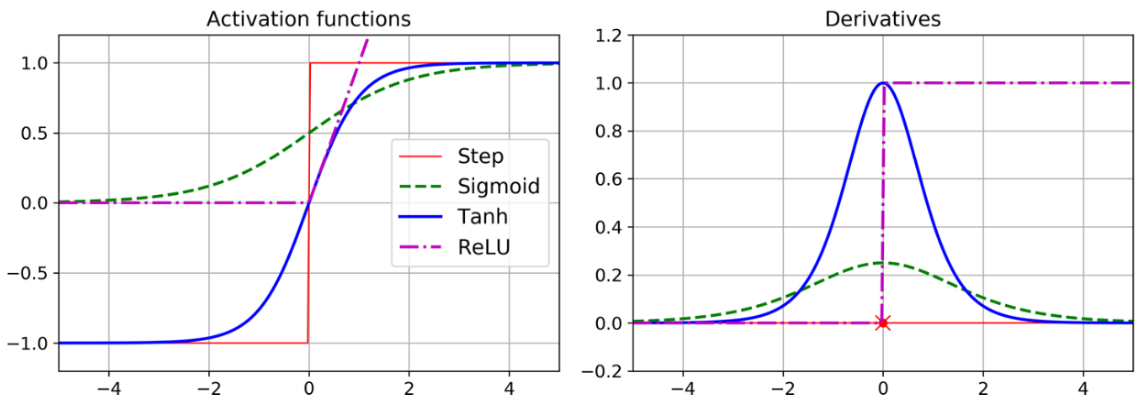


Figure 2.6: Activation functions and their derivatives: MLP [12]

In case of classification problems or regression prediction problems, MLPs are very suitable. They are flexible and generally good for basic operations such as data visualization, encryption and data compression.

2.2.2 Recurrent Neural Network(RNN)

Until now, we have seen feed forward architectures which means that the information only flows from the input in direction to the output layer. In this section, it will be covered a ANN known as recurrent neural network.

"Recurrent Neural Network (RNN) is a Deep learning algorithm and it is a type of Artificial Neural Network architecture that is specialized for processing sequential data"[19].

Imagine a network similar to the one previously discussed but with the particularity that it also has backward connections. Looking at the smallest RNN possible, network with one recurrent neuron which will receive an input, compute an output outcome and then propagate it. To better understand this consider the illustration in Fig. 2.7.

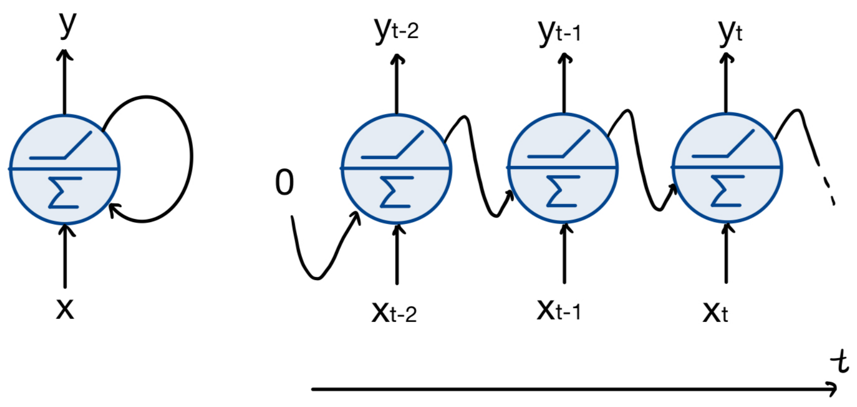


Figure 2.7: Illustration of a simple neuron recurrent neural network. On the left we the simple neuron and on the right its progression through the time frames.

It is noticeable that the output from the last time frame is used as an input to the next. frame. Usually in the first time frame, the first last output is considered zero.

As we should expect, recurrent neurons also have weights associated with the connections between neurons. In this case, we will have two types of weights: the ones referring to the inputs of the neuron and others to balance the previous outputs.

Another point to cover is grouping neurons to create a recurrent layer. It works exactly in the same way as before but instead of a having an input value and the previous output value, it will receive a couple of vectors, one for inputs values for the neurons of that layer and a previous output vector with the values regarding the last time frame (see Fig. 2.8).

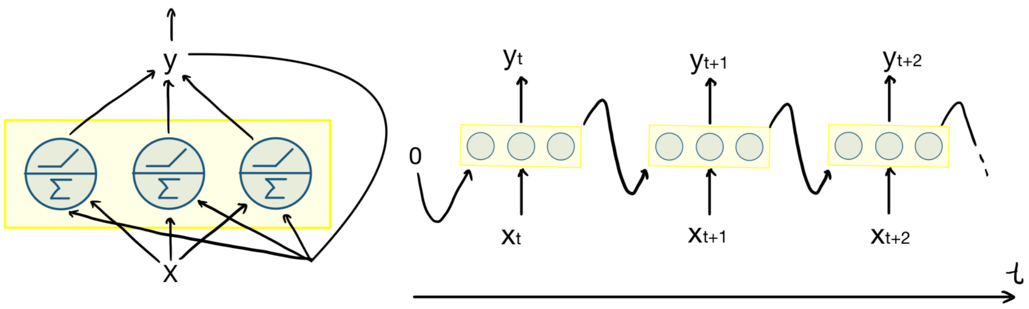


Figure 2.8: Illustration of a neuron layer in a recurrent neural network. On the left we the a layer of neurons and on the right its progression through the time frames.

Logically, the weights associated with the connection are now the respective vectors with weights matrices, $\mathbf{W}_x$ and $\mathbf{W}_y$. To compute the outputs of a recurrent layer, we use the following equations, (2.16) for a single instance and (2.17) for multiples instances[12].

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x^\top \mathbf{x}_{(t)} + \mathbf{W}_y^\top \mathbf{y}_{(t-1)} + \mathbf{b}\right) \tag{2.18}$$

$$\begin{aligned}\mathbf{Y}_{(t)} &= \phi\left(\mathbf{W}_x \mathbf{X}_{(t)} + \mathbf{W}_y \mathbf{Y}_{(t-1)} \mathbf{W}_y + \mathbf{b}\right) \\ &= \phi\left(\mathbf{W}\begin{bmatrix} \mathbf{X}_{(t)} \\ \mathbf{Y}_{(t-1)} \end{bmatrix} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \mathbf{W}_y \end{bmatrix}\end{aligned} \tag{2.19}$$

Regarding the previous equation:

- Starting with $\mathbf{Y}_{(t-1)}$, it corresponds to the layer output matrix for the t time frame, for each instance in a batch (m - instances times n - neurons).
- The $\mathbf{X}_{(t)}$ refers to the input matrix of all instances. (n -> inputs features times m -> instances).
- $\mathbf{W}_x$ is the matrix of the weights focused of the inputs.
- $\mathbf{W}_y$ is the matrix of connection weights minding the previous time frame output.
- $\mathbf{b}$ is again the vector for the bias neuron values.
- Note that $\mathbf{W}_x$ and $\mathbf{W}_y$ are combined into a matrix of weights $\mathbf{W}$ sized (n -> inputs plus n -> neurons) times n -> neurons.

Since RNNs can learn from the previous outputs at each time frame and propagate it through the time framework, it can be said that it has memory capability. A zone in a neural network that saves some kind data across the time is named *memory cell*.

2.2.2.1 I/O Sequencing

RNNs can tolerate a few types of input / output sequencing resulting in different models of network such as one to one network, one to many network, vector-to-sequence and encoder / decoder network.

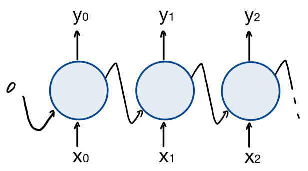
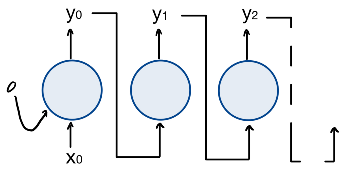
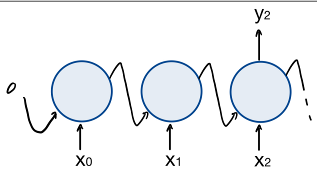
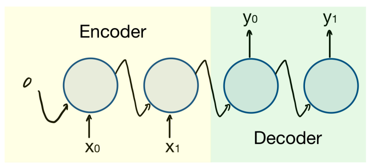
Type of RNN	Illustration	Example
Seq-to-Seq		Name entity recognition
Vector-to-Seq		Music Generation RNN
Seq-to-Vector		Sentiment classification
Encoder-Decoder		Machine translation

Table 2.2: Types of RNNs and examples.[1][12]

Note that the encoder-decoder network is used in implementations where the need to see a batch of data to understand it, for example, to make a accurate translation it is necessary to know at least the full sentence.

2.2.2.2 Training RNNs

Following the MLPs training logic, the algorithm behind the training process is called back-propagation through time (BPTT). Similar to the normal back-propagation. There is a first pass through the network. Then a cost function is calculated based on the output. Mind that the cost function can ignore some outputs dependent one the network. After that all the gradients of the cost function go in backwards through the network. To conclude it updates the the model.

2.2.3 Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) were pioneered by Yann LeCun, in 1980s/1990s [6]. The subjective experience is not to be trusted, perception is a hard thing to describe, it's not trivial at all [12]. A CNN form is inspired on the connectivity pattern of neurons and in the organization of the human visual cortex. One of the very first CNNs, LeNet5 was a result from the LeCun's work [27]. The LeNet5 architecture was fundamental to the research and development of more complex models. The enormous growth in computer power made possible the search for more architectures and more applications of CNN, such as in computer vision, image/video recognition, classification, natural language processing and time-series analysis [42] [43].

2.2.3.1 Convolutional Layer

In a CNN constitution, the principal block is the convolutional layer, in opposition to the previous ANN architectures, these layers only establish connections within their respective receptive fields instead of being connected to every single unit. These operations allow the concentration of small features in the first layer and then move on to larger and higher-level features in the next layers(Fig. 2.9).

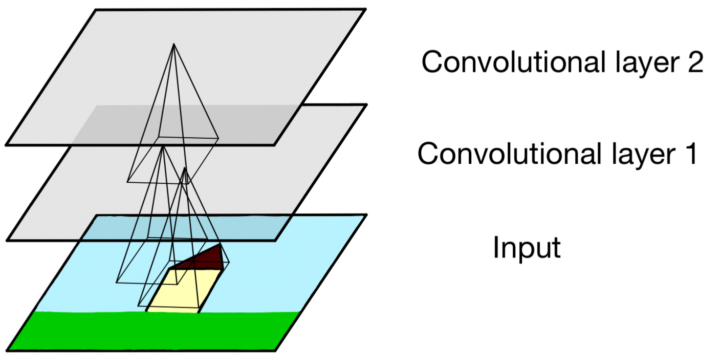


Figure 2.9: Representation of convolutional layer and a receptive field between layers (adapted from [12]).

Imagine that you want to establish a connection between neuron (row i, column j) in a certain layer and the group of neurons within the receptive field in the previous layer (row i to i + f_h - 1, column j to j + f_w - 1). In the figure below (Fig. 2.10) f_h and f_w refer to height and width of the receptive field.

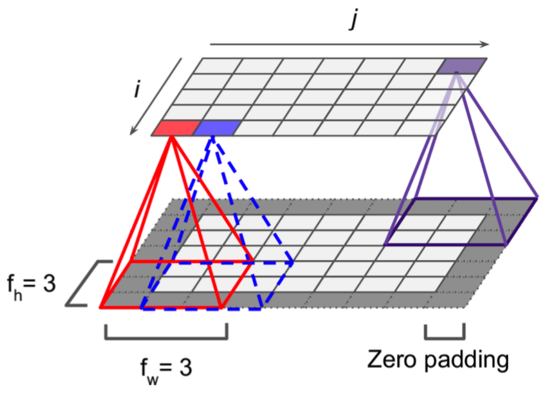


Figure 2.10: Connections between convolutional layers with zero padding [12].

In order to reduce the model complexity, it is possible to define a stride parameter which means to shift one or more receptive fields in line. This results in significant reduction in the model computational complexity. To establish connection between neuron (row i, column j) in a certain layer and the group of neurons within the receptive field in the previous layer (row i x s_h to i x s_h + f_h - 1, column j x s_w to j x s_w + f_w - 1). In the figure below (Fig. 2.11)s_h and s_w are referring to the strides, both horizontal and vertical.

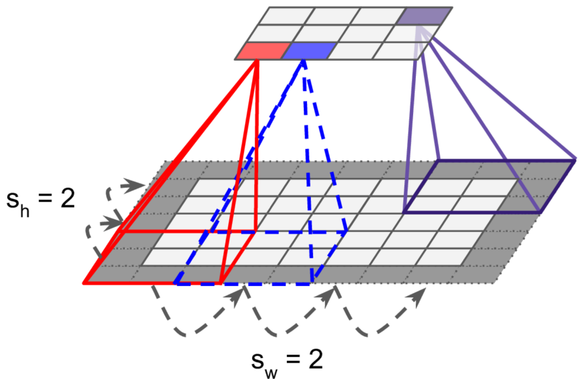


Figure 2.11: Dimensionality reduction with a stride of two[12].

2.2.3.2 Filters

Filters or convolution kernels are the representation of weights in CNNs. They are the size of the receptive field. Take a look at the filter below (Fig. 2.12):

0	1	0
1	1	1
0	1	0

Figure 2.12: Example of a 3x3 filter to be applied in order to get a feature map.

The use of the same filter through the neurons in convolutional layer will output a feature map, which means that areas where the filter has 1's will be enhanced in contrast to where the filter has 0's.

2.2.3.3 Multiple Feature Mapping

A convolutional layer can have multiple filters which will result in an output of one feature map per filter. One neuron per pixel in each feature map and each neuron has the same parameters. A convolutional layer applies multiple filters which will be trained with inputs in order to be able to detect multiple features across data (see Fig. 2.13).

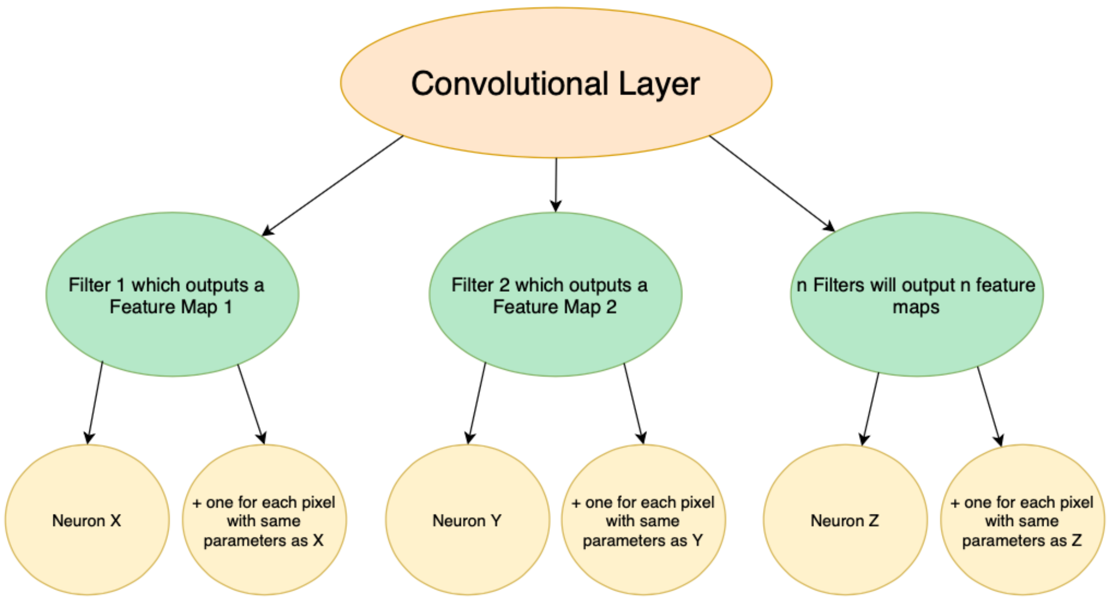

Convolutional Layer
Filter 1 which outputs a Feature Map 1
Filter 2 which outputs a Feature Map 2
n Filters will output n feature maps
Neuron X
+ one for each pixel with same parameters as X
Neuron Y
+ one for each pixel with same parameters as Y
Neuron Z
+ one for each pixel with same parameters as Z

Figure 2.13: Diagram of a convolutional layer.

To calculate the output of a neuron in a convolutional layer, it can used the following equation [12]:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \tag{2.20}$$

Regarding the previous equation:

- Staring with $z_{i,j,k}$, corresponds to the neuron (row i, column j, feature map k, layer l) output.
- $x_{i',j',k'}$, corresponds to the neuron (row i', column j', feature map k', layer - 1) output.
- $\mathbf{b}_k$ is a bias term per feature map k in layer l.
- $w_{u,v,k',k}$ is the weight of the connection between a neuron (feature map k, layer l) and the input located at row u, column v (relative to the receptive field), and the feature map k'.

2.2.3.4 Dropout layer

A very common element in artificial neural network architectures is the dropout layer. It simply drops out or randomly ignores a unit, which means temporarily hide them from the network, plus its connections to other elements[47].

"With unlimited computation, the best way to regularize a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data"[47].

Dropout relies on nodes in a layer to assume less or more responsibility for the inputs during the training procedure making it noisier, on a probabilistic basis.

Dropout will attempt to deal with instances in which network layers co-adapt to rectify previous layers' mistakes, resulting in a more robust model.

If we look at keras dropout leyer class [8], the dropout layer is meant to avoid over-fitting during training process, it will set the input to 0 with certain frequency f at each time step.

The other inputs are scaled up by $1/(1-f)$ such that the sum over all inputs is unchanged.

2.2.3.5 Pooling layers

Usually used in CNN, pooling layers are used to consolidate features learned by the convolutional layer feature map. It helps preventing over-fitting. Pooling layers are not very complicate because frequently it only involves average or maximum values of input to downsample data[48].

Commonly used pooling layers:

- Max Pooling layer $\Rightarrow$ Usually used to extract low-level features from data, max pooling chooses the maximum values in the section captured by the filter in any feature map.

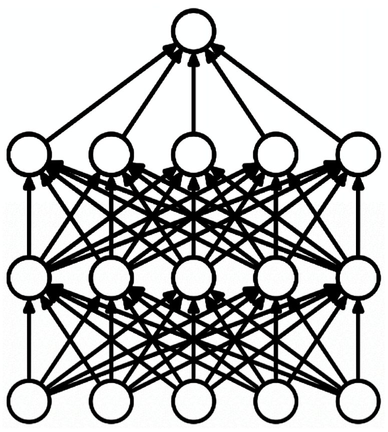


Figure 2.14: Standard Neural Net: Before Dropout

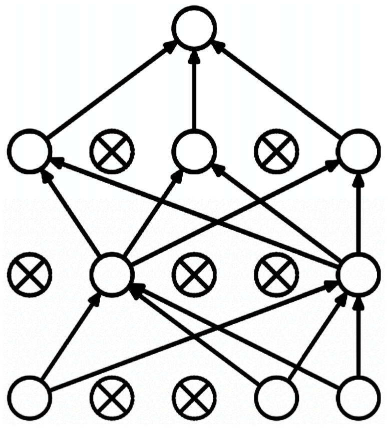


Figure 2.15: Standard Neural Net: After Dropout

- Average Pooling layer ⇒ This layer is used to helps to extract the smooth features, average pooling chooses the average of values in the section captured by the filter in any feature map.
- Global Max/Average Pooling layer ⇒ Both global average and global max layers are used in place of the the flatten layer to prevent over-fitting. Also, the global average pooling can be viewed as a structural regularizer that enforces features maps to be confidence maps of categories. Basically, the flobal pooling layer take the max or average values of each features map and send them to the activation layer.

The pooling operation (see Fig. 2.14 and Fig. 2.15) consists in passing a two dimensional filter across a three-dimensional feature map and sum up the features that come in shape of filter. Said that, if it is used a feature map sized $h * wc$, the output retrieved from the pooling is (see Fig 2.16):

$$dim(output) = (h - f + 1)/s * (w - f + 1) * c \tag{2.21}$$

In which variable represents:

- h is the height of the feature map
- w is the width of the feature map
- c is the feature map channel
- s is the length of the stride

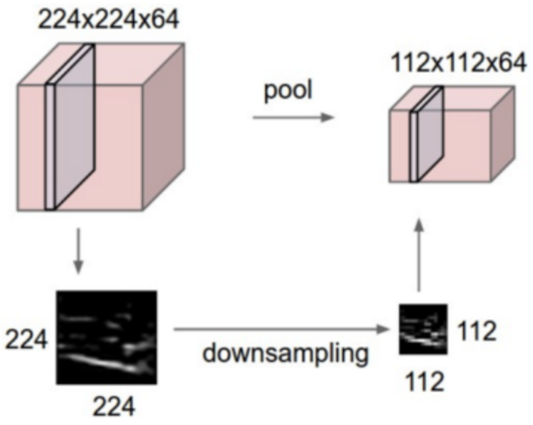


Figure 2.16: Overview of a pooling layer and how it downsamples data[48].

2.3 Performance metrics

In this section, it will be reviewed some performance metrics used during this dissertation, namely Mean Square Error (MSE), Mean Average Error (MAE) and Pearson Correlation Coefficient (PCC).

2.3.1 Mean Square Error (MSE)

The mean square error is aimed to measure the amount of error in model. It calculated the average square difference between the observed values and the predicted values.

For instance, if we want to predict the value of an unobserved variable X given that we observed $Y = y$. We can say that:

$$\hat{x} = g(y) \tag{2.22}$$

The error in our predict is given by:

$$\tilde{X} = X - \hat{x} = X - g(y) \tag{2.23}$$

The MSE of an estimator is given by:

$$E\left[(X-\hat{x})^2 \mid Y=y\right] = E\left[(X-g(y))^2 \mid Y=y\right] \tag{2.24}$$

The objective to minimize the MSE in order to get the best prediction, we are looking for the estimator with the lowest MSE among all possible estimators:

$$\hat{X_M} = E[X \mid Y] \tag{2.25}$$

2.3.2 Mean Average Error (MAE)

The mean average error measures the average magnitude of errors in model. MAE is teh average over the absolute values of the difference between predicted values and observed values.

For instance, if we want to predict the value of an unobserved variable X given that we observed $Y=y$. We can say that:

$$\hat{x} = g(y) \tag{2.26}$$

The error in our predict is given by:

$$\tilde{X} = X - \hat{x} = X - g(y) \tag{2.27}$$

The MAE of an estimator is given by:

$$E\left[|X-\hat{x}| \mid Y=y\right] = E\left[(|X-g(y)| \mid Y=y\right] \tag{2.28}$$

The objective to minimize the MAE in order to get the best prediction, we are looking for the estimator with the lowest MAE among all possible estimators:

$$\hat{X_M} = E[X \mid Y] \tag{2.29}$$

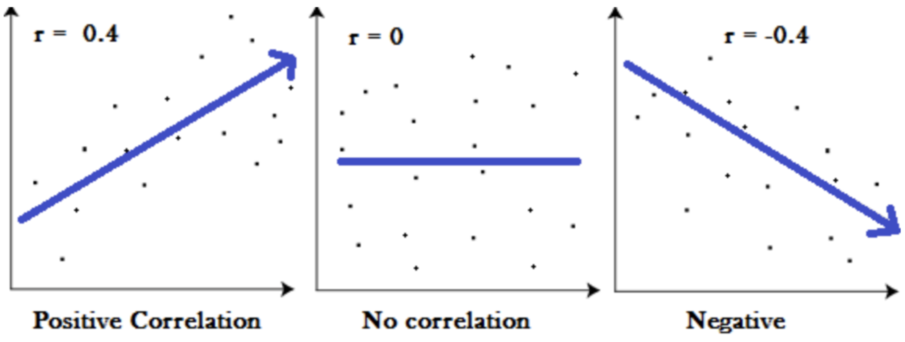


Figure 2.17: Correlation graph example.

2.3.3 Pearson Correlation Coefficient (PCC)

Correlation coefficients are usually aimed to discover how strong is a relationship between data. Frequently, formulas return values between -1 and 1 (see Fig. 2.17).

The Pearson Correlation Coefficient (PCC) shows the linear relationship between two sets of data being the ratio between the covarience of two variables and the product of their deviations.

The PCC ratio is given by:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{2.30}$$

where,

- r is the correlation coefficient
- x_i are the values of the x-variable
- $\bar{x}$ is the mean of the values associated with x-variable
- y_i are the values of the y-variable
- $\bar{y}$ is the mean of the values associated with y-variable

2.4 Related Works

In this section, it will be covered some related projects which are interesting to take a look at, in order to understand applications and methodologies used.

2.4.1 Identification and control for nonlinear systems using a self-evolving function-link interval type-2 fuzzy neural network

This research work says that, in recent years, many researchers have conducted studies regarding the combination of a fuzzy inference system and artificial neural networks, namely fuzzy neural networks(FNN)[28].

One crucial aspect in these FNN that can be difficult is selecting its network size. The authors developed a self-evolving function-link interval type-2 fuzzy neural network which builds a rule base autonomously.

The function-link is applied to an interval type-2 fuzzy neural network, in order to get a closer approximation to the function. The steepest descent gradient approach is used to establish the adaptive laws for the proposed system. The authors based themselves on a Lyapunov function technique to ensure the system is stable.

Finally, the system performance is verified using numerical simulations of nonlinear system identification and control of time-varying plants. Notice that, the proposed method performed above other methods. [28].

2.4.2 Temporal convolutional networks applied to energy-related time series forecasting

The presented work in [26], explored the use of TCNs to predict a time-series model in comparison with LSTM (Long Short Term Memory) models. They refer that many deep learning models have been introduced to interact with this kind of data but recurrent or convolutional network can learn and predict complex patterns over time-series. Temporal convolutional neural networks are specialized architectures which are able to understand various patterns using dilated convolutions and residual blocks, capturing longer-term dependencies while managing information loss.[26]

The results of this article showed that TCNs dilated causual convolutions outperforms the recurrent LSTM units. The results were conclusive on the performance of TCNs. Aspects to take in mind:

- Importance of the past observations input size window.
- The use of residual blocks the better handling of longer sequences, but LSTMs have shown more accuracy when using smaller windows.
- TCNs models have many trainable parameters which make them a bit costly, but they pay off in terms of performance.

2.4.3 Short-term traffic speed forecasting based on graph attention temporal convolutional networks

The importance of traffic prediction in real-time motivated Ge Guo and Wei Yuan to present an interesting combination of a graph attention network and a temporal convolutional network termed Graph Attention Temporal Neural Network (GATCN) to search for an answer to the issue in their paper[15].

This paper introduces an interesting DP architecture which will learn from the input data, all possible spatial-temporal features, process all information and output the traffic speed forecast.

The capability of combining two modules results in a better spatial-temporal correlation of data. "Comparisons using several benchmark models show the advantage of GATCN in capturing the spatiotemporal characteristics for traffic forecast."[15].

2.4.4 Tracking of Dynamical Processes with Model Switching Using Temporal Convolutional Networks

The paper[14] made in collaboration with the Department of Engineering and Design Western Washington University and the Department of ECE Worcester Polytechnic Institute reflects the problem of modeling and predicting a dynamical process with model switching. To address this, they developed a study of TCNs structural advantage in time-series sequence prediction problems [14].

With the application of TCNs to predict states of dynamical systems with model switching, they come to the conclusion that without any prior knowledge of the system model TCNs demonstrate to be at least as good as the classical algorithms. TCNs research is advancing at fast pace, and they are showing great results.

3 Systems Identification

3.1 Hybrid Systems

Many dynamical systems arrange in a combination of interactions which are common in continuous-time systems with typical interactions from discrete-time systems. For example, if we observe things such as an switched electric circuit, both voltage and current have continuous behaviors align with classical electrical laws but when the switches open or close it create an discontinuous change. If we take our thinking a little bit further, generally systems that combine both analog and digital components are a big group of examples. Also, in modern literature, control algorithms lead to both, continuous and discrete behaviors due to decision making encoded in the algorithm, logic or simply digital components used in their implementation[13].

3.1.1 Hybrid Automata

Imagine a hybrid system model divided into two parts:

- A discrete state s which makes reference to the configuration of the system,
- A continuous state ξ representing the continuous behaviour of the system.

Note that, the values that appear in s may depend on the system but for good comprehension, let assume, for example, LED control system, this system have an "ON"configuration and "OFF"configuration which is our s while our ξ the amount of light produced. Generally, we can say that continuous state usually changes during flows however they also can happen during jumps. While discrete state only changes during jumps. These kind of system have been named as hybrid automata, differential automata or commonly hybrid systems[13].

There few key data points to an hybrid automata:

- the set of configurations S,
- the domain map, domain: $S \rightrightarrows \mathbb{R}^n$, for each $s \in S$, the set domain(q) where the continuous state ξ will express its behavior,

- the flow map, $f : S \times \mathbb{R}^n \to \mathbb{R}^n$, expressing the continuous behavior of ξ through differential equations,
- the set of edges, $Edg \subset S \times S$, which corresponds to the set of pairs (s,s') where a transitions from configuration s to configuration s' is possible,
- the guard map, $GM(s,s') \rightrightarrows \mathbb{R}^n$, which corresponds for each set of pairs $(s,s') \in Edg$ to the continuous state ξ it must belong for the configuration shift from s to s' to occur.
- the reset map, $Reset : Edg \times \mathbb{R}^n \to \mathbb{R}^n$, which corresponds for each set of pairs $(s,s') \in Edg$ to the value set in the continuous state $\xi \in \mathbb{R}^n$ during a configuration shift. If the continuous state stays constant after a jump from s to s', it is assumed that the reset map, $Reset$, can be considered to be the identity, $Reset(s,s',\cdot)$.

Lets formulate a hybrid automata as an hybrid system with clear configurations. Therefore, for every $s \in S$, we will have:

$$C_s = \text{Domain}(s) \tag{3.1}$$

$$D_s = \bigcup_{(s,s')\in \text{ Edg}} \text{GM}(s,s') \tag{3.2}$$

$$F_s(\xi) = f(s,\xi), \quad \text{for all } \xi \in C_s \tag{3.3}$$

$$G_s(\xi) = \bigcup_{\{s':\xi\in GM(s,s')\}} (\text{Reset}(s,s',\xi),s'), \text{ for all } \xi \in D_s \tag{3.4}$$

Whenever ξ is part of two different guard maps, $GM(s,s')$ and $GM(s,s'')$, $G_s(\xi)$ has a minimum of two points, therefore G_s is not null[13]. With 3.1, 3.2, 3.3 and 3.4, we can define a hybrid system with state $(\xi,s) \in \mathbb{R}^n \times \mathbb{R}$ by:

$$\dot{\xi} = F_s(\xi), \quad s \in S, \xi \in C_s \tag{3.5}$$

$$(\xi^+,s^+) \in G_s(\xi), \quad s \in S, \xi \in D_s \tag{3.6}$$

3.1.2 Switching Systems

A switching system is within a group of hybrid systems in which combine continuous and discrete dynamic systems. Examples include applications such as manufacturing control systems [38], communication networks and car control systems and aircraft control[2]. To be able to model these systems we cannot focus only on continuous methods or discrete methods, their behavior require a more open approach.

"A switching system is a differential equation whose right-hand side is chosen from a family of functions based on a switching signal[13]", which implies that every time there is a switching signal, the model will be defined as a time-varying differential equation. In the work bench of hybrid systems, info related to the switching signal is many times embedebed into data using timers and reset rules.

Let us define a switched systems as:

$$\dot{\xi} = f_s(\xi) \tag{3.7}$$

Where, for every $s \in S = \{1, 2, \ldots, s_{\max}\}$, $\quad f_s : \mathbb{R}^n \to \mathbb{R}^n$ will be the corresponding continuous function.

So a solution to 3.7 may consist in:

- $\xi : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, a locally continuous function,
- $s : \mathbb{R}_{\geq 0} \to S$, for each time step interval, this is a constant with a finite number of discontinuities which fulfill the requirements for $\dot{\xi}(t) = f_{s(t)}(\xi(t))$ for almost all $t \in \mathbb{R}_{\geq 0}$

Consider the following, for the next example, I is number of discontinuities in s, with the possibility of I may be infinite, $t_0 = 0$ and $\{t_i\}_{i=1}^{\prime}$ be the crescent sequence of times at which s is not continuous[13]. As an example, we present a solution (ξ, s) to 3.7 named dwell-time solution with dwell time characterized by (see Fig. 3.1):

$$\tau_D > 0 \text{ if } t_{i+1} - t_i \geq \tau_D \text{ for all } i = 1, 2, \ldots, I - 1 \tag{3.8}$$

Note that, τ_D refers to the minimum time that separates switchings. Every dwell-time answer to the problem can be created as part of the hybrid system:

$$x = (\xi, s, \tau) \in \mathbb{R}^{n+2} \tag{3.9}$$

provided through equations 3.10 and 3.11.

$$\left.\begin{aligned}\dot{\xi} &= f_s(\xi)\\ \dot{s} &= 0\\ \dot{\tau} &\in [0, 1/\tau_D]\end{aligned}\right\} =: F(x), x \in C := \mathbb{R}^n \times S \times [0,1] \tag{3.10}$$

$$\left.\begin{aligned}\xi^+ &= \xi\\ s^+ &\in S\\ \tau^+ &= 0\end{aligned}\right\} =: G(x), x \in D := \mathbb{R}^n \times S \times \{1\} \tag{3.11}$$

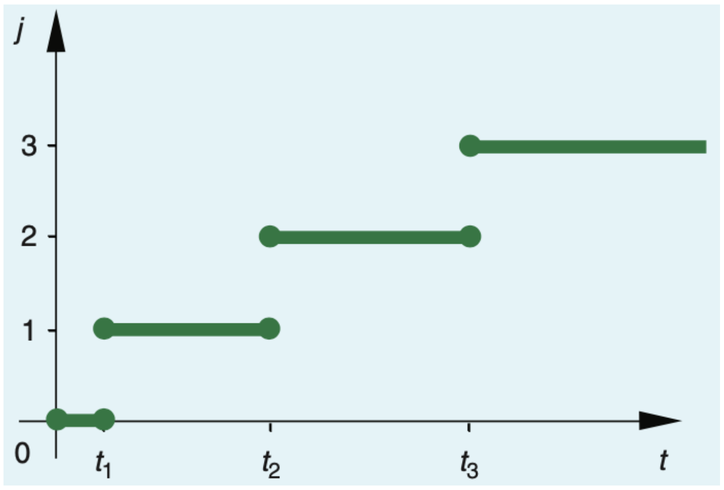


Figure 3.1: Hybrid time domain: Solution based on dwell-time wit a dwell constant τ_D

3.2 Hybrid Systems Identification

Modeling [16, 5], stability analysis [21, 5], control [5][31][4], verification , and fault detection [44] [11] have dominated the majority of the hybrid systems literature.

Hybrid systems emerge from the cooperation of continuous and discontinuous processes. A system like this produces a combination of continuous and discrete signals, with values in a continuum (like the real numbers $\mathbb{R}$ and a finite set (like a, b, c), respectively. Therefore, a hybrid dynamical system is one where the behavior is influenced by integrating continuous and discrete dynamics. Hybrid dynamical systems produce variables or signals, which are created from continuous or discrete values, time signals, with other systems and the environment through them. Further to that, independent variables, which might be continuous or discrete, may be dependent on these continuous or discrete-valued signals. Another point to emphasis is that some of the signals could be time-driven, while others could be asynchronously event-driven.

When portions of a model are unknown, however, identification requires a parameterization of the uncertainty. The universal function approximator property of neural networks[17], has made them ubiquitous[10].

There are various advantages to identifying continuous time system dynamics over discretized alternatives. Physical systems' dynamics are frequently influenced by continuous dynamics, making modeling and identification in continuous time natural. Not only does this mean that continuous model parameters are closely related to real physical features, but it also means that inherent structure, such as sparsity, may be captured in a continuous time model, but a discretized version would lose it [10].

In many circumstances, discrete events alter the dynamics, necessitating the use of hybrid models. In many real-world situations, for example, open loop identification is impossible, and a discrete time controller must be included in the model[10].

Let us start by defining the equation which represents a discrete time switching system:

$$x(k+1) = f_{\sigma(k)}(x(k), u(k)), \quad k \in \mathbb{N} \tag{3.12}$$

$$x(0) = \xi \tag{3.13}$$

Notice that $x(k) \in \mathbb{R}^n, n$ is a positive integer; $u(k) \in \mathbb{R}^m$ is the input and m is also a positive integer. The function σ from $\mathbb{N}$ to S, where $S = \{1, 2, \ldots, p\}$ is finite set and p is also a positive integer. If any $j \in S$, f_j: $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a locally Lipschitz function, complying with $f_j(0,0) = 0; \xi \in \mathbb{R}^n$[37].

Lets consider a finite group of pairs $(i, j) \in S \times S$, $E(S)$ to be able to switch between a system f_i and another system f_j [25]:

1. The group of indexes in S corresponding to the set of vertices.
2. A set of edges $E(S)$ containing on one hand, a directed edge (i, j) When it is possible to switch between a vertex (system) i and other vertex (system) j such that $i, j \in S, i \neq j$ and on other hand a self-loop (j, j) at vertex (system) j when it stays on vertex (system) j for range from two to ∞ consecutive time steps.

Said that, the set of sequences are described by:

$$\mathcal{M}_S = \{\sigma : \mathbb{N} \to S \mid (\sigma(k), \sigma(k+1)) \in E(S), \forall k \in \mathbb{N}\} \tag{3.14}$$

$$\mathcal{M}_u = \{u : \mathbb{N} \to \mathbb{R}^m\} \tag{3.15}$$

$$\mathcal{M}_B = \{v : \mathbb{N} \to B_1^m\} \tag{3.16}$$

It is also important to mention a constraint to any switching signal $\sigma \in \mathcal{M}_S$ in order to comply with the provided switched digraph. We defined $x(k, \xi, \sigma, u), k \in \mathbb{N}$ as the solution to 3.12 associated with the initial condition ξ, the switching signal σ and the corresponding input signal u[20].

3.3 Proposed approach

On this thesis, the main goal is to implement a temporal convolutional neural network architecture capable of capturing the dynamic of a nonlinear system, with structural reconfiguration.

This architecture will have two principal objectives:

1. Capture the spatio-temporal nonlinear system dynamics.
2. Reflect the system configuration structure associated with the dynamical changes.

As there aren't any scientific rules with a good support theory to design an ideal deep neural network structures, it was necessary to search for proper network scheme that suits the objectives[51].

The design of a new deep neural network structure is never trivial, being commonly oriented by the required performance and computational burden [51]. As there is no design support framework, to the best of the author knowledge, to help researchers not only tuning hyperparameters but also in layers topology selection, namely convolution layers and pooling layers, and the underlying activation functions and dropout mechanisms, the synthesis approach is usually carried out under a trial-and-error heuristics or meta-heuristics [24].

In this work several candidate topologies were first analysed and the most fitted architecture in terms of generalization capability and lower computational complexity was the one to be chosen. This topology is schematically presented in Fig. 3.2. The first part of the structure consists of a 1D convolutional layer, followed by a dropout layer, and by a three cascaded 1D convolutional layers, a max polling 1D layer and two dense layers. With respect to the input to the CNN neural network, it consists of two types of information, namely the structural configuration and regressors, which comprise previous inputs to the plant and outputs. As the CNN input includes tapped delay lines based on the system inputs and outputs, the proposed architecture will be denote, in the following, as temporal convolutional neural network (TCN).

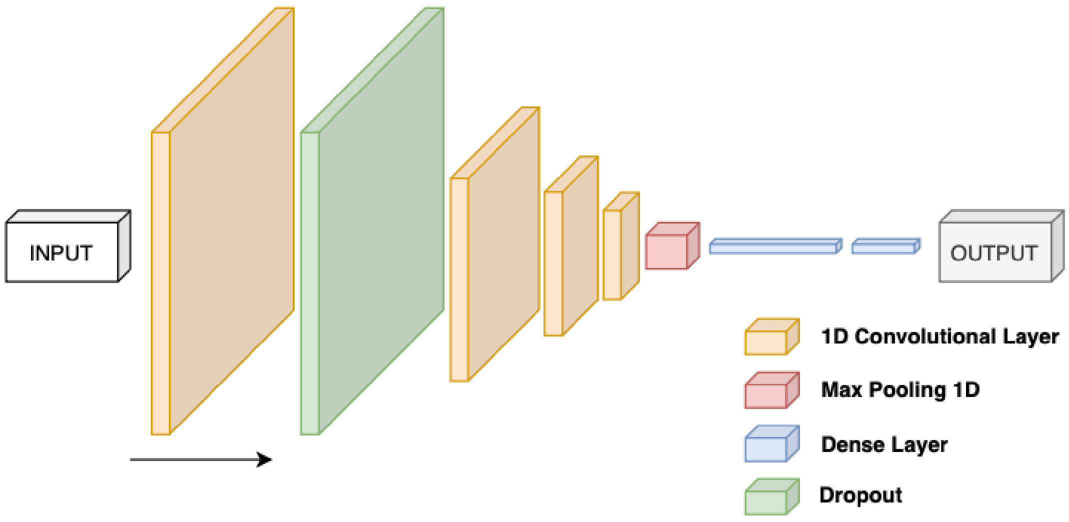


Figure 3.2: Proposed architecture

4 Case Study

4.1 Three-Tank System

The case study experiment consists of a three-tank system benchmark with the main goal of modeling the nonlinear dynamics in tanks using the proposed architecture.

The nonlinear controlled system AMIRA© DTS 200 three-tank system benchmark (see Fig. 4.1) consists in three plexiglas cylindrical tanks with identical cross-section supplied with distilled water. The water levels, respectively h_1, h_2 and h_3 are measured by piezoresistive transducers. The middle tank T_3 is connected to the other two tanks by means of circular cross-section pipes provided with manually adjustable ball valves. The connecting pipes and the tanks are additionally equipped with manually adjustable valves and outlets for the purpose of simulating clogs as well as leaks. The main outlet of the system is located in the tank T_2, which is directly connected to the collecting reservoir by means of a circular cross-section pipe provided with an outflow ball valve. Additionally, this system includes two pumps u_1 and u_2 for feeding tanks T_1 and T_2 with water.

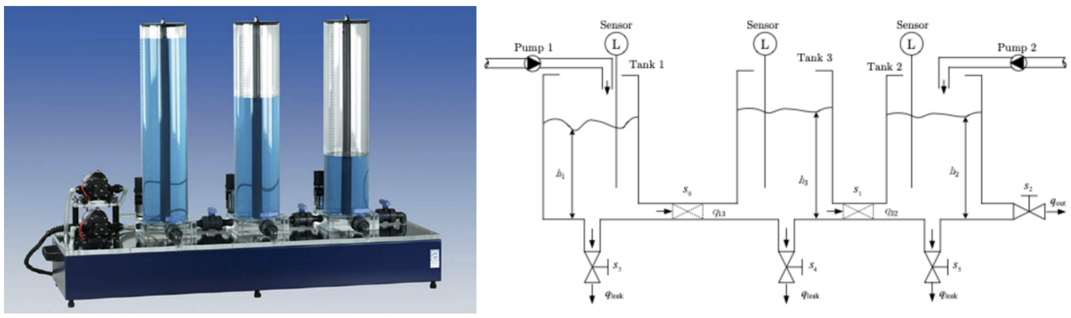


Figure 4.1: Three-Tank System AMIRA DTS200 and respective schematic.

The nonlinear dynamics of the three-tank system can be described by following equations:

$$\begin{aligned}
\frac{dh1}{dt} &= \frac{1}{At}\Big[q1 - \xi 1,3Spsgn(h1-h3)\sqrt{2g|h1-h_3|} \\
&\qquad -\xi 1,0Sp\sqrt{2gh_1}\Big] \\
\frac{dh2}{dt} &= \frac{1}{At}\Big[q2 + \xi 3,2Spsgn(h3-h2)\sqrt{2g|h3-h_2|} \\
&\qquad -\xi 2,0Sp\sqrt{2gh_2}\Big] \\
\frac{dh3}{dt} &= \frac{1}{At}\Big[\xi 1,3Spsgn(h1-h3)\sqrt{2g|h1-h3|} \\
&\qquad -\xi 3,2Spsgn(h3-h2)\sqrt{2g|h3-h2|} \\
&\qquad -\xi 3,0Sp\sqrt{2gh_3}\Big]
\end{aligned} \tag{4.1}$$

where:

- Tank levels are noted as h_i, $i = 1,2,3$,
- Cross section of each tank as A_t,
- The cross section interconnection pipes , S_p,
- q_j, $j = 1,2$ is the flow pump rate,
- The acceleration of gravity g,

Note that sgn stands for "sign"and $|\cdot|$ the absolute value notation.

4.2 Data Preparation

One important step in modeling the dynamics of a system is the way we collect data. As explained in the previous section, the system has two inputs: the first pump *u1* and the second pump *u2* so to collect our dataset we will send input signals to generate some behaviour in the tank system.

To generate the values for the pumps, $u1andu2$, which means the input signal of the system, it was chosen a random number between 1 and 5 (minimum and maximum values for each pumps).

This signal will be sent to the system until a maximum water level is reached or if the system becomes unchanged for at least two minutes. After one of these situation, the system will jump to another set of pump values. In case of the first one, the system will wait until water levels are at least at half the maximum capacity.

For the Temporal Convolutional Neural Network (TCN) training and test data, the structural configuration has made after 25 different pump values sets.

For the MLP training and test data, a dataset was collected per configuration in order to train each MLP separately.

Regarding the outputs of the system, it was saved the water level for each tank *h1*, *h2*, *h3*.

Summing up, the datasets that we extract from the system to be able to train, validate and test follow the structure in 4.2, for each time sample k:

$$\begin{bmatrix} k & u1 & u2 & h1 & h2 & h3 \end{bmatrix} \tag{4.2}$$

4.3 TCN based Identification

For training the temporal convolutional neural network, it was collected a dataset from the three-tank system with 20.000 samples, going thought two different structural configurations two times each.

The structural configuration of the valves was incorporated in a vector S containing six values (0 or 1) associated with a state of the six valves, in which zero means closed and one means the corresponding value is open.

$$S_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

$$S_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.4}$$

Concerning the other data collected from the system, which means the pump values and the sensor data regarding the water level, there were normalized between 0 and 1.

In order to capture the three-tank system dynamics, a dataset collected from this system was first obtained by feeding the 2 pumps with a sequence of inputs $\{u_1, u_2\}$ as shown in Fig. 4.2, and sampling the levels of the 3 tanks, $\{h_1, h_2, h_3\}$, as presented in Fig. 4.3, while using a sampling period of 1 s. In the course of the excitation of the system its structural configuration, defined by a binary array $S[1:6], X[i] = \{0, 1\}$, and dictated by the status of the valves, is alternated every 5000 samples between configuration 1, defined by S_1 and configuration 2, defined by S_2.

Besides the input S provided to the TCN, five additional vectors were considered as inputs, consisting of fifth order tapped delay lines associated to $u_i, i = 1, 2$ and $h_j, j = 1, 2, 3$. This data structure is schematically presented in Fig. 4.4.

In this supervised training, we will have an input shaped as (31,1) to an output of the 3 tank water levels. After this preparation of data, it is fitted into the model. To tune the neural network, it was conducted some grid searches to look for better hyperparameters such as number of epochs, the learning rate or batch size. For the optimizer it was made some tests using Adam, NAdam and RMSProp, the one that achieved the best results during training was the NAdam.

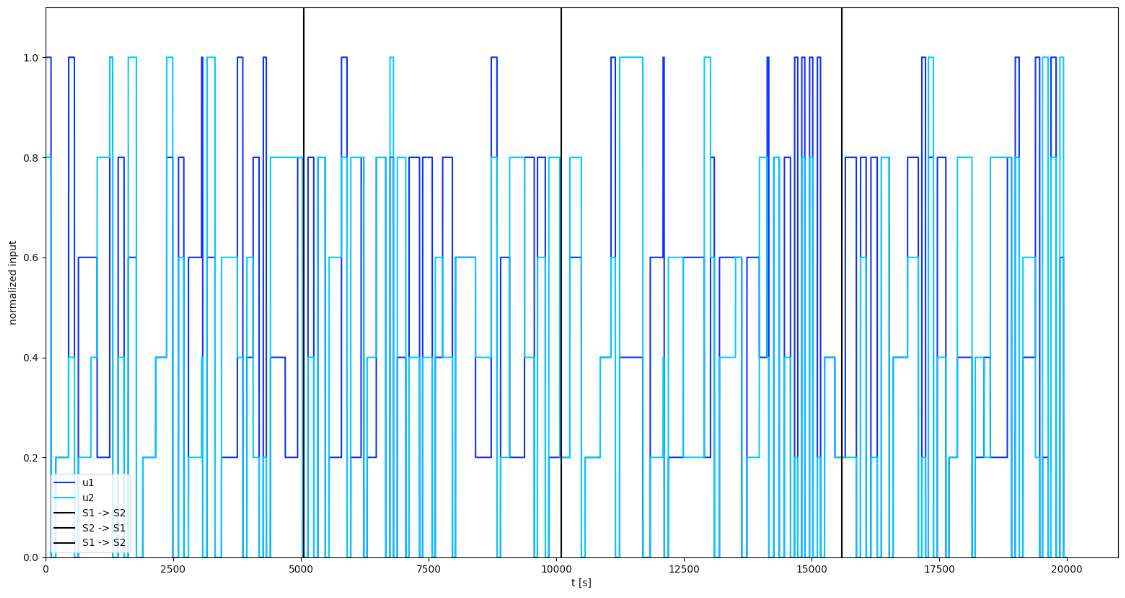


Figure 4.2: Train Data for TCN: Input Data

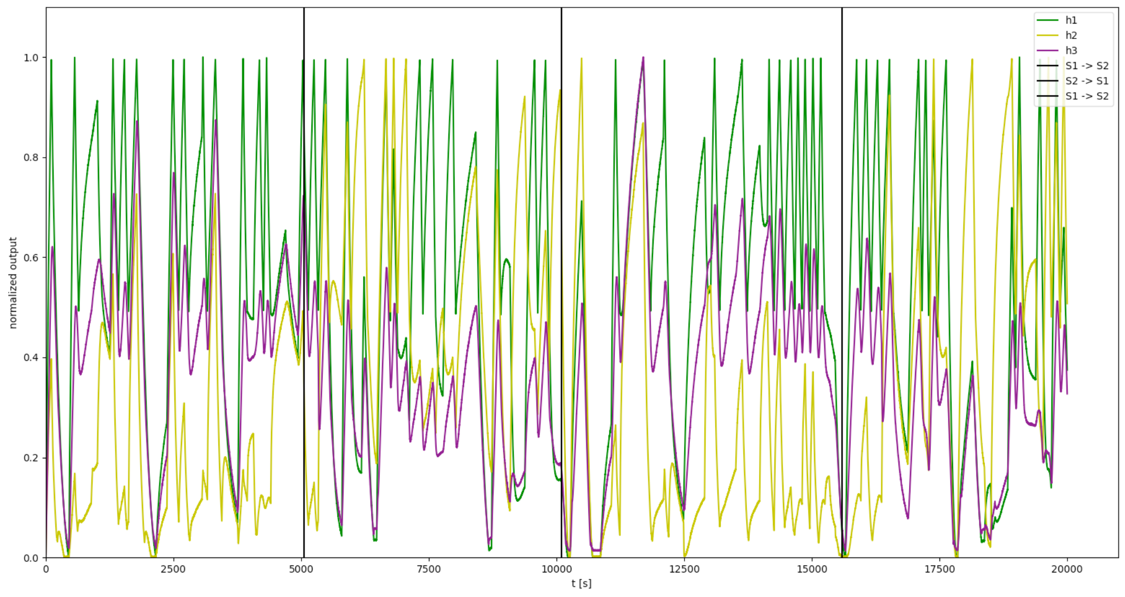


Figure 4.3: Train Data for TCN: Output Data

One important step in assessing the quality of the training process is the validation process that consists in a prediction with a new dataset. For this process we will use the signals shown in Fig. 4.5 and Fig. 4.6 as well as the structural configuration as the input to the TCN and compare it with the real outputs.

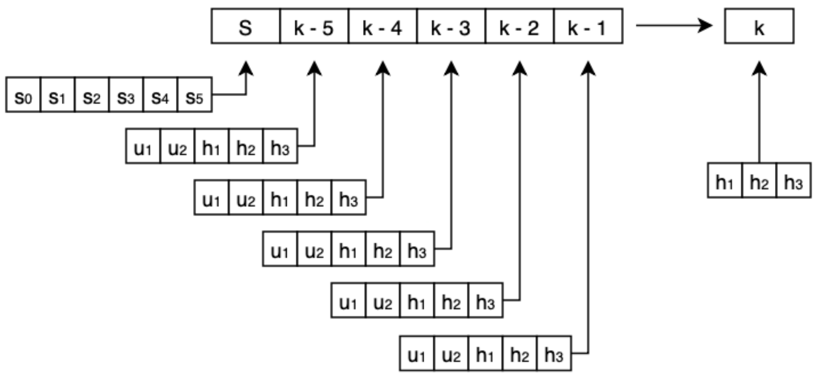


Figure 4.4: Input and Output structure for TCN train

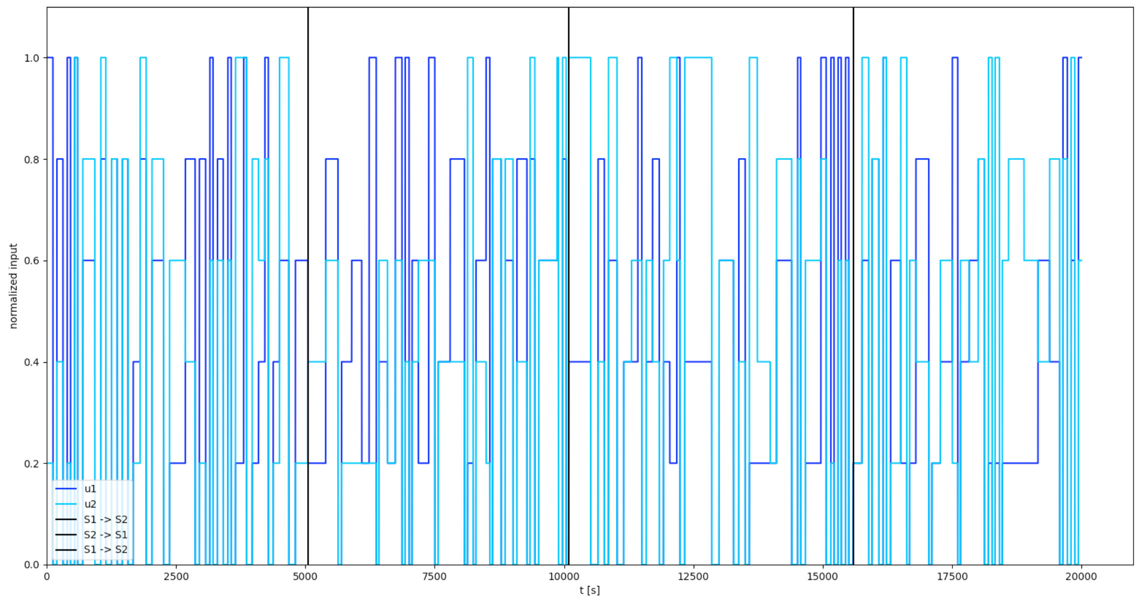


Figure 4.5: Validation Data for TCN: Input Data

We present the TCN predictions made for the water level in the tank 1, tank 2 and tank 3:

As we can observe in Fig.4.7, Fig.4.8 and Fig.4.9, our convolutional architecture follows the real values closely as we can conclude through the elaboration of table4.1.

Model	Metric	h_1	h_2	h_3	Average
TCN	MSE	4.70311×10^{-5}	4.25167×10^{-5}	5.15767×10^{-5}	4.70415×10^{-5}
	MAE	4.73700×10^{-3}	4.64720×10^{-3}	4.49151×10^{-3}	4.62524×10^{-3}
	PCC	9.99721×10^{-1}	9.99693×10^{-1}	9.99509×10^{-1}	9.99641×10^{-1}

Table 4.1: Performance metrics for TCN Validation.

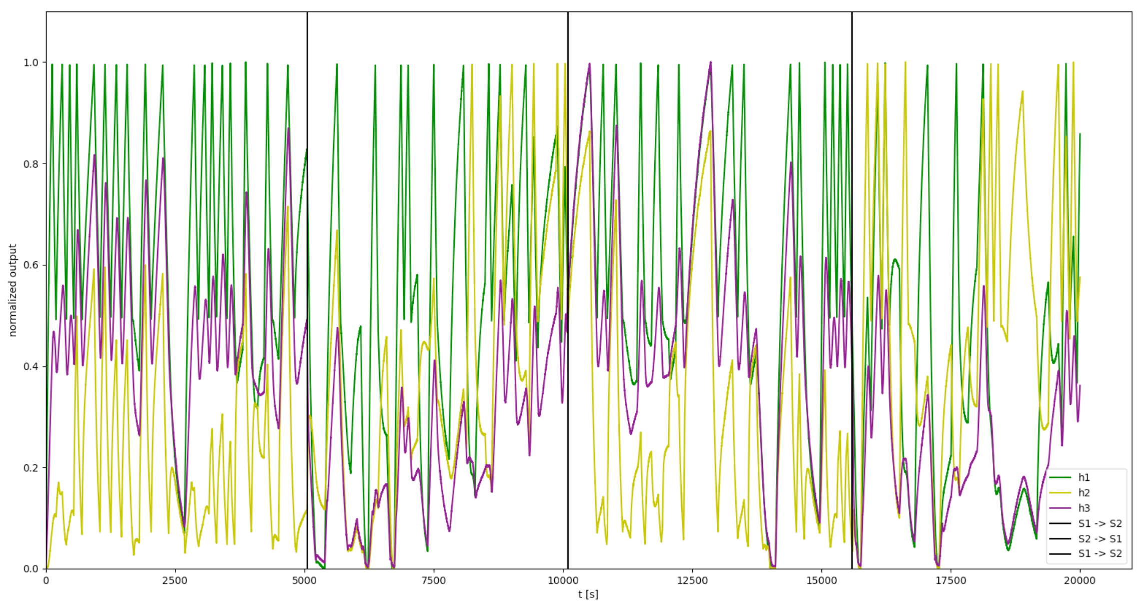


Figure 4.6: Validation Data for TCN: Output Data

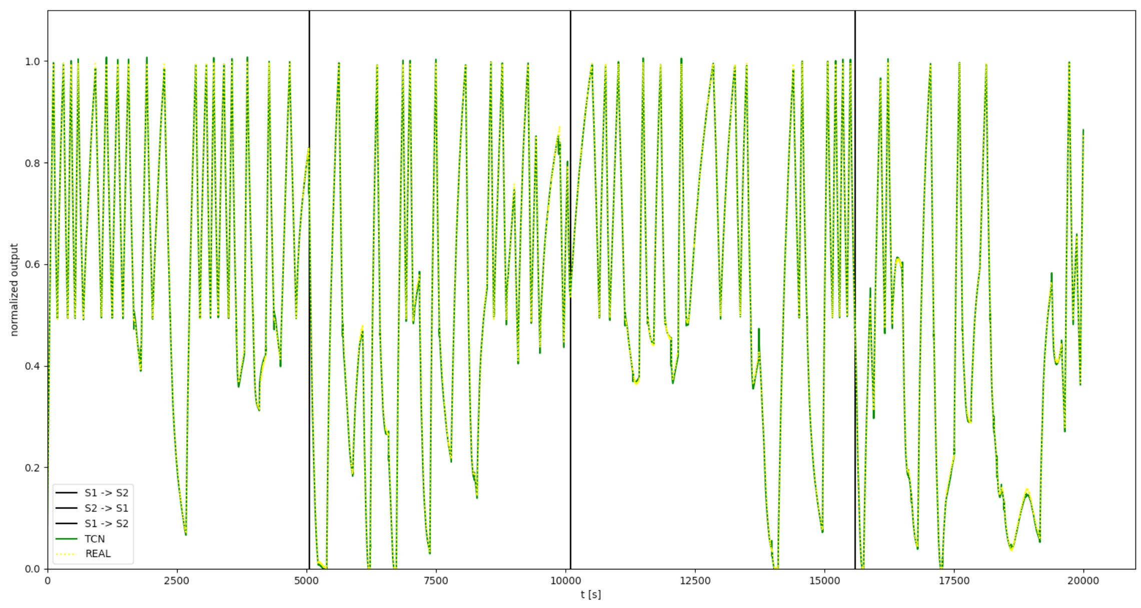


Figure 4.7: TCN predicts for tank 1

4.4 MLP based Identification

Since the dataset collected for TCN training is not suitable to be used in the train of MLP networks, as it reflects the effect of structural switching, two new datasets, concerning configurations S_1 and S_2 were collected, as presented in Fig. 4.10, Fig. 4.11, Fig. 4.12 and Fig. 4.13.

The two MLPs, each associated with one of the configurations, include as inputs a fifth order tapped delay lines formed with $u_i, i = 1, 2$ and $h_j, j = 1, 2, 3$ (see Fig. 4.14), but

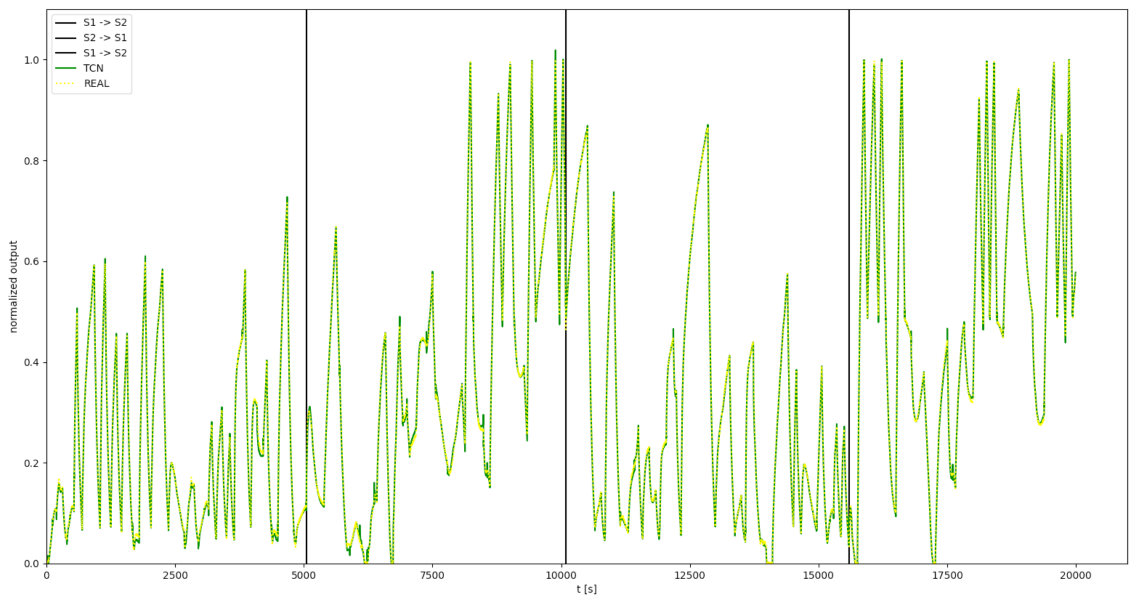


Figure 4.8: TCN predicts for tank 2

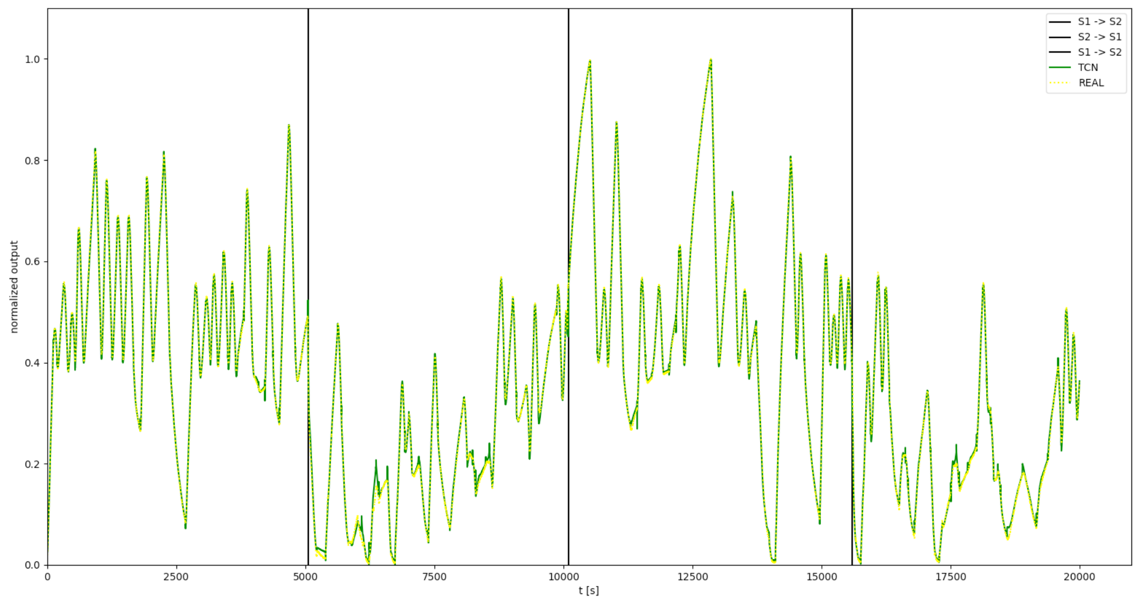


Figure 4.9: TCN predicts for tank 3

unlike the TCN topology, there is no configuration selection. In this case it is externally provided by choosing the appropriate MLP. As such, the MLPs concerning each one of the configurations, S_1 and S_2, are trained separately using the corresponding datasets.

With respect to the internal layers, both MLPs include 2 layers, comprising 20 neurons, in the first layer and 10 in the second one, with hyperbolic tangent functions. The output layer contains 3 neurons, corresponding to the number of the tank levels, and presenting RELU activation functions. In the training stage the Adam algorithm was used as optimiser.

So for the training of multilayer perceptron neural networks, it was present two datasets with 10.000 samples, one for each structural configuration, presented in Fig. 4.10, Fig. 4.11, Fig. 4.12 and Fig. 4.13.

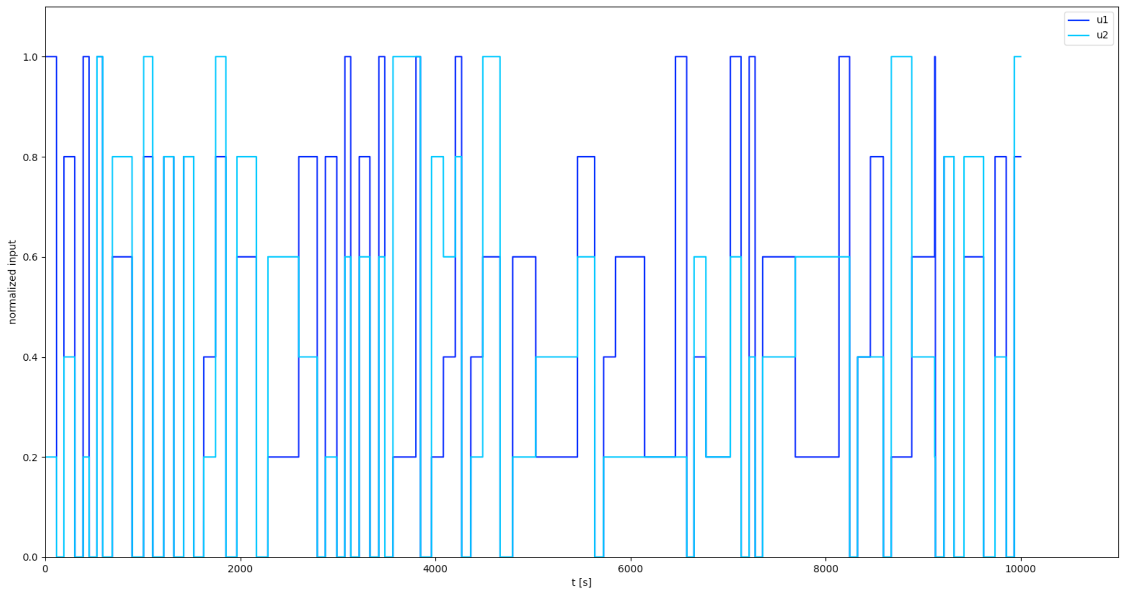


Figure 4.10: Train Data for MLP1: Input Data

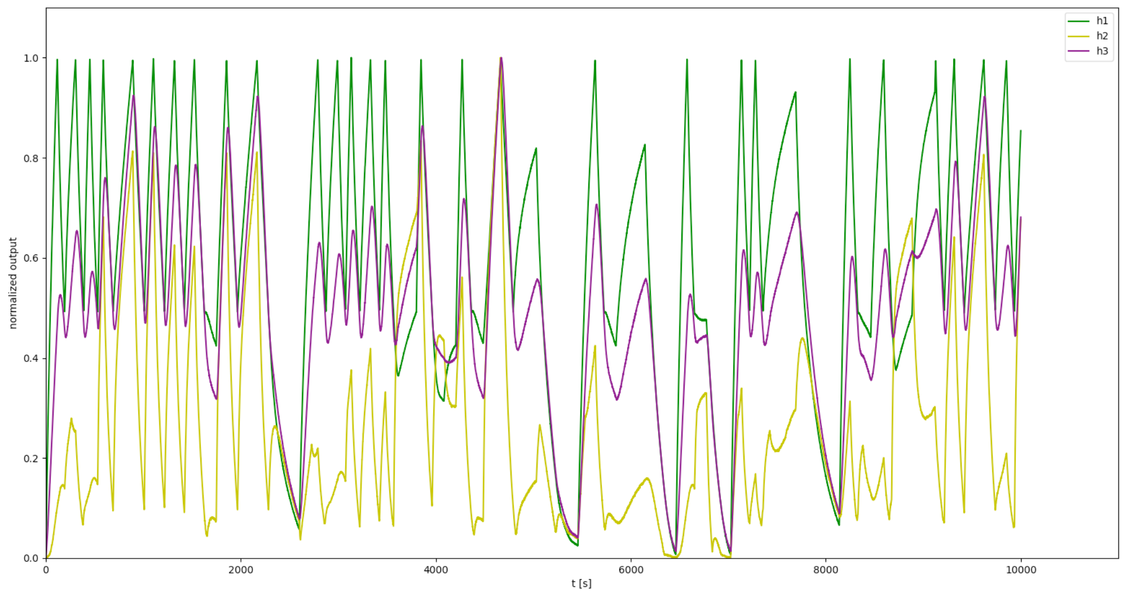


Figure 4.11: Train Data for MLP1: Output Data

Both of these MLPs will have a similar input and output format to the TCN, except the structural configuration vector S. Therefore, it leaves us with a 25 inputs if we consider a regressor containing 5 past observations.

In order to prepare all the data as the input for both MLPs as well as incorporate into

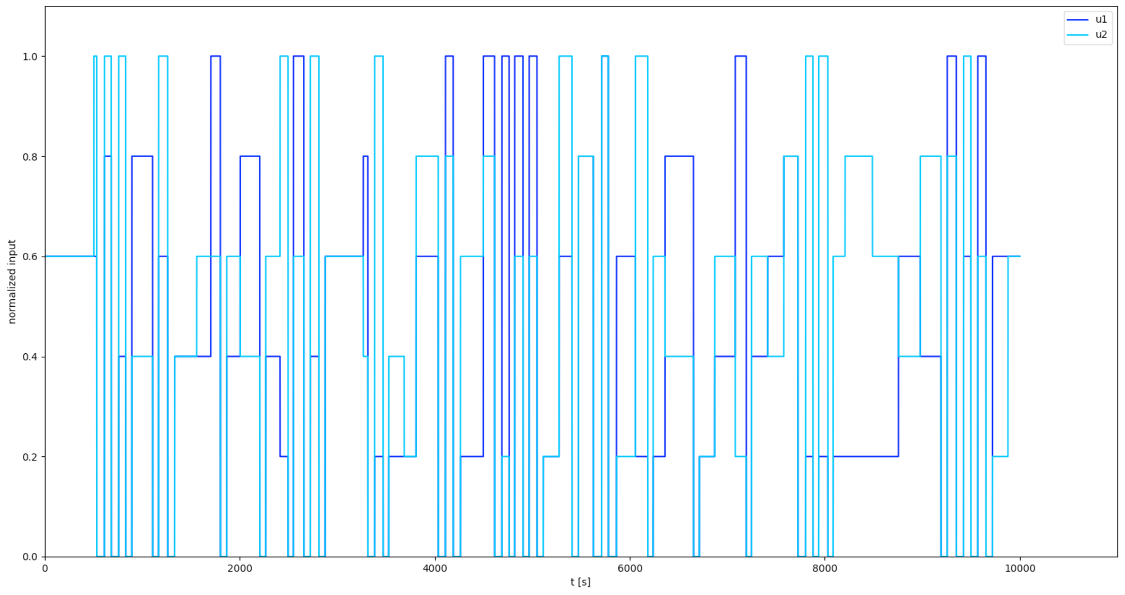


Figure 4.12: Train Data for MLP2: Input Data

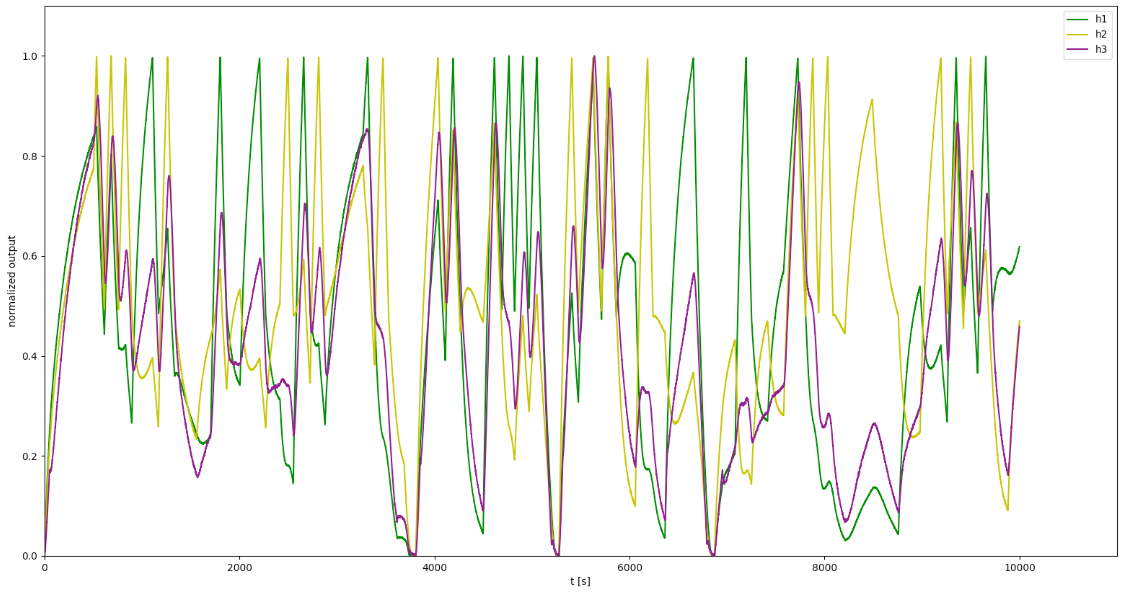


Figure 4.13: Train Data for MLP2: Output Data

the data the temporal information, it was used a regressor of k (in this case, it was k = 5) past observations to predict the level of water in the tanks. Which means that the input of the neural network will have 5 entries per each k past observations. So, we end up with a twenty-five entries as the network input and three as the output at the time k.

For the validation of MLP networks, to reflect the effect of structural switching, two new datasets, concerning configurations S_1 and S_2 were collected, as presented in Fig. 4.15, Fig. 4.16, Fig. 4.17 and Fig. 4.18.

Here, we present the MLP1 predictions made from the respective validation dataset

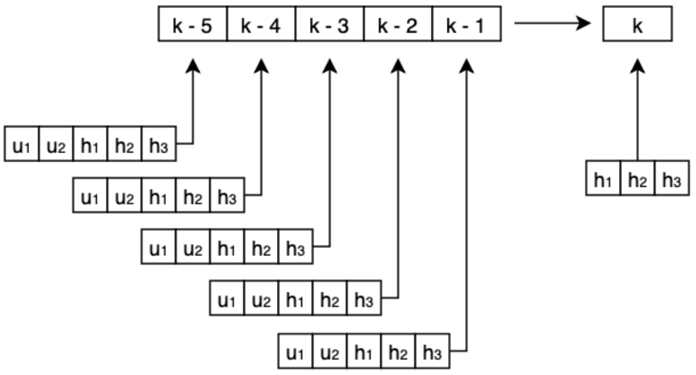


Figure 4.14: Input and Output structure for MLP train

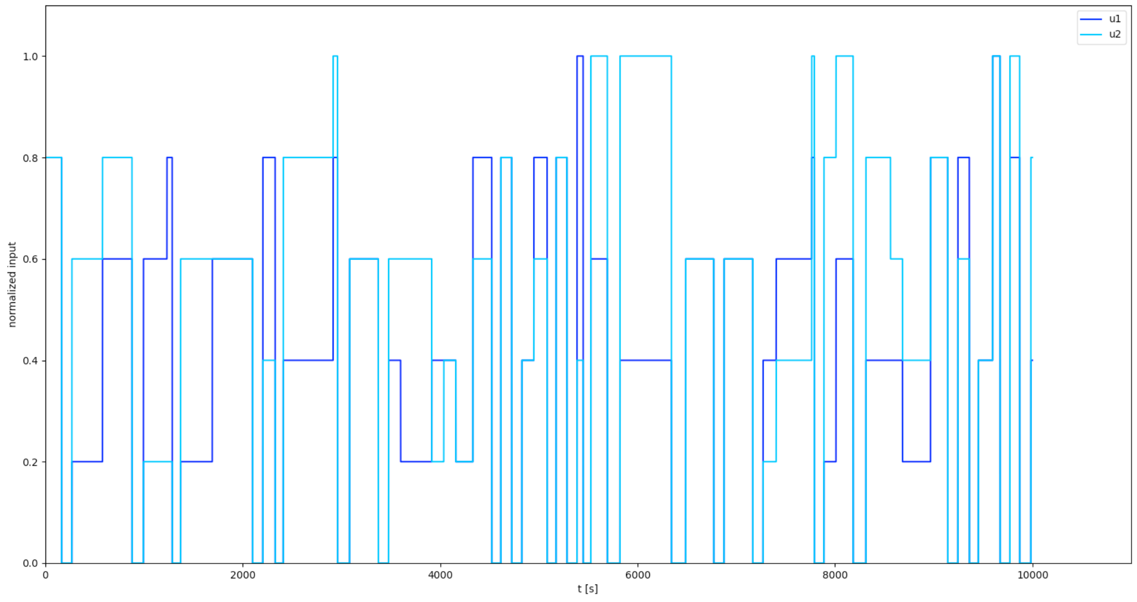


Figure 4.15: Validation Data for MLP1: Input Data

Model	Metric	h_1	h_2	h_3	Average
MLP1	MSE	3.44120×10^{-5}	7.30357×10^{-6}	9.3354×10^{-6}	1.70170×10^{-5}
	MAE	4.56637×10^{-3}	1.84189×10^{-3}	2.35174×10^{-3}	2.92000×10^{-3}
	PCC	9.99911×10^{-1}	9.99918×10^{-1}	999913×10^{-1}	9.99914×10^{-1}

Table 4.2: Performance metrics for MLP1.

for the water level in the tank 1, tank 2 and tank 3:

In order to quantify the error of the MLP1 network, three metrics have been considered, namely, the mean squared error (MSE), mean absolute error (MAE) and Pearson's correlation coefficient (PCC). As it can be observed from Table 4.2, the identification framework based on a multilayer perceptron network show a good approximation for configuration 1.

Here, we present the MLP2 predictions made from the respective validation dataset for the water level in the tank 1, tank 2 and tank 3:

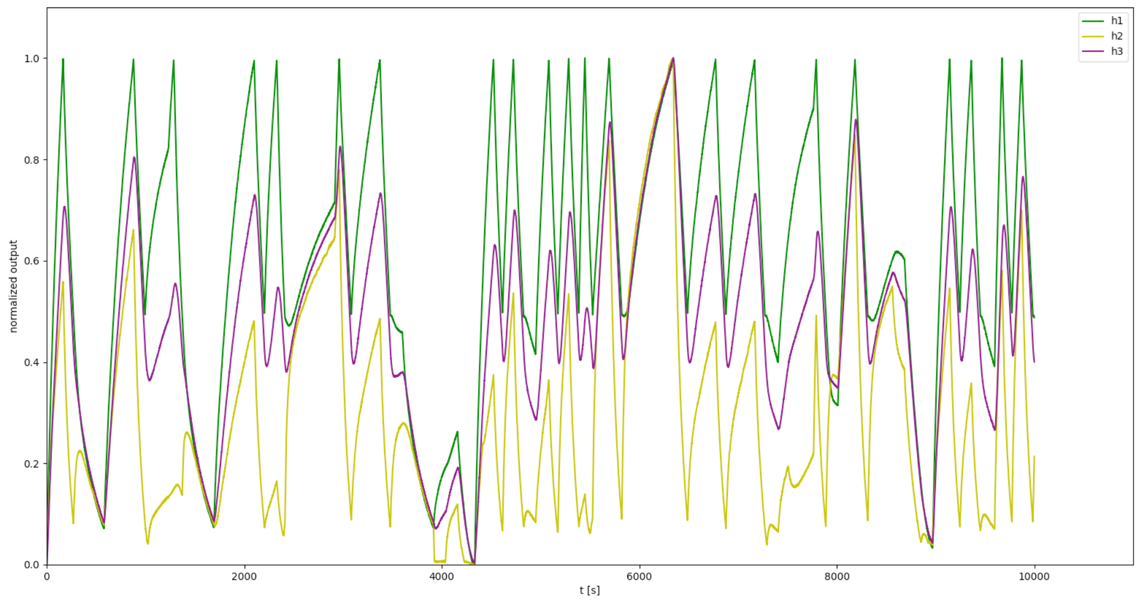


Figure 4.16: Validation Data for MLP1: Output Data

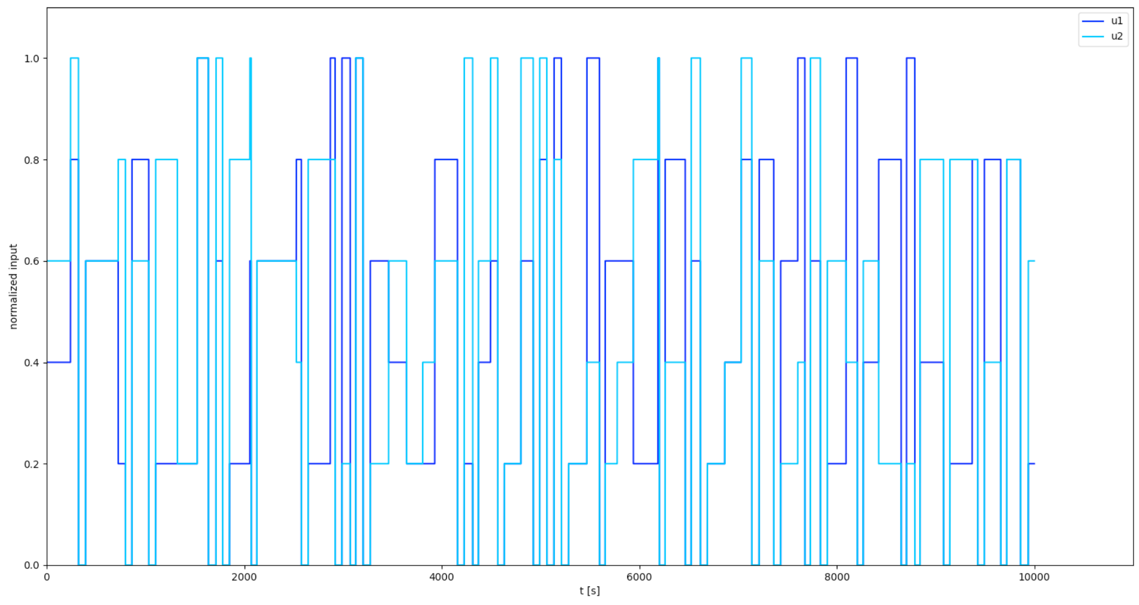


Figure 4.17: Validation Data for MLP2: Input Data

In order to quantify the error of the MLP2 network, three metrics have been considered, namely, the mean squared error (MSE), mean absolute error (MAE) and Pearson's correlation coefficient (PCC). As it can be observed from Table 4.3, the identification framework based on a multilayer perceptron network show a good approximation for configuration 2.

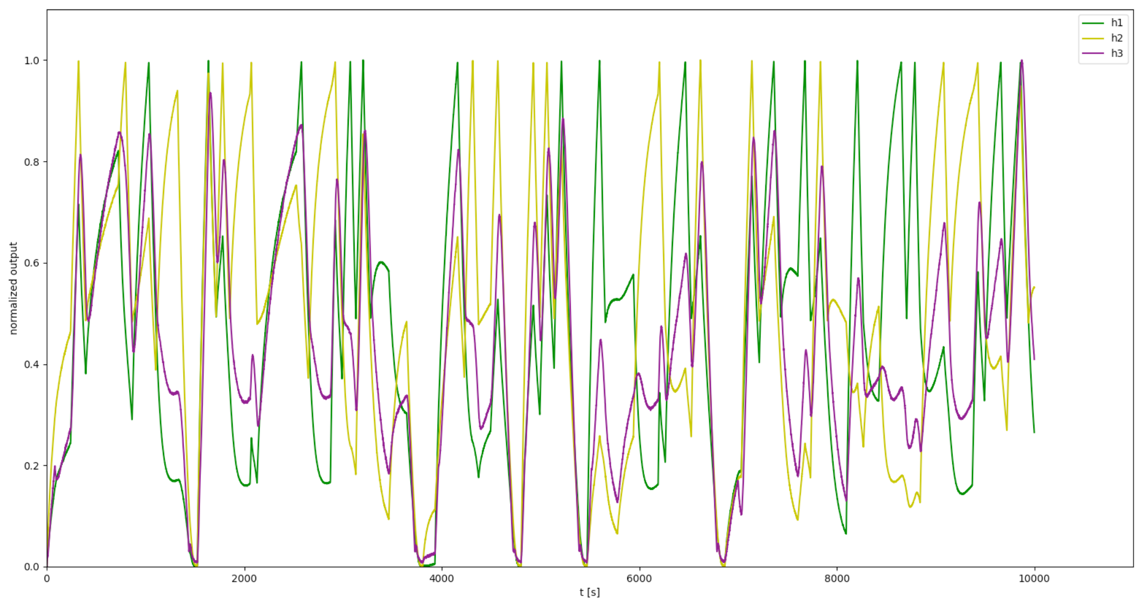


Figure 4.18: Validation Data for MLP2: Output Data

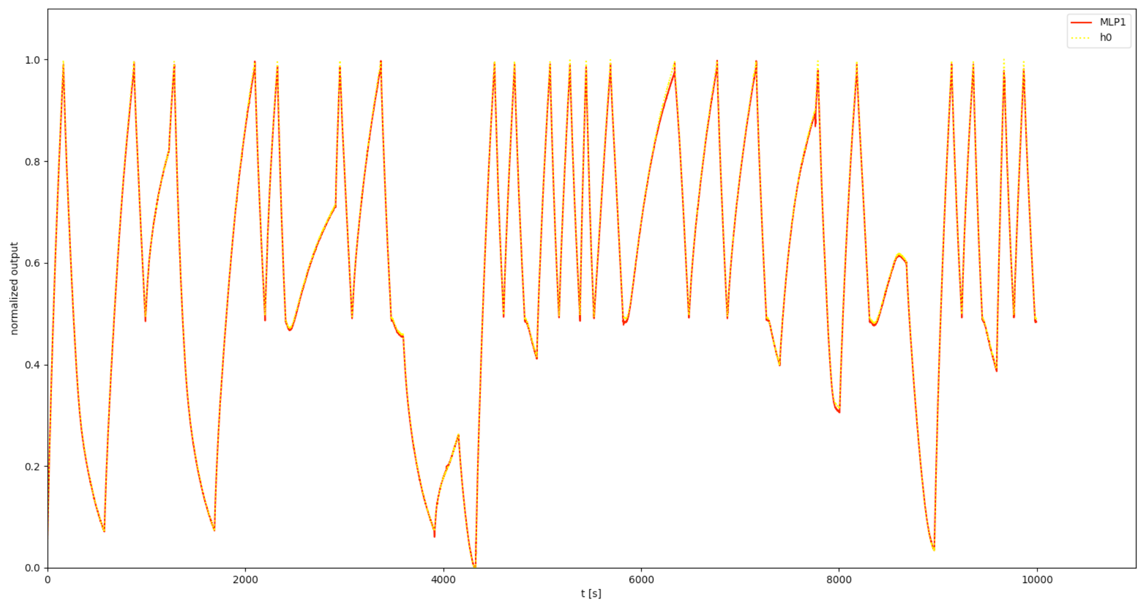


Figure 4.19: MLP2 predicts for tank 1

Model	Metric	h_1	h_2	h_3	Average
MLP2	MSE	7.87857×10^{-6}	8.61725×10^{-6}	1.35906×10^{-5}	1.00288×10^{-5}
	MAE	2.02687×10^{-3}	1.99963×10^{-3}	2.74633×10^{-3}	2.25762×10^{-3}
	PCC	9.99944×10^{-1}	9.99946×10^{-1}	9.99887×10^{-1}	9.99926×10^{-1}

Table 4.3: Performance metrics for MLP2.

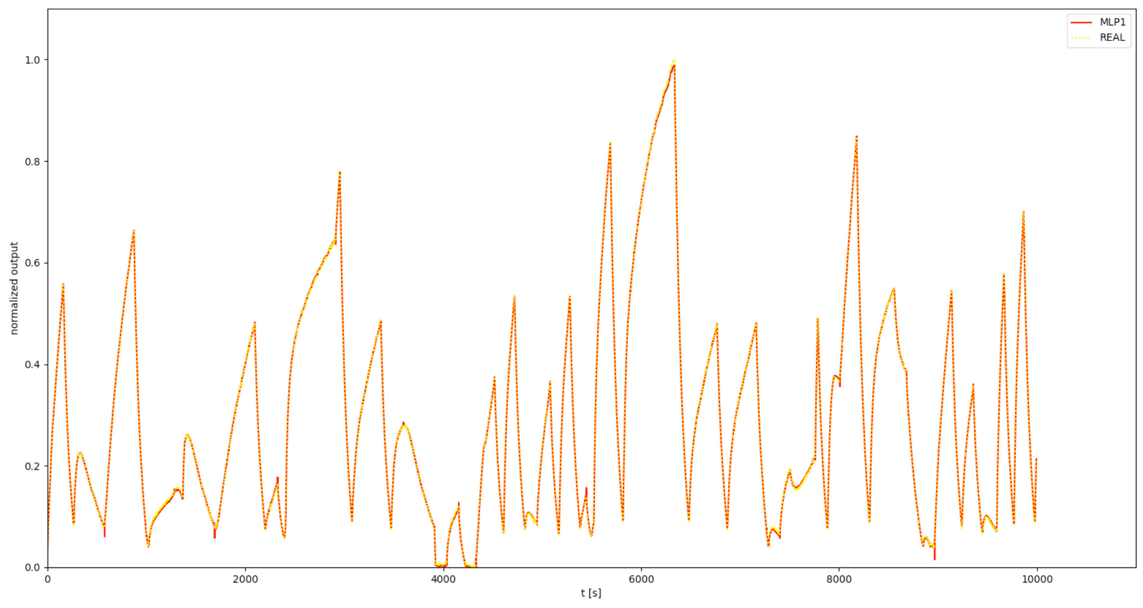


Figure 4.20: MLP2 predicts for tank 2

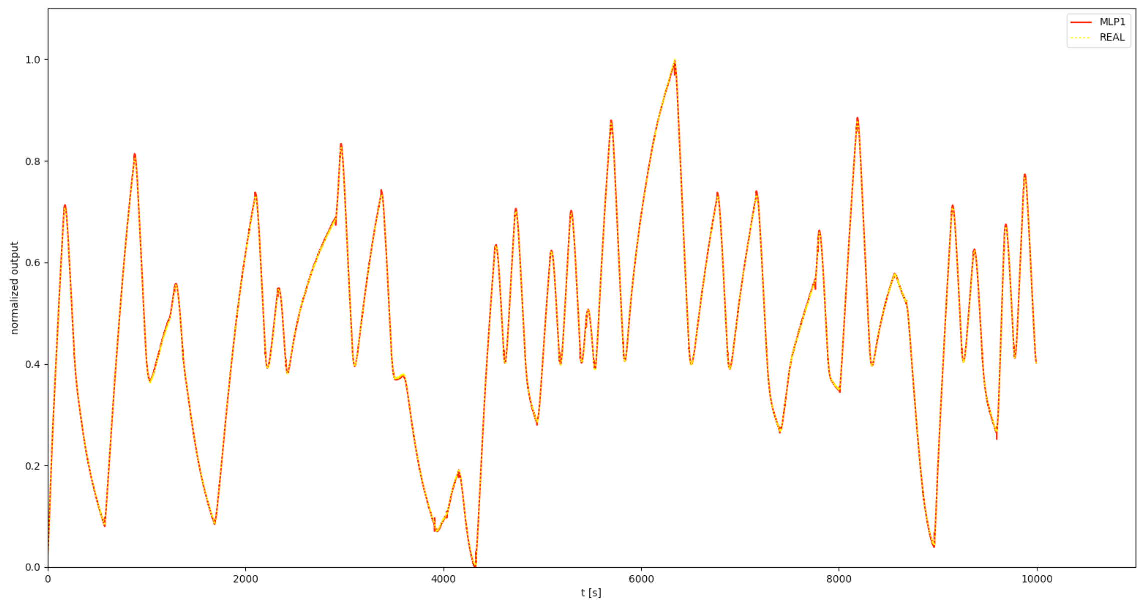


Figure 4.21: MLP2 predicts for tank 3

4.5 Discussion

In this section, we present the prediction experiments that allows to truly find out if our proposed architecture can manage to predict the system dynamic comparing to the MLP architecture.

After the training and validation stages have been completed, the TCN and the MLP-based frameworks were tested on a new dataset consisting of a data quadruplet $Z = Z\{k, S, u_1, u_2, h_1, h_2, h_3\}$. The time response of the two frameworks is presented in

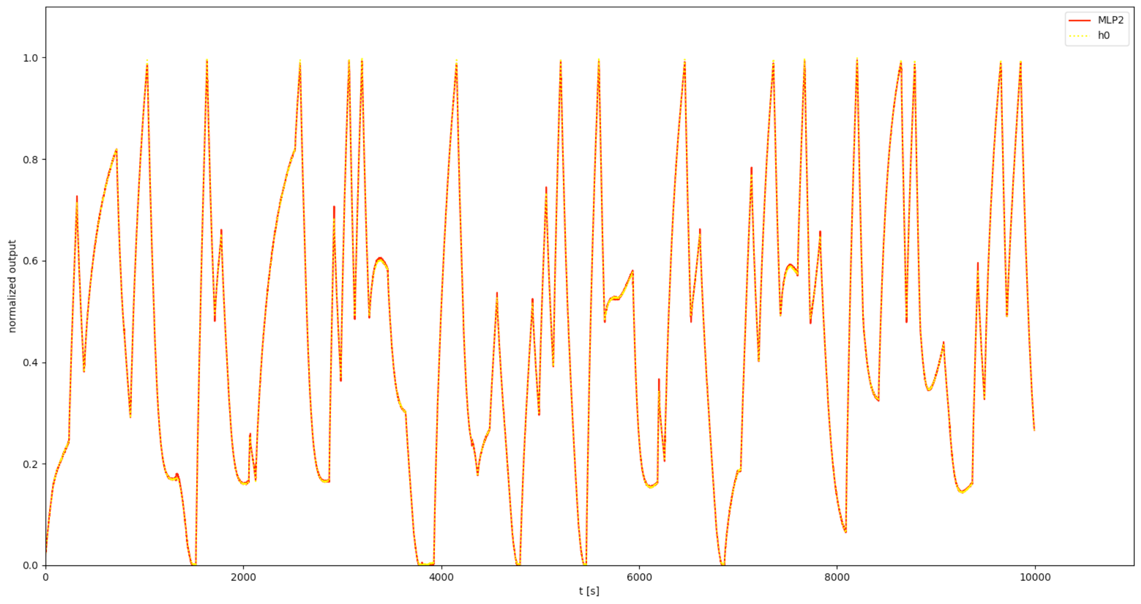


Figure 4.22: MLP2 predicts for tank 1

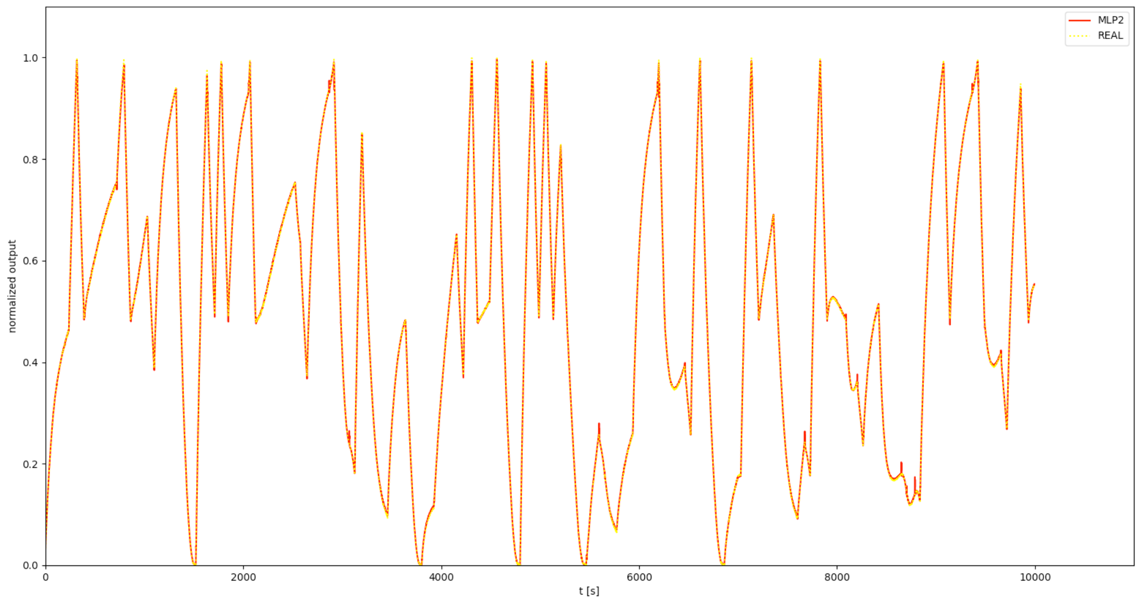


Figure 4.23: MLP2 predicts for tank 2

Fig. 4.27, for tank 1, Fig. 4.28, for tank 2 and Fig. 4.29 for tank 3. As can be observed from these figures, both frameworks can effectively approximate the system response, even in conditions not presented to the networks during the training stages. Nevertheless, it is noticed that the MLP-based predictors exhibit a response deterioration in the neighbourhood of pump transitions, particularly in the case of tank 3.

As mentioned above, multilayer perceptron architecture models nonlinear data, but its numbers of parameters can grow very high and it disregards spatial information. Meanwhile, convolutional architecture is simpler and cleaner than recurrent architecture. It

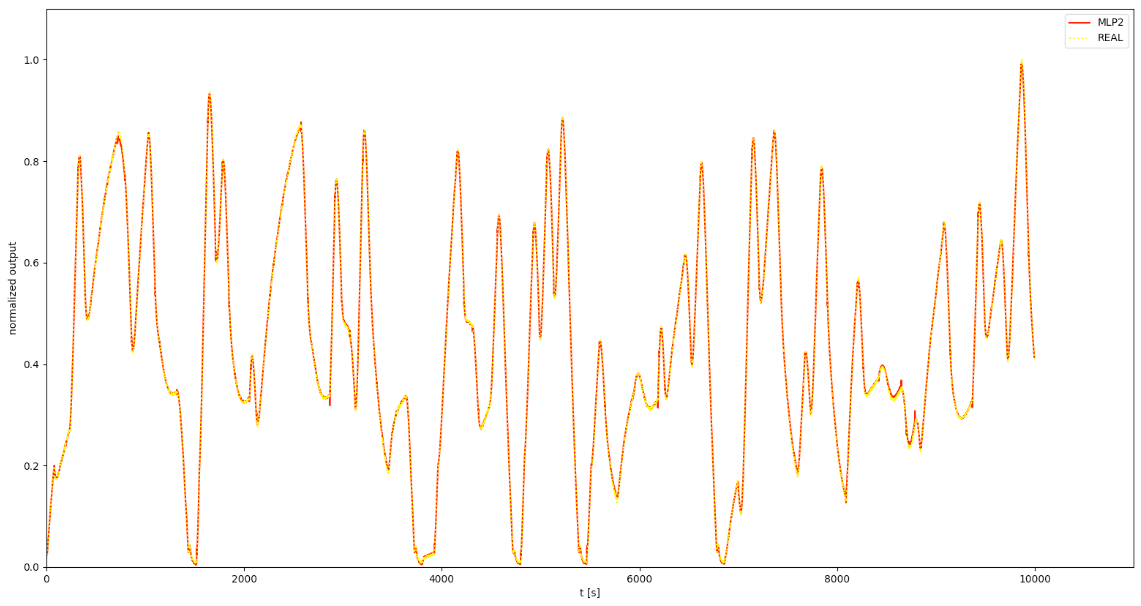


Figure 4.24: MLP2 predicts for tank 3

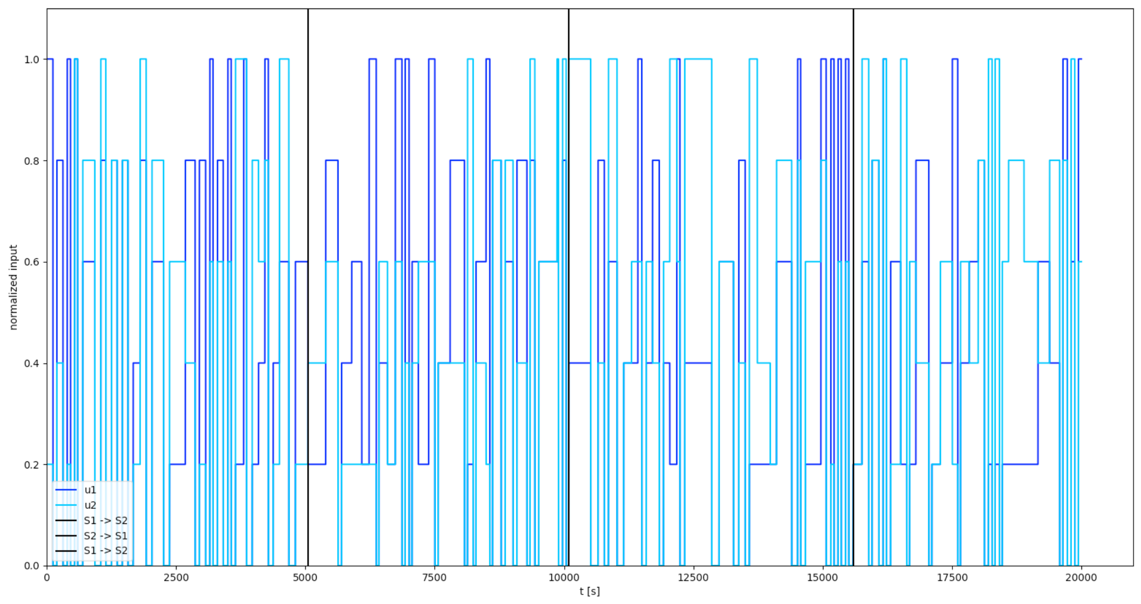


Figure 4.25: Test Data for comparison (TCN vs MLP) : Input Data

makes the convolutional architecture a good candidate for modelling complex sequence data[49]. Thus, we compare the two architectures to prove the merit of our method.

To proceed with verification of the reliability of the model proposed in this dissertation, we will use the new test dataset, Fig 4.25 and Fig 4.26, feed the input data thought the proposed frameworks.

As comparison, we present graphs for the water level prediction results on a test set for both architectures.

If we analyse the figures 4.27, 4.28 and 4.29 of the systems predictions, respectively

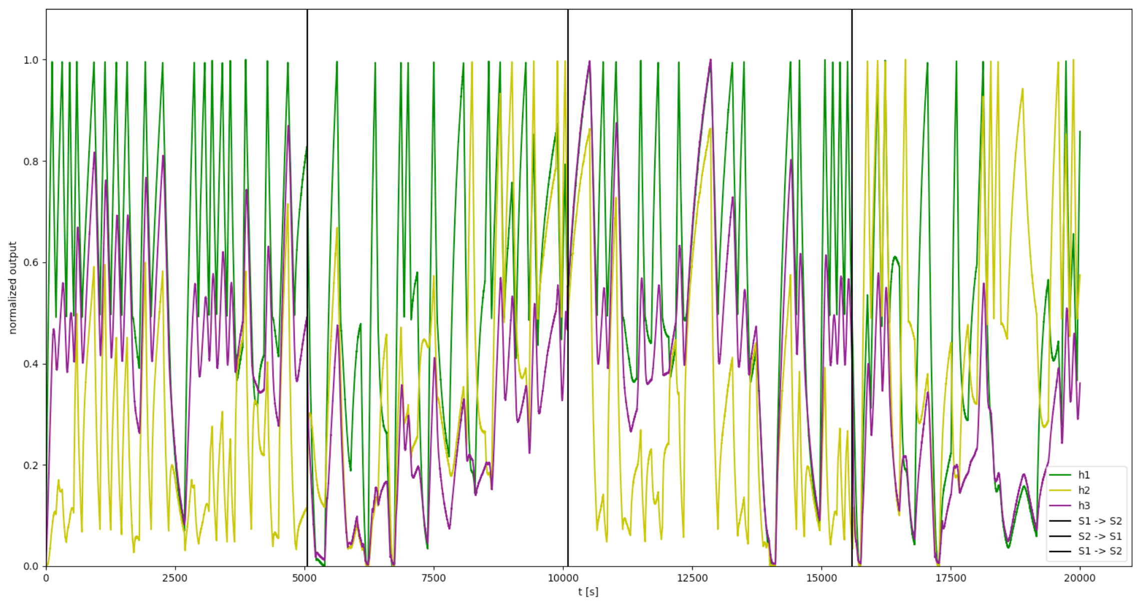


Figure 4.26: Test Data for comparison (TCN vs MLP) : Output Data

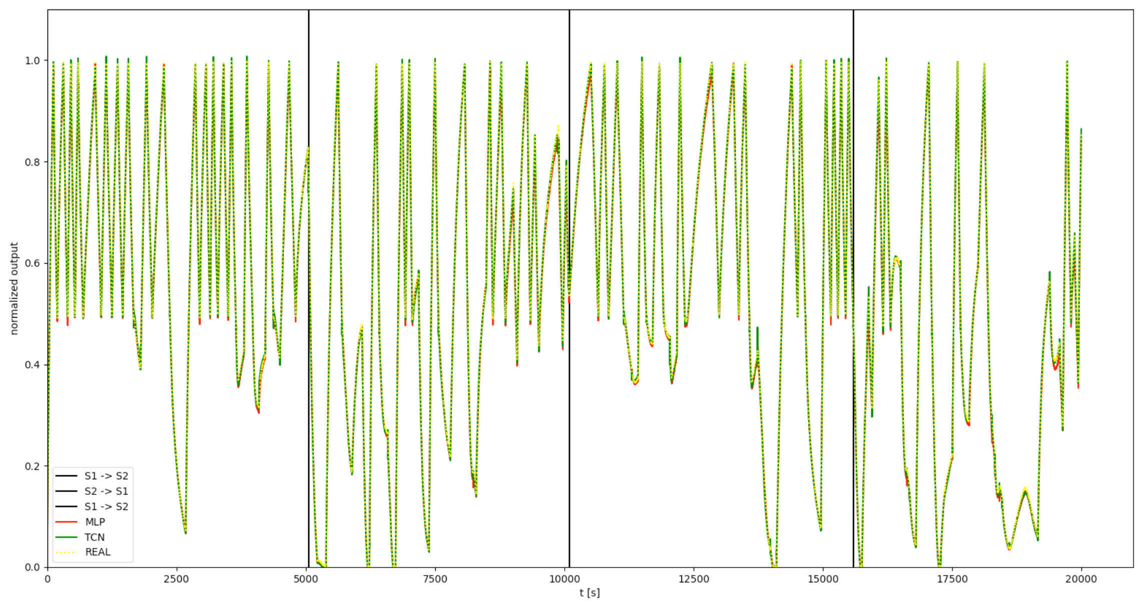


Figure 4.27: TCN and MLP predicts for tank 1

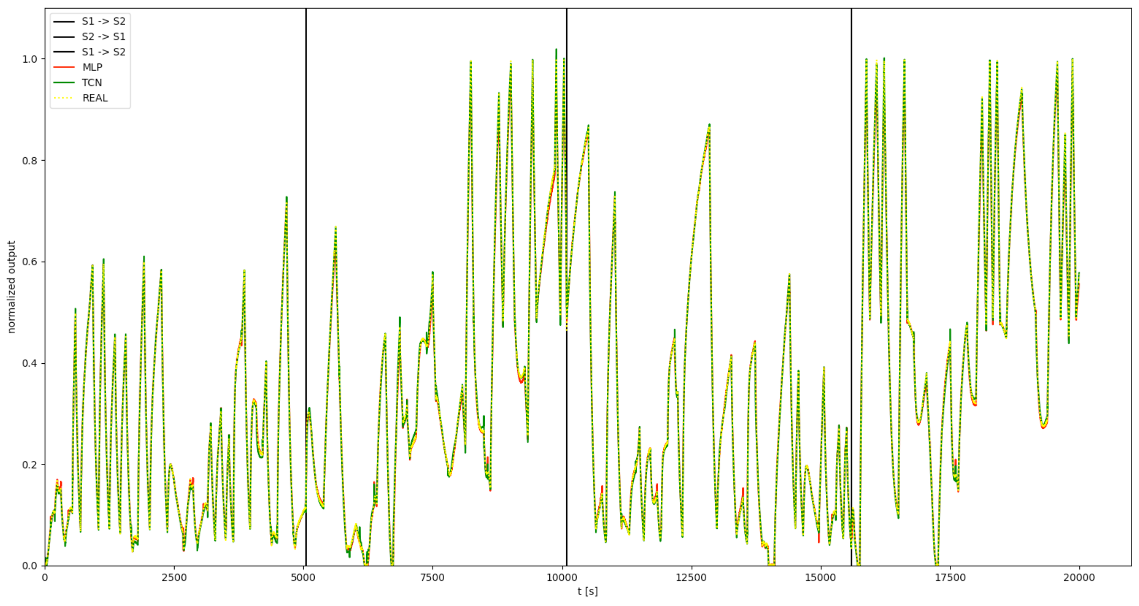


Figure 4.28: TCN and MLP predicts for tank 2

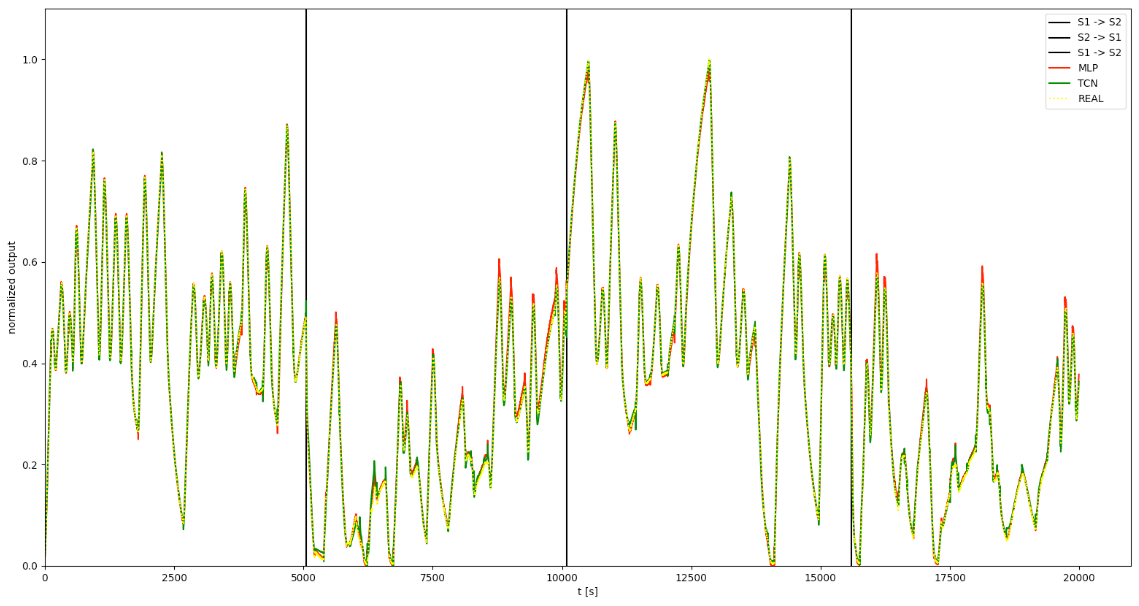


Figure 4.29: TCN and MLP predicts for tank 3

Model	Metric	h_1	h_2	h_3	Average
TCN	MSE	4.70311×10^{-5}	4.25167×10^{-5}	5.15767×10^{-5}	4.70415×10^{-5}
	MAE	4.73700×10^{-3}	4.64720×10^{-3}	4.49151×10^{-3}	4.62524×10^{-3}
	PCC	9.99721×10^{-1}	9.99693×10^{-1}	9.99509×10^{-1}	9.99641×10^{-1}
MLP	MSE	8.53658×10^{-5}	5.89277×10^{-5}	7.23104×10^{-5}	7.22013×10^{-5}
	MAE	6.81670×10^{-3}	4.72254×10^{-3}	6.31358×10^{-3}	5.95094×10^{-3}
	PCC	9.99747×10^{-1}	9.99806×10^{-1}	9.99599×10^{-1}	9.99717×10^{-1}

Table 4.4: Performance metrics.

for the water level in the tank 1, 2 and 3, we can see that for both architectures, the TCN and the MLP predictions follow quite good the real outputs. The MLP have a bit more trouble following the real output when the change is too abrupt or in transitions.

In order to quantify performances, three metrics have been considered, namely, the mean squared error (MSE), mean absolute error (MAE) and Pearson's correlation coefficient (PCC). AS can be observed from Table 4.4, the identification framework based on a temporal convolutional network topology outperforms the approach propped up on MLPs, in which each one of the neural network is trained to emulate a given structural configuration. On the other hand, as the trigger signal, under the form of a topological change, is transparently incorporated into the TCN architecture, as an additional input, it confers an extra degree of plasticity to the switching system predictor.

5 Conclusions

5.1 Final Remarks

The present work dealt with the problem of data driven-based hybrid systems modelling, in particular of those subject to structural modification or parametrical change induced by a switching signal.

The proposed approach relied on a temporal convolutional neural network, whose inputs comprise tapped delay lines of inputs and outputs from the system and, additionally, an extra array specifying the structural configuration.

Results based on a benchmark three-tank system, in which the structural configuration is specified by shut-off valves show clearly the effectiveness and potential of this class of black-box models, outperforming the approach considered for comparison purposes, based on two independent feedforward neural networks.

Said that, we can point out the following accomplished objectives:

- The development of an convolutional architecture capable of such kind of nonlinear modulation,
- This TCN architecture allows to model systems which have subsystems that can be internally chosen through a configuration variable,
- The development of an comparison architecture using two swallow multilayer perceptron neural networks,
- Relied on a data-driven approach to model the systems statio-temporal information, as well as, the structural configuration.

5.2 Future works

This dissertation thesis was focused on the search for a convolutional architecture capable of complex systems modeling. Modeling system can follow many approaches and this work was one of them.

However the focus here was to model the dynamic of a system capable for handling a structural configuration using a convolutional neural network, it was concluded that,in the future, other works should be conducted to address other issues.

Here, we leave out some paths of discovery that may be interesting to follow:

- A proposed path of investigation should be to test a GATCN (Graph Attention Temporal Convolutional Network) [15], which is a deep learning forecasting framework based on a graph attention network and a temporal convolutional network. This might be a good mechanism to handle this type of complex nonlinear modeling.
- Search for new modeling approaches, especially the ones using new technologies and methodologies such a machine learning, deep learning making use of the availability of computer power in our days.
- At last but not least, we suggest exploring the implementation of a neural architecture search to look of better architectures and models. The development of meta-heuristics for the architectures selecting to be able to feed the search oriented to specific parameters.

Bibliography

[1] A. Afshine and A. Shervine. *CS 230 - Recurrent Neural Networks Cheatsheet*. 2019. URL: https://stanford.edu/$%5Csim$shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (cit. on p. 18).

[2] P. Antsaklis. "Special issue on hybrid systems: theory and applications a brief introduction to the theory and applications of hybrid systems". In: *Proceedings of the IEEE* 88.7 (2000), pp. 879–887. DOI: 10.1109/JPROC.2000.871299 (cit. on p. 31).

[3] A. Barron. "Barron, A.E.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. on Information Theory 39, 930-945". In: *Information Theory, IEEE Transactions on* 39 (1993-06), pp. 930–945. DOI: 10.1109/18.256500 (cit. on p. 3).

[4] A. Bemporad and M. Morari. "Control of systems integrating logic, dynamics, and constraints". In: *Automatica* 35.3 (1999), pp. 407–427. ISSN: 00051098. DOI: 10.1016/S0005-1098(98)00178-2 (cit. on p. 32).

[5] M. S. Branicky, V. S. Borkar, and S. K. Mitter. "A unified framework for hybrid control: Model and optimal control theory". In: *IEEE Transactions on Automatic Control* 43.1 (1998), pp. 31–45. ISSN: 00189286. DOI: 10.1109/9.654885 (cit. on p. 32).

[6] E. CULURCIELLO. *THE HISTORY OF NEURAL NETWORKS*. 2019. (Visited on 2021-02-26) (cit. on p. 19).

[7] Y. N. Dauphin et al. "Language Modeling with Gated Convolutional Networks". In: (2017). arXiv: arXiv:1612.08083v3 (cit. on p. 1).

[8] *Dropout Layer - Keras.io*. URL: https://keras.io/api/layers/regularization_layers/dropout/ (cit. on p. 22).

[9] A. K. Duun-Henriksen et al. "Model identification using stochastic differential equation grey-box models in diabetes". In: *Journal of Diabetes Science and Technology* 7.2 (2013), pp. 431–440. ISSN: 19322968. DOI: 10.1177/193229681300700220 (cit. on p. 6).

[10] M. Fält and P. Giselsson. “System Identification for Hybrid Systems using Neural Networks”. In: (2019), pp. 1–13. arXiv: 1911.12663. URL: http://arxiv.org/abs/1911.12663 (cit. on pp. 32, 33).

[11] G. Ferrari-Trecate, D. Mignone, and M. Morari. “Moving horizon estimation for hybrid systems”. In: *IEEE Transactions on Automatic Control* 47.10 (2002), pp. 1663–1676. ISSN: 00189286. DOI: 10.1109/TAC.2002.802772 (cit. on p. 32).

[12] A. Gerón. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. Ed. by R. Roumeliotis and N. Tache. 2nd. O’Reilly Media, Inc, 2019, p. 820 (cit. on pp. 11–13, 15, 17–21).

[13] R. Goebel, R. G. Sanfelice, and A. R. Teel. “Hybrid Dynamical Systems”. In: *IEEE Control Systems Magazine* April (2009) (cit. on pp. 29–31).

[14] A. Grootveld et al. “Tracking of Dynamical Processes with Model Switching Using Temporal Convolutional Networks”. In: () (cit. on p. 28).

[15] G. Guo and W. Yuan. “Short-term traffic speed forecasting based on graph attention temporal convolutional networks”. In: *Neurocomputing* 410 (2020), pp. 387–393. ISSN: 18728286. DOI: 10.1016/j.neucom.2020.06.001. URL: https://doi.org/10.1016/j.neucom.2020.06.001 (cit. on pp. 28, 55).

[16] W. P. Heemels, B. De Schutter, and A. Bemporad. “On the equivalence of classes of hybrid dynamical models”. In: *Proceedings of the IEEE Conference on Decision and Control* 1 (2001), pp. 364–369. ISSN: 01912216. DOI: 10.1109/CDC.2001.980127 (cit. on p. 32).

[17] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 08936080. DOI: 10.1016/0893-6080(91)90009-T (cit. on p. 32).

[18] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208 (cit. on p. 3).

[19] S. Jayawardhana. *Sequence Models and Recurrent Neural Networks (RNNs)*. 2020. URL: https://towardsdatascience.com/sequence-models-and-recurrent-neural-networks-rnns-62cadeb4f1e1 (cit. on p. 16).

[20] Y. W. Jiang, Zhong-Ping. “Input-to-state stability for discrete-time nonlinear systems”. In: *Automatica* (2000). ISSN: 00200255 (cit. on p. 34).

[21] M. Johansson and A. Rantzer. “Computation of piecewise quadratic Lyapunov functions for hybrid systems”. In: *ECC 1997 - European Control Conference* (1997), pp. 2005–2010. DOI: 10.23919/ecc.1997.7082399 (cit. on p. 32).

[22] A. Juditsky et al. “Nonlinear black-box models in system identification: Mathematical foundations”. In: *Automatica* 31.12 (1995). Trends in System Identification, pp. 1725–1750. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(95)00119-1. URL: https://www.sciencedirect.com/science/article/pii/0005109895001191 (cit. on p. 3).

[23] N. Kalchbrenner, L. Espeholt, and K. Simonyan. “Neural Machine Translation in Linear Time”. In: (2016). arXiv: arXiv:1610.10099v2 (cit. on p. 1).

[24] A. M. Kalteh. “Rainfall-runoff modelling using artificial neural networks (ANNs): modelling and understanding”. In: 4.1 (2016), pp. 1–23 (cit. on p. 34).

[25] A. Kundu and D. Chatterjee. “A graph theoretic approach to input-to-state stability of switched systems”. In: *European Journal of Control* 29 (2016), pp. 44–50. ISSN: 09473580. DOI: 10.1016/j.ejcon.2016.03.003. arXiv: 1509.02668 (cit. on p. 33).

[26] P. Lara-Benítez et al. “Temporal convolutional networks applied to energy-related time series forecasting”. In: *Applied Sciences (Switzerland)* 10.7 (2020), pp. 1–17. ISSN: 20763417. DOI: 10.3390/app10072322 (cit. on p. 27).

[27] Y. Lecun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *proc. OF THE IEEE* (1998). URL: http://ieeexplore.ieee.org/document/726791/#full-text-section (cit. on p. 19).

[28] C. M. Lin, T. L. Le, and T. T. Huynh. “Self-evolving function-link interval type-2 fuzzy neural network for nonlinear system identification and control”. In: *Neurocomputing* 275 (2018), pp. 2239–2250. ISSN: 18728286. DOI: 10.1016/j.neucom.2017.11.009. URL: https://doi.org/10.1016/j.neucom.2017.11.009 (cit. on p. 27).

[29] L. Ljung. “Perspectives on System Identification”. In: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, pp. 7172–7184. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20080706-5-KR-1001.01215. URL: https://www.sciencedirect.com/science/article/pii/S1474667016400984 (cit. on p. 10).

[30] J. M. Lourenço. *The NOVAthesis LaTeX Template User’s Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[31] J. Lygeros, C. Tomlin, and S. Sastry. “Controllers for reachability specifications for hybrid systems”. In: *Automatica* 35.3 (1999), pp. 349–370. ISSN: 00051098. DOI: 10.1016/S0005-1098(98)00193-9 (cit. on p. 32).

[32] K. Madani. “Industrial and Real World Applications of”. In: (2006), pp. 11–26 (cit. on p. 11).

[33] D. Margolis. “The Importance of Physical System Modelling to Industry: System Models That Could Have Prevented Some Costly Mistakes”. In: *IFAC-PapersOnLine* 48.21 (2015). 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015, pp. 484–491. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.09.573. URL: https://www.sciencedirect.com/science/article/pii/S2405896315017024 (cit. on p. 1).

[34] McCulloch, Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133 (cit. on p. 11).

[35] F. J. Montáns et al. “Data-driven modeling and learning in science and engineering”. In: *Comptes Rendus Mécanique* 347.11 (2019). Data-Based Engineering Science and Technology, pp. 845–855. ISSN: 1631-0721. DOI: https://doi.org/10.1016/j.crme.2019.11.009. URL: https://www.sciencedirect.com/science/article/pii/S1631072119301809 (cit. on p. 2).

[36] S. Pan and K. Duraisamy. “Long-time predictive modeling of nonlinear dynamical systems using neural networks”. In: *Complexity* 2018 (2018). ISSN: 10990526. DOI: 10.1155/2018/4801012. arXiv: 1805.12547 (cit. on p. 11).

[37] P. Pepe. “ISS small-gain theorem for networked discrete-time switching systems”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 1900–1905. ISSN: 24058963. DOI: 10.1016/j.ifacol.2020.12.2581. URL: https://doi.org/10.1016/j.ifacol.2020.12.2581 (cit. on p. 33).

[38] D. Pepyne and C. Cassandras. “Optimal control of hybrid systems in manufacturing”. In: *Proceedings of the IEEE* 88.7 (2000), pp. 1108–1123. DOI: 10.1109/5.871312 (cit. on p. 31).

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0 (cit. on p. 14).

[40] S. S., J. I. Zong Chen, and S. Shakya. “Survey on Neural Network Architectures with Deep Learning”. In: *Journal of Soft Computing Paradigm* 2.3 (2020), pp. 186–194. DOI: 10.36548/jscp.2020.3.007 (cit. on p. 2).

[41] D. Saha. “How The World Became Data-Driven, And What’s Next”. In: *Forbes* (2020). URL: https://www.forbes.com/sites/googlecloud/2020/05/20/how-the-world-became-data-driven-and-whats-next/?sh=4ce026e657fc (cit. on p. 2).

[42] S. Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 2021-02-26) (cit. on p. 19).

[43] E. Shelhamer, J. Long, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2572683. arXiv: 1411.4038 (cit. on p. 19).

[44] B. I. Silva et al. "Modeling and verifying hybrid dynamic systems using CheckMate". In: *Proceedings of 4th International Conference on Automation of Mixed Processes* (2000), pp. 323–328 (cit. on p. 32).

[45] K. Simonyan et al. "W n : a g m r a". In: (2016), pp. 1–15. arXiv: arXiv:1609.03499v2 (cit. on p. 1).

[46] J. Sjöberg et al. "Nonlinear black-box modeling in system identification: a unified overview". In: *Automatica* 31.12 (1995), pp. 1691–1724. ISSN: 00051098. DOI: 10.1016/0005-1098(95)00120-8 (cit. on pp. 1, 3, 6–10).

[47] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 22).

[48] Y. VERMA. *Comprehensive Guide to Different Pooling Layers in Deep Learning*. 2021. URL: https://analyticsindiamag.com/comprehensive-guide-to-different-pooling-layers-in-deep-learning/ (cit. on pp. 22, 24).

[49] P. Wu et al. "Data-driven reduced order model with temporal convolutional neural network". In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112766. ISSN: 00457825. DOI: 10.1016/j.cma.2019.112766. URL: https://doi.org/10.1016/j.cma.2019.112766 (cit. on p. 50).

[50] Z. Xu et al. "Sparsely-Connected Cascade Recurrent Neural Network-Based Nonlinear Equalizer for a 100-Gb/s PAM4 Optical Interconnect". In: *Asia Communications and Photonics Conference 2021*. Optica Publishing Group, 2021, M5H.4. DOI: 10.1364/ACPC.2021.M5H.4. URL: http://opg.optica.org/abstract.cfm?URI=ACPC-2021-M5H.4 (cit. on p. 3).

[51] X. Yuan et al. "A dynamic CNN for nonlinear dynamic feature learning in soft sensor modeling of industrial process data". In: *Control Engineering Practice* 104.August (2020), p. 104614. ISSN: 09670661. DOI: 10.1016/j.conengprac.2020.104614. URL: https://doi.org/10.1016/j.conengprac.2020.104614 (cit. on p. 34).

[52] D.-X. Zhou. "Universality of deep convolutional neural networks". In: *Applied and Computational Harmonic Analysis* 48.2 (2020), pp. 787–794. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2019.06.004. URL: https://www.sciencedirect.com/science/article/pii/S1063520318302045 (cit. on p. 3).

2022 Modeling of complex nonlinear dynamic systems using temporal convolution neural networks Vasco Bastos

NOVA SCHOOL OF SCIENCE & TECHNOLOGY