Neural network-based imitation of model predictive control for power converters

Aitor Teran, Cristina Roche, Pere Izquierdo

Energy Technology, MCE3-922 and PED3-943, 2019-12

3rd Semester Master Project

Programming software: Keras API of the TensorFlow library, programmed in Python 3. Simulation software: Matlab R2019b with Simulink 10.0, Simscape Electrical 7.2 and PLECS 4.3. Laboratory tests: dSPACE. Text formatting software: Overleaf/Latex. Image formatting software: Inkscape.

Energy Engineering
Aalborg University
http://www.aau.dk

Title:
Neural network-based imitation of model predictive control for power converters

Theme:
Control of power electronic converters

Project Period:
Autumn Semester 2019

Project Group:
MCE3-922 and PED3-943

Participant(s):
Aitor Teran
Cristina Roche
Pere Izquierdo

Supervisor(s):
Tomislav Dragičević

Copies: 1

Page Numbers: 82

Date of Completion:
December 16, 2019

Abstract:

Machine learning and neural networks have been at the forefront of many technological breakthroughs in recent years. This study aims to make use of these techniques to implement the control scheme of power converters, imitating the more widespread model predictive control method. Model predictive control (MPC) has been shown to offer great advantages compared to more traditional techniques, but it has an important drawback in its high computational requirements. By predicting the behavior of the system for longer periods of time, performance can be improved at the cost of an exponentially increasing computational time. This can be potentially overcome by imitating such a method using a neural network, which can greatly reduce computational cost. Training data is collected using an accurate simulation model of conventional MPC schemes, for a wide range of parameters. This data is used to train an artificial neural network using the Python implementation of Tensorflow, which allows for a highly flexible neural network structure. The trained neural network is then experimentally verified by implementing it in a laboratory environment.

Preface

This project has been carried out within the Aalborg University Energy Engineering Master's degree programme, by a group formed by three students from the Mechatronic Control Engineering and Power Electronics and Drives specializations. This report aims to show the results and insights obtained during the development of the project. It has been written in the period comprised between the 2nd of September and the 16th of December of the year 2019. The authors would like to thank Mateja Novak and Tomislav Dragičević for their support and assistance during the completion of this project.

Aalborg University, December 16, 2019

Aitor Teran
<ateran18@student.aau.dk>

Cristina Roche
<croche18@student.aau.dk>

Pere Izquierdo
<pizqui18@student.aau.dk>

Contents

Nomenclature

Symbols

λ	Weighting factor of the cost function derivative term	$[-]$
$\dot{\mathbf{x}}$	Derivative of the state vector in a state-space model	$[-]$
$\mathbf{A}$	System coefficient matrix in a state-space model	$[-]$
$\mathbf{B}$	Input coefficient matrix in a state-space model	$[-]$
$\mathbf{b}$	Neural network bias vector	$[-]$
$\mathbf{C}$	State coefficient matrix for the output of a state-space model	$[-]$
$\mathbf{D}$	Input coefficient matrix for the output of a state-space model	$[-]$
$\mathbf{G}$	Discrete state coefficient matrix of a state-space model	$[-]$
$\mathbf{H}$	Discrete input coefficient matrix of a state-space model	$[-]$
$\mathbf{u}$	Input vector in a state-space model	$[-]$
$\mathbf{W}$	Neural network weight matrix	$[-]$
$\mathbf{x}$	State vector in a state-space model	$[-]$
$\mathbf{y}$	Output vector in a state-space model	$[-]$
ω_{ref}	Angular velocity of the voltage reference	$[rad/s]$
$\vec{i}_L$	Inductor phase current or inverter output phase current	$[A]$
$\vec{i}_{load}$	Load phase current	$[A]$
$\vec{v}_c$	Phase voltage drop across the capacitor in the LC filter	$[V]$
$\vec{v}_f$	Phase voltage vector output of the inverter	$[V]$

$\vec{v}_{ref}$	Voltage reference vector	$[V]$
C_f	Capacitance of the capacitor of the LC filter	$[F]$
f_{sw}	Switching frequency	$[Hz]$
fs	Sampling frequency	[Hz]
J	Cost function used for neural network training	$[-]$
j	Imaginary unit	$[-]$
L_f	Inductance of the inductor of the LC filter	$[H]$
M_a	Modulation index	$[-]$
R_f	Internal resistance of the inductor in the LC filter	$[\Omega]$
R_{load}	Load resistance	$[\Omega]$
s	Laplace transform variable	$[-]$
S_i	Leg i of the inverter	$[-]$
t_k	Current sampling instant	$[-]$
T_s	Sampling time	[Hz]
v_f	Output voltage of the inverter of total voltage drop across the LC filter	$[V]$
v_i	Phase voltage drop across the inductor in the LC filter	$[V]$
V_{dc}	DC-link voltage	$[V]$
V_{XN}	Phase voltage between phase X and neutral	$[V]$
V_{XY}	Line-to-line voltage between phase X and phase Y	$[V]$

Abbreviations

AC	Alternating current
ANN	Artificial Neural Network
API	Application Programming Interface
CCE	Categorical cross-entropy
CL	Closed-Loop

DC	Direct current	
DC-AC	Direct current to alternating current	
FS-MPC	Finite-set Model Predictive Control	
LPF	Low pass filter	
LTI	Linear Time-Invariant	
ML	Machine Learning	
MPC	Model Predictive Control	
OL	Open-Loop	
PWM	Pulse-Width Modulation	
ReLU	Rectified Linear Unit	
RMSProp	Root Mean Square Propagation	
sat	Saturation function	
SGD	Stochastic Gradient Descent	
sgn	Sign function	
SVM	Space vector modulation	
THD	Total Harmonic Distortion	
VSI	Voltage-source inverter	
G	FS-MPC cost function	$[-]$
G_d	Derivative term of the FS-MPC cost function	$[-]$

Chapter 1

Introduction

Artificial neural networks (ANN) have been a fundamental part of many research breakthroughs in recent years. They present very interesting properties that motivate their potential application in fields as diverse as finance, medicine, climate science, or even the arts, as well as engineering. These properties can be summarized as [1]:

- The parallel structure of artificial neural networks allows for high-speed processing. This means that neural networks are computationally very cheap, allowing for complex structures to be executed in short processing times.
- Artificial neural networks are universal function approximators, as proven formally in 1989 by Hornik, Stinchcombe, and White [2]. Any arbitrarily complex continuous function can be approximated by a neural network containing at least one hidden layer and using a non-polynomial activation function.
- Neural networks are able to model relationships in data without requiring any information other than the data itself. In practice, this means that neural networks are able to model relationships for which no mathematical description exists or for which the mathematical description is not sufficiently accurate, assuming that enough data is available.

By considering these properties together, the great potential of artificial neural networks becomes obvious. Using neural networks arbitrarily complex relationships in any type of data can be modelled accurately and in a computationally efficient way.

1.1 Problem analysis

Finite-set model predictive control (FS-MPC) is an advanced control method that has been gaining attention in recent years in its application to the field of power electronics. It offers advantages over more traditional methods due to its straightforward design, easy inclusion of control objectives, and discrete nature, which makes it a natural fit for power converters [3]. To better understand the benefits that neural networks can provide to such an algorithm, a short introduction to FS-MPC is provided here.

Essentially, an FS-MPC algorithm makes use of a model of the system to be controlled (often formulated in state-space) to predict in real time the future values of the system's states. Predictions are obtained for each possible control action in the power converter. Consider a simple DC-to-DC converter with a single switching device: the system has then only two possible states, according to the state of the device, and its switching is the control action to be considered. Once the predictions have been obtained, they are compared by making use of a previously defined cost function, which should be constructed to minimize the error between states and its references. The flexibility of FS-MPC is most noticeable in this step, as virtually any condition can be penalized as desired. For example, switching losses can be minimized by modifying the cost function in such a way that favors the current switching state if the control error is not significantly large. Assuming the model of the system to be accurate enough, FS-MPC often yields better stationary and transient response than linear closed-loop control structures.

FS-MPC has, however, an important shortcoming. When predicting the values of states for more than a single step ahead, the number of calculations to be performed increases exponentially. A traditional six-switch three-phase inverter, for example, has a total of eight switching states. When making predictions for two steps ahead, eight predictions must be calculated for each prediction in the first step; resulting in a total of 64 calculations. More generally, the number of model evaluations that must be performed in FS-MPC follows an exponential law of the form n^h, where n is the number of possible states of the system and h the prediction horizon, or number of steps ahead to be evaluated. More details on FS-MPC and its application to the six-switch inverter can be found in Chapter 2.

The exponentially increasing computational requirements of FS-MPC means that it is often not possible to implement such an algorithm that is able to run in a time short enough to guarantee adequate performance while using a prediction window larger than very few steps ahead. For multi-level and multi-cell converters, for which many more possible states exist than for a six-switch inverter, this problem

is further accentuated.

As previously stated, artificial neural networks are able to approximate any function; this obviously includes an FS-MPC algorithm. Moreover, they are very computationally efficient. Therefore, the main goal of this project is to obtain an artificial neural network that is able to imitate FS-MPC algorithms, achieving similar performance while reducing its computational requirements.

1.2 State of the art

The application of neural networks to the control of power converters has been considered in research since as early as 1989 [4]. Applications of neural networks to control systems had already been considered for years, especially with regards to [5]:

- Modelling of nonlinear systems. The ability of neural networks to approximate any continuous function can allow complex nonlinear systems to be modelled in a more computationally efficient way. Moreover, such a modelling process does not require a explicit formulation, as the network can be trained by simply using input and output data obtained from simulation.
- System identification. Since neural networks can be trained with any data, the modelling of physical systems can be simply based on gathering enough data from the system to be modelled. Both forward and backward dynamics of any system can be modelled using such an approach.
- Control structures. Any arbitrarily complex control structure can, in principle, be substituted by a neural network that acts as an imitator.

Some areas of power electronics where neural networks have proven to be useful are: delay-less filtering and waveform processing, the design of estimators for motor drives (such as position, speed, or flux), as well as controller design.

There has been research showing the potential of neural networks for use in the control of power electronics systems. To provide some examples, [6] details the development of a recurrent neural network for control of a grid-connected DC-to-AC converter without making use of the back-propagation algorithm, [7] uses a neural network to control a DC-to-DC converter for use in photovoltaic applications, while [8] uses a neural network model of a photovoltaic system to obtain optimal control actions. Most similarly to the goals of this project, [9] shows how a neural

network imitator of MPC can be used to control a multilevel converter, albeit with a relatively low imitation accuracy of around 90%.

1.3 Problem statement and methodology

The main goal of this project is to develop a neural network-based controller that is able to imitate the behavior of finite-set model predictive control algorithms for use in a three-phase, six-switch, DC-to-AC converter.

To do so, one or several FS-MPC algorithms will be implemented as *Matlab* code and verified using *Simulink*. The MPC algorithm will be based on a discretized state-space model of the three-phase converter with an LC output filter and a purely resistive load. The verification of the model will be performed on a *Simscape* model of the physical system, which will itself be based on the experimental setup made available to the authors. FS-MPC algorithms will be developed for predictions of one, two, and three steps ahead.

Once the MPC algorithm has been verified, it can be used to generate data for use in training the artificial neural network. Training data will be obtained as a grid for all combinations of input data, to ensure that the network can operate for all values inside the operating limits of the setup. Validation and test data will be obtained by picking random points inside the operating range.

The neural network must be designed and tested for its many parameters, such as number of neurons, number of layers, or activation functions; as well as the paremeters required for the training of the network, which include cost functions, optimizers, batch size, or number of training epochs. The network will be designed and trained using the implementation of the *TensorFlow* library for *Python 3*, and making extensive use of the *Keras* API.

Once the neural network has been obtained, it will be translated to *Matlab* code for its testing in *Simulink*. In this way, the performance of the developed controller can be verified as well as compared to FS-MPC in simulation.

With all controllers verified in simulation, they will be tested in the laboratory setup using a *Simulink* model for use through *dSpace*. Tests will be performed to check both stationary and transient performance, and the robustness of the controllers to variations in voltage and frequency will also be verified.

1.3.1 Performance criteria

To evaluate the performance of the trained networks the main metric that will be used is accuracy, which measures the ratio of correct predictions (predictions matching their label) to total predictions. Confusion matrices will also be used, which expand the information provided by the single accuracy metric.

The performance of the different controllers will be mainly evaluated by measuring total harmonic distortion (THD) on the three-phase output voltage across the load resistors. The analysis of transient responses will rely on common metrics such as response time or overshoot.

1.3.2 Limitations and assumptions

The modelling and design of controllers will be based on the available experimental setup. The operating range for which the neural networks are to be trained will depend on the parameters and limits of the setup, some of which are fixed and others variable.

When testing in the laboratory, the sampling period of *dSpace* will be fixed to 20 μs, as MPC requires high frequencies to result in good performance but its computational requirements prevent it from being able to run in real time at shorter periods.

1.4 Project outline

Chapter 2 details the modelling process and implementation of an FS-MPC algorithm for a three-phase, six-switch DC-to-AC converter. Chapter 3 explains the principles of artificial neural networks and the procedure to obtain a neural network imitator of the MPC algorithm. Chapter 4 documents the analysis of the laboratory setup and the obtained controllers, and presents the obtained experimental data. Finally, in Chapters 5 and 6 the results and insights obtained in this project are discussed, with Chapter 7 suggesting possibilities for future work based on it.

Regarding appendices, Appendix A expands the theoretical background of artificial neural networks introduced in chapter 3, appendix B gives an introduction to the Lagrange polynomial approximation used in the implemented MPC algorithms, and appendix C gives details on the chosen optimization algorithm.

Chapter 2

MPC

2.1 Three-phase inverter description

The converter to be controlled in this project is a two-level three-phase inverter, which is also known as a 6-switch converter. It is considered to be one of the most traditional DC-to-AC converter and forms the basis for many new and advanced converters with different topologies being currently researched. As it is a three-phase converter it consists of three phase legs with two power switches in each. This enables each leg to connect either to the positive load terminal or the negative one. A simplified schematic of the structure of the converter is shown in Figure 2.1. The main goal is to generate a three-phased sinusoidal waveform that emulates the reference voltage signal with a frequency and amplitude given by e.g. a power grid.

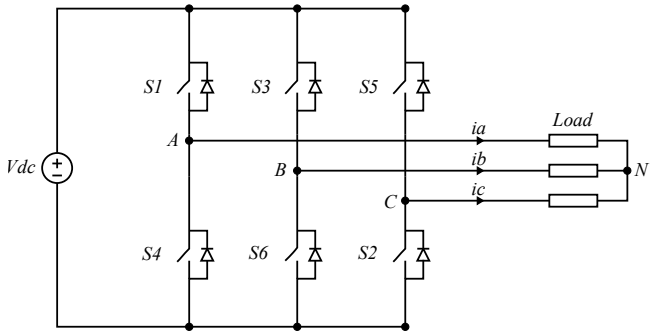

Figure 2.1: Simplified schematic of the traditional two-level voltage source three-phase inverter.

There are different control schemes to generate these output signals, each one using its own logic to control the combination of the switches with time.

In this case, it is not considered when both transistors in one leg are open, in order to prevent floating voltages from occurring. Since this inverter uses a voltage source, the switching combinations that short-circuit the input (when both transistors in one leg are closed) are not feasible because they would provide ideally infinite current to the circuit and might destroy the switching devices. Therefore, there will always be one switch in the on state and the other in the off state in each leg. This simplifies the circuit control by reducing from 6 switches to combine between each other to just 3 combinations, i.e. three legs and two positions for each, thus from $2^6 = 64$ to $2^3 = 8$ possible states $\vec{v}_0, ...\vec{v}_7$ [10]. As it will be seen in Section 2.3, this reduced number of possible control outputs allows for the design of an MPC algorithm with lower computational cost.

However, this inability to have both of the switching devices in the same leg turned-on introduces what is called 'dead time'. This safety method creates a blanking time in which both switches are turned-off every time they have to switch to another state, this is why the turn-off time of a switching device is always longer than the turn-on time.

To simplify the calculations the diodes will be considered ideal, which means that they are assumed to conduct in the forward direction with no voltage drop and to not conduct in the reverse direction, always without any transients or parasitics. Switching devices are considered infinitely fast when changing state. This way, the output line-to-line and phase voltage provided by each state can be derived though a basic calculation using the current divider equation.

From the eight possible states, six of them correspond to basic vectors and the other two are called zero vectors and give zero volts on the terminals. The switching between states generates output voltage and current signals that take discrete values. The available values that the phase voltage square wave can take are only five different values and are the ones shown in Table 2.1.

$States(\vec{v}_f)$	$S1$	$S2$	$S3$	V_{AB}	V_{BC}	V_{CA}	V_{AN}	V_{BN}	V_{CN}
0	0	0	0	0	0	0	0	0	0
1	1	0	0	V_{dc}	0	$-V_{dc}$	$\frac{2}{3}V_{dc}$	$-\frac{1}{3}V_{dc}$	$-\frac{1}{3}V_{dc}$
2	1	1	0	0	V_{dc}	$-V_{dc}$	$\frac{1}{3}V_{dc}$	$\frac{1}{3}V_{dc}$	$-\frac{2}{3}V_{dc}$
3	0	1	0	$-V_{dc}$	V_{dc}	0	$-\frac{1}{3}V_{dc}$	$\frac{2}{3}V_{dc}$	$-\frac{1}{3}V_{dc}$
4	0	1	1	$-V_{dc}$	0	V_{dc}	$-\frac{2}{3}V_{dc}$	$\frac{1}{3}V_{dc}$	$\frac{1}{3}V_{dc}$
5	0	0	1	0	$-V_{dc}$	V_{dc}	$-\frac{1}{3}V_{dc}$	$-\frac{1}{3}V_{dc}$	$\frac{2}{3}V_{dc}$
6	1	0	1	V_{dc}	$-V_{dc}$	0	$\frac{1}{3}V_{dc}$	$-\frac{2}{3}V_{dc}$	$\frac{1}{3}V_{dc}$
7	1	1	1	0	0	0	0	0	0

Table 2.1: Line-to-line and phase voltage output of each state of the three-phase inverter [10].

Binary digits are used to indicate whether a switch is turned-on or off, where the value 1 means that the top switch is closed, and zero that it is open.

The output signal that the switches of the inverter deliver to the load is a staircase-shaped one. It is required to be filtered before being supplied to the load for the it to receive smooth sinusoidal waves.

However, the sinusoidal phase voltage signal delivered to the load can reach at most $V = \frac{2}{3}V_{dc}$ in case of maximum over-modulation and at the expense of non-linearities and low frequency harmonics introduction. Nevertheless, in the linear range or without over-modulation ($M_a < 1$) the sinusoidal fundamental amplitude will be lower than that value, assuming that simple carrier-based pulse-width modulation is used. The maximum output is therefore defined by the DC link voltage and the modulation scheme employed.

It can be seen that the spatial addition of the three vectors of the AC balanced voltage phase signals *abc* and frequency f results in a rotating vector with constant speed $\omega = 2\pi f$; that is passing through the phase axes drawn radially separated by 120°, as depicted in Figure 2.2 and Equation 2.1.

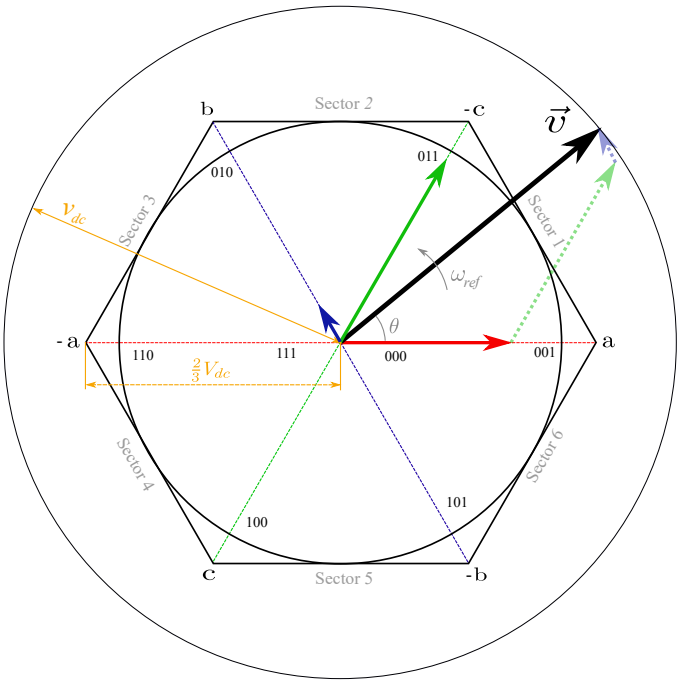

Figure 2.2: Space vector hexagon formed by the three phases. $\vec{v}$ is the resultant space vector, obtained by adding the three vectors. It can be noticed that the magnitude of the space vector is constant, assuming a balanced three-phase system with constant amplitude. Picture based on [11].

$$\vec{v}_{ref} = \frac{2}{3} V_{dc} \left(v_a e^{0j} + v_b e^{\frac{2\pi}{3} j} + v_c e^{\frac{-2\pi}{3} j} \right) \tag{2.1}$$

One of the most typically used modulation techniques for converters is based on this representation and is therefore called Space Vector Modulation (SVM).

Typically, a linear Closed-Loop (CL) controller is used for generating the voltage reference command signal. Then, SVM is a modulation technique that synthesizes the voltage vector, its amplitude and direction, into the switching devices turn-on/off signals.

The representation of the three phases divides the vector space into six sectors. The phase axes coincide in the hexagon with the six switching states because each state except from the zero ones gives either a maximum or a minimum value in one phase. In this way, state vectors separated by 180° are inverse of each other, i.e. one represents a maximum positive voltage in a phase and the other the maximum negative voltage in that phase, this can be observed by looking at the binary digits in Figure 2.3.

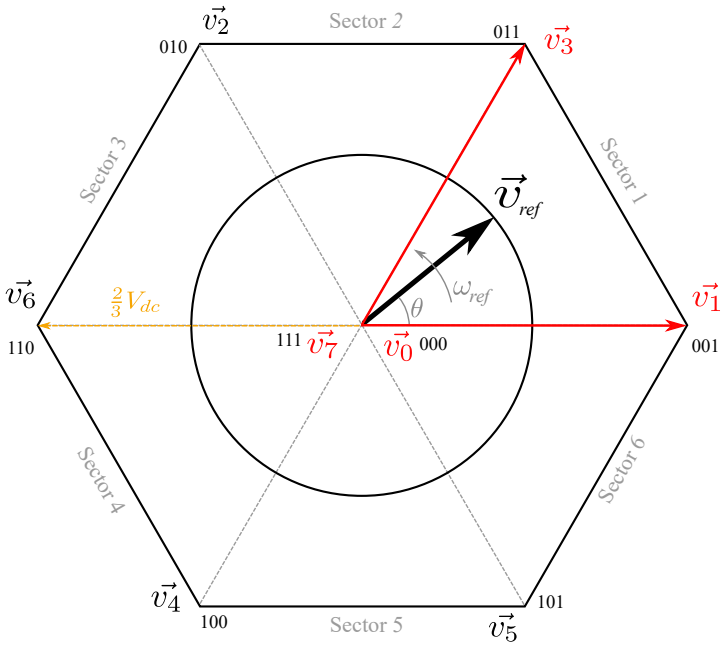

Figure 2.3: Hexagon representation of SVM for a two-level three phase inverter. The states that are active for the drawn space vector are shown in red. Picture based on [11].

First, the sector in which the reference vector is instantaneously located has to be identified because it determines the possible four states that are going to be used. Two of them are the adjacent ones in that sector and the other are always the two zero vectors. SVM decomposes the voltage vector by projecting it onto the three phases and uses a high frequency triangle signal, called carrier, as a trigger to determine the duty cycles; i.e., to determine the sequence of the states and the percentage of time that each state is selected during a sampling period in order to follow the reference. This is shown in Figure 2.4. The frequency of the reference waves must be much lower than the carrier signal in order to maximize the frequency of switching harmonics. In cases where the carrier wave amplitude is larger than the phase ones, the converter operates in linear operation range.

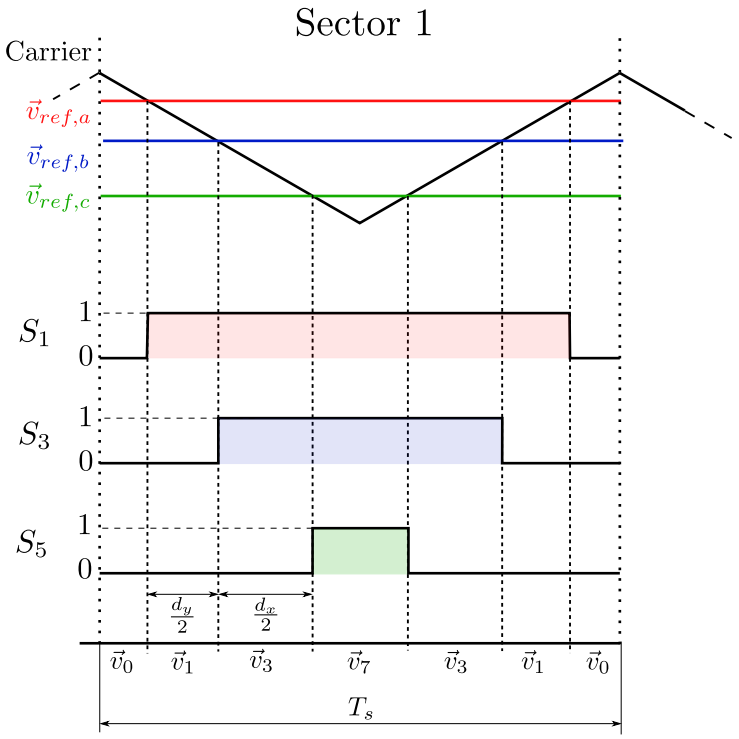

Figure 2.4: Switching sequence and duty cycles calculated with a triangular carrier wave for the first sector using SVM. The horizontal length shadowed in each phase colour indicates the time during which each switch remains on. Picture based on [11].

The fast and proper combination of the four states can reproduce from discrete voltage values the smoothly rotating reference voltage vector. In order to reduce harmonics and switching losses the sequence of the switching changes only the state of one leg at a time [11]. In this way the optimal state sequence followed is first the state 0, then the two adjacent states of the sector, followed by the state 7. Additionally, the sequence of half a sampling period is the same as the other half mirrored because the sequence is symmetric.

The calculation of the duty cycles is directly derived from the intersection between the phase voltage signals and the carrier wave for each sampling time. The shadowed region created by each phase indicates the total time that the switching leg has value 1. This calculation can be visualized in Figure 2.4 for the first sector. The duty cycle calculation and switching states used vary from sector to sector.

In SVM, the length that each of the two zero vectors is active in a sampling period lasts for half of the total required zero-state time, and is called dwell time [12]. By introducing the zero states in such a manner, SVM allows for larger DC-link voltage utilization by introducing a triangular wave that reduces the voltage peaks

of each phase. This wave is cancelled out at the load.

2.2 Converter state space model

The voltage and current signals are pulse-width modulated by fast switching combinations. A sine wave can be synthesized because the higher the frequency of the switching harmonics is with respect to that of the sine wave, the further the fundamental frequency is from that of the undesired signals. In this way, if the fundamental frequency is at a much lower frequency than the noise signals, it is much easier to design a filter to remove them.

A filter is aimed to change the frequency contents of a signal. In this case, a Low-Pass Filter (LPF) is used for the purpose of attenuating the high frequency signals of both converter output voltage and current. The filter used in this application is a second-order LPF. Therefore its gain presents a resonance peak, and from the cut-off frequency onward it attenuates the signals with -40 dB per decade. It also introduces delay in the signal, with -90°delay at the cut-off frequency and -180°from one decade more than the cut-off frequency onward. Hence, for such a filter, the cut-off frequency of the filter should always be placed at at least a decade higher than the fundamental frequency of the reference signals.

Physically, a LPF is accomplished through a LC-circuit, which is located between the inverter and the load as shown in Figure 2.5. It consists of an inductor with inductance $L_f = 2.4\ mH$ in series with the inverter and a capacitor in parallel with the load, with capacitance $C_f = 14.2\ \mu F$. The output phase current from the inverter passes through the inductor i_L, and due to the internal inductor resistance $R_f = 0.1\ \Omega$ it creates a voltage drop across it v_L. The voltage drop in the capacitor is the same as in the load, v_c. The voltage drop across the filter v_f is the addition of v_c and v_L. The current that flows to the load R_{load} is called i_{load}. This filter structure is connected to each one of the converter phase output terminals in order to filter the three phases.

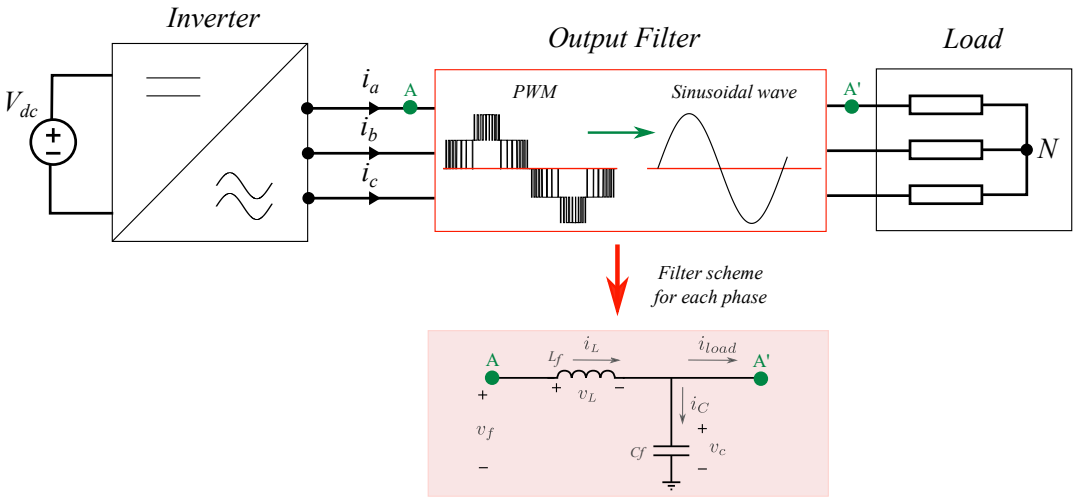

Figure 2.5: LC-filter schematic and variable names for the model.

The filter configuration is chosen due to the physical properties of the components. An inductor has low impedance at low frequencies and increasing impedance at higher frequencies. This way, it is placed in series so as to attenuate only the undesired high frequencies. However, a capacitor has the opposite physical behaviour, i.e. inductors an capacitors are duals of each other. A capacitor has low impedance at high frequencies and its impedance keeps increasing for lower frequencies. Hence, if the capacitor is placed in parallel, at high frequencies its impedance is low, and thereby those undesired high frequency components take the capacitor path directed to the ground and do not reach the load, because the load path offers higher resistance. As a summary, both the inductor in series and the capacitor in parallel avoid the load to receive the high undesired frequency signals created by the switching of the inverter, which is why the filter is second-order.

Both inductor and capacitor dynamics are shown in Equation 2.2 and are used to describe the system through the voltage and current equations, this is shown in Equation 2.3.

$$v_L = L_f \frac{di_L}{dt} \qquad i_c = C_f \frac{dv_c}{dt} \tag{2.2}$$

$$\begin{cases} L_f \frac{di_L}{dt} = -R_f i_L - v_c + v_f \\ C_f \frac{dv_c}{dt} = i_L - i_{load} \end{cases} \tag{2.3}$$

A Linear Time-Invariant (LTI) dynamic system, such as this one, can be described in a matrix form using state space modelling. Using the state space scheme higher order differential equations can be written as first order differential equations, although first order ones are enough to model an LC filter. The general scheme is shown in Equation 2.4.

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t) \end{cases} \tag{2.4}$$

The subject system is already an LTI system: it is a linear system because a linear combination of its inputs gives the same outputs as the linear combination of the individual responses to those inputs, i.e. the outputs and inputs are linearly related. Therefore, changing the inputs in a linear way will change the outputs in the same linear way. It is also time invariant because its output does not depend on when the input was applied [13], i.e. the matrix terms **A**, **B**, **C**, and **D** that relate inputs and outputs are constant.

In this case, the input vector to the system $\mathbf{u}(t)$ is the consequence of the switching between states $\vec{v}_0, ...\vec{v}_7$, which is the voltage drop across the filter v_f and the current through the load i_{load}. These two variables are therefore the inputs to the second-order system. The behavior of the inverter is considered ideal in the model, so that it does not add any more dynamics to the system. Therefore, the complete state-space model of the system is considered to be simply that of the filter.

Since inductor current i_L and capacitor voltage v_c evolve through time in a way that depends on the values that they have at that time and on the externally imposed values of input variables [13], they are chosen as the two states of the system $\mathbf{x}(t)$ and thus define the dynamics of the system. The continuous state-space model for this system is depicted in Equations 2.5 and 2.6.

$$\begin{bmatrix} \frac{di_L(t)}{dt} \\ \frac{dv_c(t)}{dt} \end{bmatrix} = \mathbf{A} \begin{bmatrix} i_L(t) \\ v_c(t) \end{bmatrix} + \mathbf{B} \begin{bmatrix} v_f(t) \\ i_{load}(t) \end{bmatrix} \tag{2.5}$$

$$v_{load}(t) = \mathbf{C} \begin{bmatrix} i_L(t) \\ v_c(t) \end{bmatrix} + \mathbf{D} \begin{bmatrix} v_f(t) \\ i_{load}(t) \end{bmatrix} \tag{2.6}$$

Where the coefficient matrices of the state-space model are:

$$\mathbf{A} = \begin{bmatrix} -\frac{R_f}{L_f} & -\frac{1}{L_f} \\ \frac{1}{C_f} & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & -\frac{1}{C_f} \end{bmatrix} \tag{2.7}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{2.8}$$

The goal of the state space model is to simulate the behavior of the physical system in order to find the rate of change of the state variables $\dot{\mathbf{x}}(t)$. Given an input signal the model of the LC-filter can be used to predict system behaviour through updating the state values according to the obtained derivatives and derive also the output signal.

An LTI system can be described by a single function called its impulse response. The transfer function of a system is the Laplace transform of the impulse response, and therefore can also describe the system. This transformation changes the function from the time domain to the frequency domain and it is useful because it turns differential equations into algebraic equations, e.g. it turns convolution into multiplication. This way, in the frequency domain, the output is the product of the transfer function with the transformed input [13] without requiring initial condition values. The Laplace transform of the state space model is shown in Equations 2.9 and 2.10.

$$\mathcal{L}\{\dot{\mathbf{x}}(t)\} = \mathcal{L}\{\mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)\} \tag{2.9}$$

$$\begin{cases} s\mathbf{X}(s) = \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathbf{U}(s) \\ Y(s) = \mathbf{C}\mathbf{X}(s) + \mathbf{D}\mathbf{U}(s) \end{cases} \tag{2.10}$$

To be able to implement the state space model in a digital system, it has to be transformed from the continuous S-domain to the discrete Z-domain. The Z-transform is the discrete equivalent of the Laplace transform. To convert the model to discrete frequency, i.e. to map from s to z, the forward Euler method can be used. It follows the substitution described in Equation 2.11, where T is the sampling period and is chosen to be $T_s = 20$ µs. The method used in this project to discretize the state-space model makes use of the *c2d* function of *Matlab*, which numerically computes the exact transformation.

$$s \leftarrow \frac{z-1}{T} \tag{2.11}$$

Then, to map to discrete time domain to see the system's impulse response, Equation 2.12 is used.

$$M(z) = E(z) \implies m(k) = e(k) \tag{2.12}$$

This allows the model to be described in discrete time through Equation 2.13, with its own new **A** and **B** matrices that are discretized to **G** and **H**, and are shown in Equations 2.14 and 2.15.

$$\begin{bmatrix} i_L((k+1)T) \\ v_c((k+1)T) \end{bmatrix} = \mathbf{G}(T) \begin{bmatrix} i_L(kT) \\ v_c(kT) \end{bmatrix} + \mathbf{H}(T) \begin{bmatrix} v_f(kT) \\ i_{load}(kT) \end{bmatrix} \tag{2.13}$$

$$\mathbf{G}(T) = e^{\mathbf{A}T} \tag{2.14}$$

$$\mathbf{H}(T) = \mathbf{A}^{-1} \left(e^{\mathbf{A}T} - \mathbf{I} \right) \mathbf{B} \tag{2.15}$$

Once **H** and **G** are substituted, Equation 2.11 can be directly used to update the values of the states of the model at any given time, the only requirement is that the values of both states and inputs from the previous sampling step are known. These equations will be used for prediction in FS-MPC.

2.3 FS-MPC

Model Predictive Control is an advanced Closed-Loop control algorithm that makes use of the system states prediction to take the control decision that results in the system tracking its references as well as possible. It is therefore flexible in its decision process, as it can include any state or variable of the system and decide based on linear or non-linear behaviour. This broadens the control possibilities and is one of the main reasons why it is gaining more and more interest in the field of power electronics.

In this case, MPC is used to control a 2-level three-phase power converter, and since the possible control actions are finite in a power converter, i.e. the control decision is reduced to just the choice of one of the 8 possible switching sates, the MPC used is called Finite Set (FS-MPC) [14].

Predictions are obtained for each possible control action in the power converter through the use of a model of the system, often formulated in the state-space. These future state predictions of the system are derived in real time. Then, the performance of each possible control action is compared by calculating for each of them the value of a previously defined cost function. The control action chosen will be the one with minimum cost function value and will then be applied in the next sampling instance. It is important to point out that this control scheme does not need a modulation technique afterwards, because it already obtains the instantaneous switching commands.

In FS-MPC with a one-step horizon the decision of the algorithm is based only on the one step ahead state value prediction. When making predictions for more than a single step ahead, the number of calculations to be performed online increases exponentially. For the converter addressed in this project, if an MPC algorithm for two steps ahead is implemented, eight predictions must be calculated in each step, resulting in evaluating the state-space model of the converter and the cost function a total of 64 times for each sampling instance. In this way, the total number of model evaluations follows an exponential law of the form n^h, where n is the number of possible states of the system and h the prediction horizon, or number of steps ahead to be evaluated. The MPC execution time cannot be longer than the sampling time because that would mean that when the state switching has to be applied it is still calculating the optimum state. This limits the number of steps ahead horizon that can be implemented in real-time applications.

Moreover, more complex converter topologies have more possible control actions, which further contributes to the exponential number of model evaluations when trying to implement MPC with a longer prediction horizon. In this way, the fact that the model must be run online many times, together with the small time between control actions, i.e. a converter sampling time in the order of µseconds, entails an inherent limitation in the feasibility of MPC for these topologies.

2.3.1 FS-MPC working principles for one step prediction

Synchronized switching and sampling instants play a big role in the working principle of MPC, since the algorithm is normally implemented in digital control platforms [15], such as *dSpace*.

In the figures along this section the set of input variables that are measured are named $x_i(t_{k+h})$, where i are the switching states for which the variables are calculated, h is the time step, and k is the current sampling instant. $\hat{x}_i(t_{k+h})$ uses the same nomenclature but refers to the predicted input variables calculated using

the discrete state-space model. A diagram of the calculation steps that one-step FS-MPC follows for each time step is depicted in Figure 2.6.

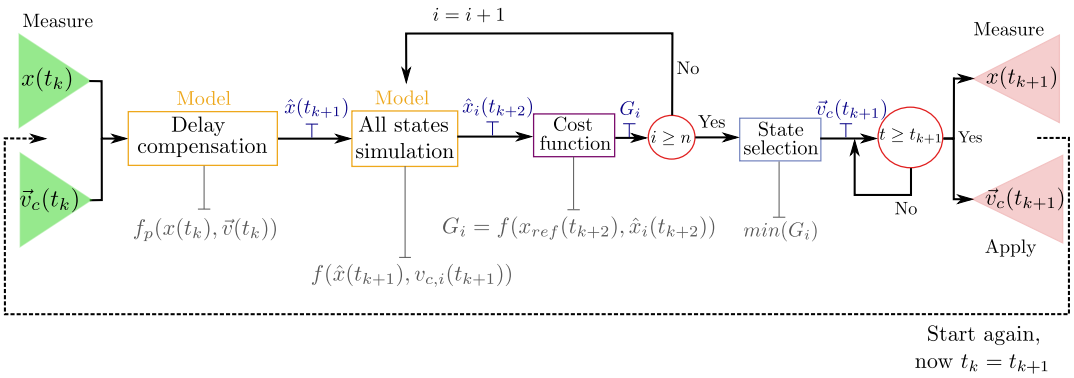

Figure 2.6: Information flow along one step time calculation for FS-MPC with one step horizon prediction. Picture based on [14].

The basic operation principle starts at the beginning of every sampling period by receiving the new measurements (from time t_k) for the three phases of the filter currents ($i_{L,a}, i_{L,b}$ and $i_{L,c}$) and the capacitor voltages ($v_{c,a}, v_{c,b}$ and $v_{c,c}$), which are the previous values of the states in the state-space model. It is also needed to measure the load currents ($i_{load,a}, i_{load,b}$ and $i_{load,c}$) and to know the previously applied voltage to the filter ($v_{f,a}, v_{f,b}$ and $v_{f,c}$) which is determined by the previous switching state chosen. The output voltage of the converter is also named voltage input vector as it is the control input defined by the switching state. Load currents and filter voltages form the inputs of the system in the state-space model, despite the load current being a measured value.

Since the system is balanced, and therefore its zero component is null, the three phase variables of each component can be transformed to the $\alpha\beta$ reference frame. In this way, each vector that was formed previously by three components a, b, c, now only has two components $\alpha\beta$. Thereby the total number of inputs and states of the system is reduced from 12 ($\vec{i}_{L,abc}$, $\vec{v}_{c,abc}$, $\vec{i}_{load,abc}$ and $\vec{v}_{f,abc}$) to 8 ($\vec{i}_L = i_{L,\alpha\beta}$, $\vec{v}_c = v_{c,\alpha\beta}$, $\vec{i}_{load} = i_{load,\alpha\beta}$, and $\vec{v}_f = v_{f,\alpha\beta}$). All these measurements are transformed from the *abc* reference frame to $\alpha\beta$ through vector projection to the real axis (α) and the imaginary axis (β).

Once the algorithm receives all the measurements from the previous sampling time in $\alpha\beta$, it makes use of the state-space model to predict the values of the states in the next sampling time, i.e. it applies the differential equations of the LC filter model to calculate the next values of the filter currents and capacitor voltages. This is needed since it is unfeasible in a digital system to read measurements,

perform calculations and apply commands in the same sampling time, due to the time needed for computation and because signals can only be updated at the beginning of a sampling period. This model prediction of $x(t_{k+1})$ is called delay compensation and gives the predicted value of those variables [16], named $\hat{x}(t_{k+1})$. It is important to take into account that $\vec{v}_c$ is the voltage vector due to the switching state chosen for the current sampling instant t_k, i.e. in Figure 2.7 this vector would be the voltage vector due to the selection of the state in t_k that directs the system from $x_5(t_k)$ to $x_2(t_{k+1})$.

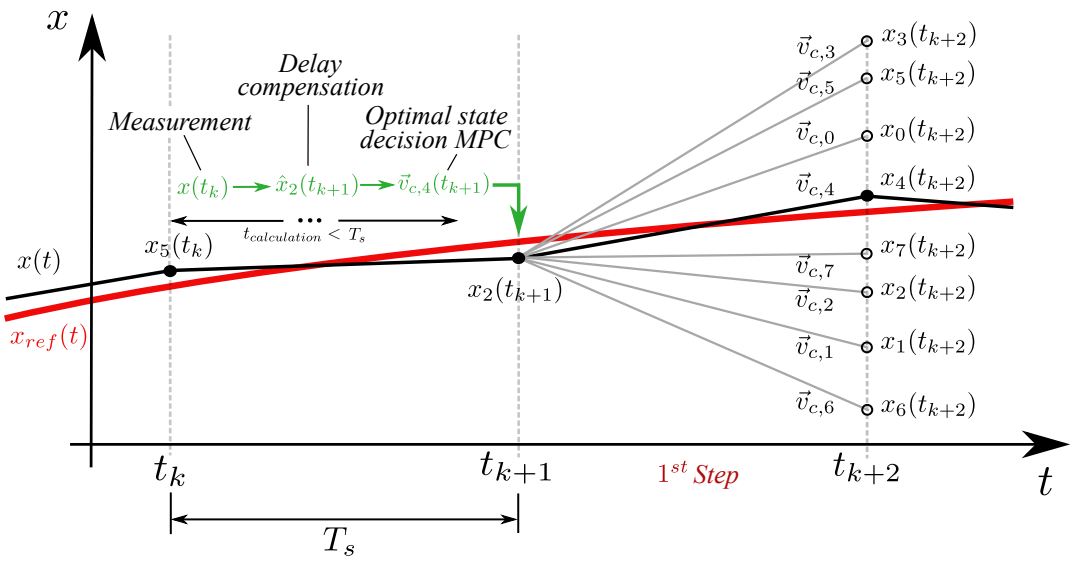

Figure 2.7: Time based representation of the FS-MPC algorithm performance. Picture based on [14].

Once the predicted values for $t(k+1)$ are obtained (delay compensation is done), it is time to apply again the state-space model with these predicted states and the new inputs, in order to predict again for the next sampling time $t(k+2)$. In this case, in a loop for all the possible switching states i, i.e. all the possible voltage vectors $v_{c,i}(t_{k+2})$), the model is applied to calculate the next values of the states. Since both zero vectors give null voltage, only the vector zero with all upper switches turned off is considered, and the 7^{th} vector is not. In this way, the number of times that the model has to be applied for each step is reduced from 8 to 7 times per step.

This simplification reduces computational load at the expense of larger switching losses.

2.3.2 Cost function

The basic control objective of the algorithm, or optimization problem to be solved, is to select the proper switching state that generates a voltage vector ($\vec{v}_c(t_{k+2})$) so that the phase voltage tracks the reference voltage with minimal error. The reference voltage signals are given at t_k for $\vec{v}_{ref}(t_{k+2})$ in the $\alpha\beta$ reference frame.

Each of these 7 predicted values are evaluated through the use of a cost function and the voltage vector with the minimum error between states and its references is selected and generated in t_{k+1}. As an example, in Figure 2.7 the voltage vector with minimum cost would be $\vec{v}_{c,4}$.

The cost function can penalize any terms as desired, and should be constructed in a way that results in the MPC outputs that best fit the requirements of the application for which the algorithm is to be designed.

The most basic cost function would be formed by the euclidean distance between the phase voltage and the reference one, as shown in Equation 2.16. In this way, big differences between the desired and the obtained voltages are penalized more and small differences have a decreased cost.

$$G = \left(\vec{v}_{ref,\alpha} - \vec{v}_{c,\alpha}\right)^2 + \left(\vec{v}_{ref,\beta} - \vec{v}_{c,\beta}\right)^2 \tag{2.16}$$

As a secondary objective, the derivative term of the load voltage is included in order to improve the performance of the algorithm. This is again penalized by minimizing the euclidean distance between the derivative of the voltage reference and the derivative of the load voltage. The term is added to the cost function with a weighting factor λ [16], as shown in equation 2.17.

$$G = \left(\vec{v}_{ref,\alpha} - \vec{v}_{c,\alpha}\right)^2 + \left(\vec{v}_{ref,\beta} - \vec{v}_{c,\beta}\right)^2 + \lambda G_d \tag{2.17}$$

This results in very improved performance of the FS-MPC for one step horizon; because the switching state chosen could be the best one for the sampling instant but have a large derivative at that point, and therefore lead to a worse value in the next sampling instant. This phenomenon is illustrated in Figure 2.8. Considering sinusoidal reference voltages:

$$\vec{v}_{ref,\alpha\beta} = v_{ref,\alpha} + jv_{ref,\beta} = v_{ref}\sin(\omega_{ref}t) + jv_{ref}\cos(\omega_{ref}t) \tag{2.18}$$

The derivative of the reference voltage is shown in Equation 2.19.

$$\frac{d\vec{v}_{ref}}{dt} = \omega_{ref}v_{ref}\cos(\omega_{ref}t) - j\omega_{ref}v_{ref}\cos(\omega_{ref}t) = \omega_{ref}v_{ref,\beta} - j\omega_{ref}v_{ref,\alpha} \tag{2.19}$$

Then, it is compared to the derivative of the load voltage, which is calculated from the voltage equation of the LC-filter.

The cost function term accounting for this derivative term results in Equation 2.20. This equation can then be simplified to Equation 2.21. Although G'_d is not equivalent to G_d, the two are directly proportional: as the term is weighted by an arbitrary λ, both terms are interchangeable.

$$G'_d = \left(\frac{dv_{ref,\alpha}}{dt} - \frac{dv_{c,\alpha}}{dt}\right)^2 + \left(\frac{dv_{ref,\beta}}{dt} - \frac{dv_{c,\beta}}{dt}\right)^2 \tag{2.20}$$

$$G_d = \left(C_f \omega_{ref} v_{ref,\beta} - i_{f\alpha} + i_{load,\alpha}\right)^2 + \left(C_f \omega_{ref} v_{ref,\alpha} + i_{f\beta} - i_{load,\beta}\right)^2 \tag{2.21}$$

The addition of the derivative term to the cost function does not modify the behavior of multiple-step FS-MPC nearly as much as it does for the single-step algorithm. This is due to the fact that undesirable voltage derivatives are already penalized by considering the future behavior of the converter in the following steps. In the rare case where resonance could occur in-between sampling periods, the derivative term would still mitigate such behavior in multi-step FS-MPC. This is illustrated in Figure 2.8.

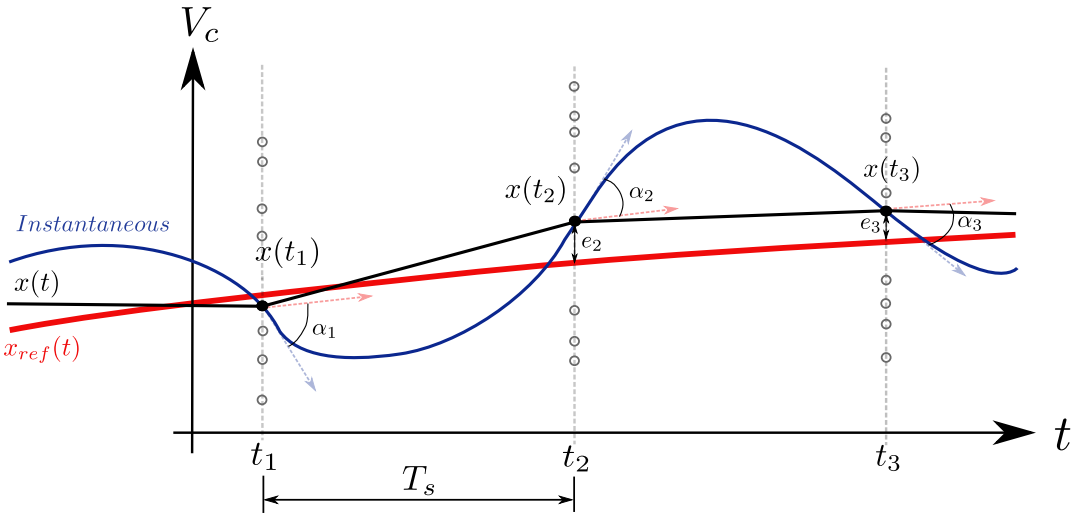

Figure 2.8: Representation of the possible scenario in which the load voltage is accurate in the sampling instants but the large derivatives at those points create big differences during the sampling time. α_i represents the difference between the reference voltage derivative and the load voltage. e_i are the error values at the sampling instants. Picture based on [16].

The selection of the weighting factor between the two terms of the cost function

is always a trade-off decision. In our case, in order to select the weighting factor that provides the better performance, measured using the lowest THD value in the phase voltages, a test with different weighting factors in the FS-MPC model was performed in simulation. The results are shown in Figure 2.9 and $\lambda = 1$ is chosen as the optimum one.

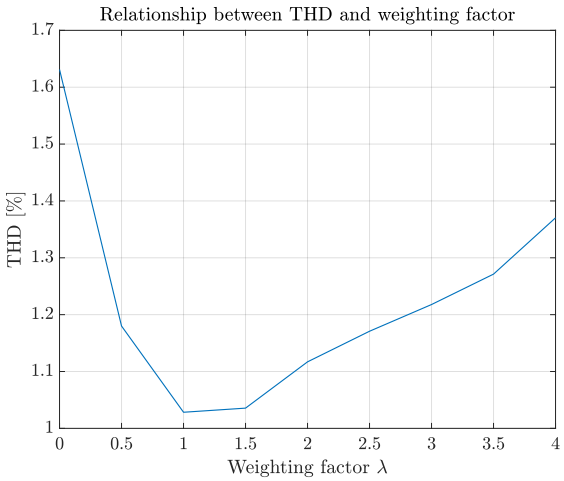

Figure 2.9: Different weighting factor in the cost function and corresponding phase voltage THD values.

Additionally, switching losses can be minimized by modifying the cost function in such a way that favors the current switching state if the control error is not significantly large, although this was not implemented in this project.

As a safety measure there is limitation set against over-current. If the current space vector reaches a maximum of 30 A, the cost function value automatically becomes infinite and thereby the switching state that generates over-current is never going to be selected.

The FS-MPC procedure described above would be quite similar for FS-MPC algorithms using multiple steps ahead prediction. As mentioned before, for multiple steps the algorithm has to predict the future values as many times as steps ahead considered, with the consequent exponential times the discrete model has to be run. Apart from this, the differences lie in the cost function structure and in how the reference voltages are calculated.

Firstly, the cost function should be extended to include the same metrics as before, but for each of the steps ahead considered. Thereby, e.g. if the consequent voltage differs much in the further steps by choosing the best voltage for the first step, that

option will have a higher cost and may be discarded. Consequently, the voltage vector with lowest cost that would be finally applied would be the one that has a slightly poorer performance in the first step but is going to give better average results for all the steps considered.

Secondly, the reference voltages for further steps used in the cost function are derived using Lagrange interpolation. This reduces the computational cost and in case that the amplitude and phase of the reference voltages was not known explicitly, accurate predictions could be obtained without needing to estimate these parameters.

To apply Lagrange interpolation, the values of the three previous reference voltages $\vec{v}_{ref}(t_k)$, $\vec{v}_{ref}(t_{k+1})$ and $\vec{v}_{ref}(t_{k+2})$ are required, in order to derive the parabola that fits them and the next points. For FS-MPC with two steps ahead prediction only one value $\vec{v}_{ref}(t_{k+3})$ is derived, while for three steps it would be this one and the next one $\vec{v}_{ref}(t_{k+4})$, and so on for cases with further ahead predictions. The approximations obtained for two and three steps ahead FS-MPC are derived in Chapter B.

2.3.3 FS-MPC advantages and drawbacks

The main advantages of FS-MPC are its robustness, excellent transient characteristics, and the possibilities it offers to account for non-linearities and constraints [16]. By allowing its cost function to be designed as desired, the algorithm is extremely flexible and can include arbitrarily many control objectives. It can also greatly simplify the control of multivariable systems, as FS-MPC can be designed to output as many values as required.

On the other hand, its main disadvantage is the large computational burden it places on the hardware. FS-MPC can then be impossible to implement in real time for complex systems such as multilevel converters, especially for algorithms with prediction windows of several steps. This also limits the sampling frequencies at which the converter can operate. Since FS-MPC outputs the control actions directly and does not make use of any modulation techniques, the smaller the sampling time of the system, the better its performance. This is the main trade-off that limits the broader implementation of FS-MPC algorithms in power electronics. Another disadvantage of FS-MPC is that the algorithm heavily relies on the accuracy of the model it uses to make predictions. If the model is inaccurate, the algorithm will result in poor performance.

One approach to minimize the drawbacks of FS-MPC introduced by its compu-

tational requirements is to solve the optimization problem offline [14]. Such an approach is taken in this project, using data obtained by offline simulations of the FS-MPC described in this chapter to train an artificial neural network-based imitator controller.

Chapter 3

Imitation learning method

One of the biggest disadvantages of the MPC technique is the exponential increase of computational burden with the rise of steps ahead prediction of the voltage vector. This problem makes the control technique unsuitable for multiple prediction layers. An approach to solving this issue with the use of machine learning (ML) techniques is shown in this report.

This section covers the explanation of imitation learning and the neural network proposed for improving the MPC performance with a longer prediction window.

3.1 Machine Learning

One of the uses of machine learning is to obtain the model of a system from recorded data. This method offers advantages over other conventional techniques: for instance, the properties of the model to simulate may not be explicitly required, since the training is performed only with some inputs (features) and outputs (labels). With this data, it is possible to construct a model such that it responds correctly to new unknown scenarios. The machine learning algorithm can also be much more efficient than a conventional model.

3.1.1 Neural networks

Artificial neural networks (ANN) can form the basic structural layout of a machine learning (ML) model. They consist on a series of nodes called neurons that are dis-

posed in different layers. Each neuron contains an activation function that outputs a combination of the neurons from the previous layer.

The first layer of an ANN is called the input layer and it is associated with the features of the system, similarly, the last layer is called the output layer and it mirrors its labels. The layers in-between are called the hidden or middle layers. Neurons include activation functions that receive information from the previous layer and define the output of the node. More information on the structure of artificial neural networks can be found in Appendix A.

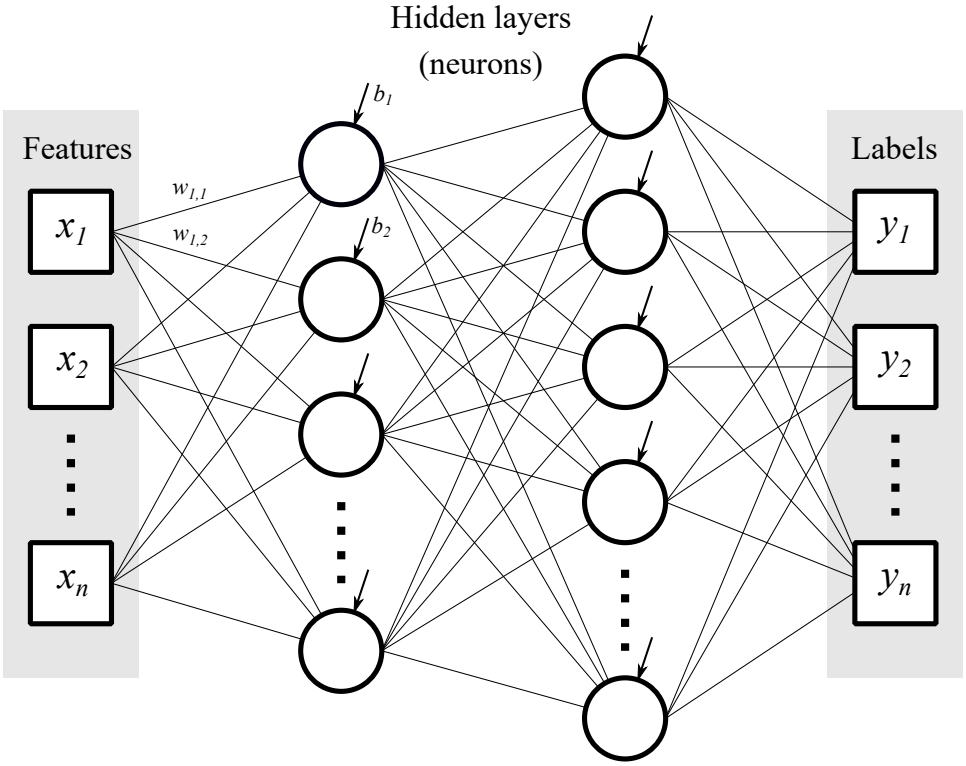

Figure 3.1: Artificial neural network diagram.

The non-linear activation function is fed with information from all the neurons in the previous layer multiplied by their corresponding weights, and a bias that is also added to its output. These weights and biases are calibrated during the training process. In order to predict complex models, non-linear activation functions need to be introduced. One of the most efficient is the rectified linear unit (ReLU), consisting in a linear output for positive inputs and zero for negative inputs, as shown in equation 3.1.

$$ReLU(x) = \begin{cases} 0 & \forall x \leq 0 \\ x & \forall x > 0 \end{cases} \tag{3.1}$$

The training process of the ANN requires some features and their corresponding labels. An algorithm called back-propagation allows gradient calculation on ANNs; its principle relies on iteratively applying the chain rule of derivatives from the output to the first layers. In order to do this, initial values for weights and biases are set and the ANN is fed with some features. The error between the correct labels and the outputs obtained by the network are calculated and the weights of all the connections to the neurons in the previous layer are adjusted according to these errors. These adjustments result in new outputs of the activation functions and new errors can be obtained continuing the sequence until all the parameters have been adjusted. The rest of the training data is then fed to the network in a similar way.

More information on how artificial neural networks process data and on how they can be trained is presented in Appendix A.

If the training data is too disperse in the data space or if the network has too many connections compared to the amount of training data, there is a risk of resulting in a model that is very accurate with this data but that is not good at extrapolating to other data. This common problem is called over-fitting.

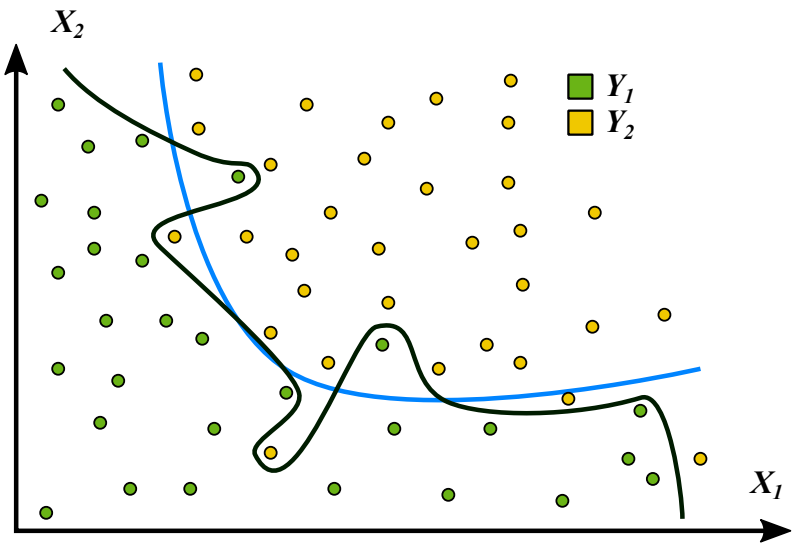

Figure 3.2: Example of overfitting. In blue: curve correctly fitted. In red: overfitted curve.

In order to check whether the network is accurate enough, the training data points cannot be used because the network could be generalizing inaccurately, leading

to wrong labels when fed with new data. For this reason, different points have to be used for testing if the model is correct. This data is usually split again in two; validation and test. Validation is used while training the model to maximize accuracy which may still lead to a model that is not generalizing correctly for other data outside the validation set. Test is then convenient to obtain the final accuracy of the model, the test data is only used at the end of the training process, once a final network has been trained.

3.1.2 Imitation Learning

Imitation learning is an artificial intelligence technique that aims to make a model take the correct decision when unknown inputs are fed to it. In order to do so, it is trained with many examples, each one of them containing multiple features and their corresponding labels. This kind of behaviour is similar to how humans learn new tasks. The use of Imitation learning in the field of power electronics has previously been covered in different studies [9] [8].

The goal of the current project is to imitate the behaviour of the FS-MPC algorithm with simpler operations, making the new model more computationally efficient while responding accurately to incoming events. This can be achieved since the computational cost of the ANN depends only on its topology. Unlike an ANN imitator, the computational cost of FS-MPC algorithms increases exponentially with the number of steps forward to perform predictions on.

The ANN is therefore imitating the FS-MPC algorithm and thus it is not expected to ever perform better than the original FS-MPC. The ANN creates labeled zones in a multidimensional map of features. This generalization may indeed cause a few wrong responses but it might also be able to react more accurately to unexpected events. For these outlier events that the MPC might not be designed for, thus responding incorrectly, the ANN would just find the new data in the multidimensional data space and choose the state that is most dominant in that area.

3.2 ANN training

3.2.1 Data generation

The best possible data to be used for training is obtained in real case scenarios, or in the least, through laboratory experimentation. However, the amount of data

points needed for training makes it impossible to harvest the data for the current project in this way, since there has been no access to any real applications where these MPC models had already been running for some time. Training the model in a laboratory would also entail high risk of damaging equipment because in order to get the necessary data points all cases need to be evaluated, taking the hardware to its physical limit.

It was then decided to obtain the data points by running simulations of the MPC models, which had previously been experimentally validated. In addition to avoiding putting the hardware at risk, training with simulated data allows to get as many points as considered necessary and also to get them evenly distributed through the features data space.

Moreover, as mentioned before, controllers than cannot be executed in real time can be simulated offline and their data used to train an ANN. This procedure is shown in this project for 3-step FS-MPC.

The features of the data points consist on load resistance, reference and measured voltages and measured current at the load. Voltage and current data are given in the alpha-beta reference frame. Also, the previous selected state is fed back as another feature. These add up to a total of 8 features. Of these, only the reference voltage (v_{ref}) is an external input, while all the others are measurements or values fed from sensors. The labels are the 7 possible output states. Each data point has the correct label set to 1 and all the others to 0. As it is explained in chapter 2, the inverter has 8 feasible states, two of which (states 0 and 7) both output 0 V in all the phases, however, to avoid switching losses, it is more convenient to use one over the other depending on the previous state. Nevertheless, switching losses have not been considered in the cost function and only the state 0 has been considered as possible label.

With the intention of obtaining evenly distributed points for the training process, the data includes all the possible permutations. The values used for training contain all combinations of the load current, with alpha-beta values between -16 A and 16 A, the resistance with values from 30 Ω to 60 Ω and also alpha and beta values of output voltage error ($v_{ref} - v_{meas}$) between -5 V and 5 V. It is not required to have all the values for v_{meas} because it directly depends on the load current and the resistance value, however, the error is considered to adjust for small perturbances or transients. Notice that v_{meas} is the value taken at the input of the load, referred as v_c during Chapter 2.

Finally, since the v_{ref} values were external inputs to the system, and thus controlled parameters, only combinations of the alpha-beta components resulting from a si-

nusoidal function are considered. Time values for one wave period are fed to the functions. Alpha and beta components are then obtained with these times, with an amplitude of 325 V and a frequency of 50 Hz.

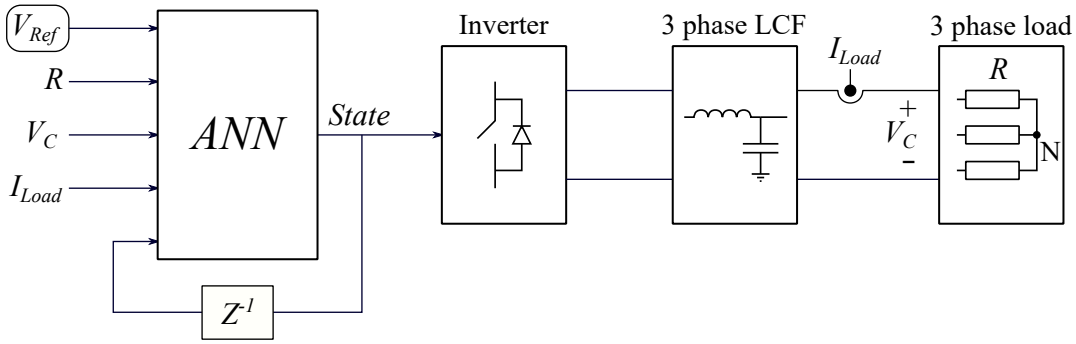

Figure 3.3: Features sources of the ANN.

These data points make up an 8-dimensional matrix that contains several million data points. These points are evenly distributed in the data space. However, as explained in section 3.1.1, some data is needed for testing the model accuracy. If the testing and validation data were taken from the same grid, it would leave some holes wherever data is not used for training. This should not affect the training much but it is easily avoidable by selecting different testing data. In order to do so, random points were taken from the data space to be used for validation and testing.

3.2.2 Network configuration

In order to decide the most optimal network layout, computational cost and performance were considered. The optimization of the ANN has been performed using data from the 1 step ahead FS-MPC model. It is assumed that a similar layout is also convenient for the 2 and 3 steps ahead FS-MPC models.

The adequate accuracy needs to be established at a certain threshold. The computational cost of the network depends, among other things, on the amount of connections existent. It was found that adding more than one hidden layer to the network does not significantly increase the accuracy of the model. For this reason, it was decided to create a network with only one hidden layer.

To optimise the amount of neurons in that hidden layer, validation error and total harmonic distortion (THD) of the designed model were considered. To do so, the model was trained with different neurons in the hidden layer. The resultant networks were simulated to obtain the performance of the sine wave and get the

THD.

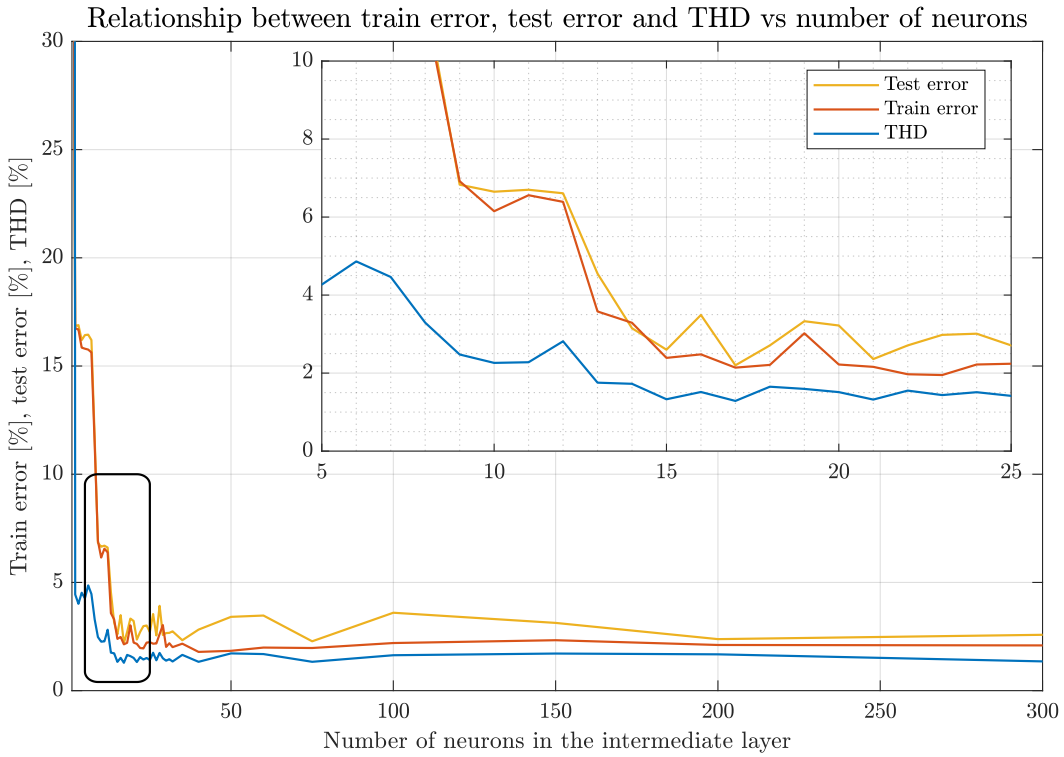

Figure 3.4: Train and test error and THD against number of neurons in hidden layer.

Figure 3.4 shows how the error and the THD become almost constant with an increasing number of neurons, for 1 step FS-MPC. It was decided that less than 2.5% validation error was sufficient. The minimum number of neurons in the middle layer that make the network fulfill this constrain is then 15, with this setup, an average THD of 1.329% is obtained. In Figure 3.4 it is also seen how the validation error and the THD flatten out and stop approaching zero with more than 15 neurons in the middle layer. After comparing several different optimizers, the ANN was trained with the *Adam* algorithm. This optimizer is explained in-depth in Appendix C.

In order to obtain more detailed information about the neural network performance a confusion matrix is obtained. Confusion matrices allow the recognition of errors more precisely. They face the actual label that should have been chosen with the predicted one, this way, the amount of correct decisions are shown in the main diagonal. The wrong decisions are placed in the rest of the cells and they show accurate information about them, this is because it is not only possible to see where does the network fail most but also what decision it takes instead of the

correct one.

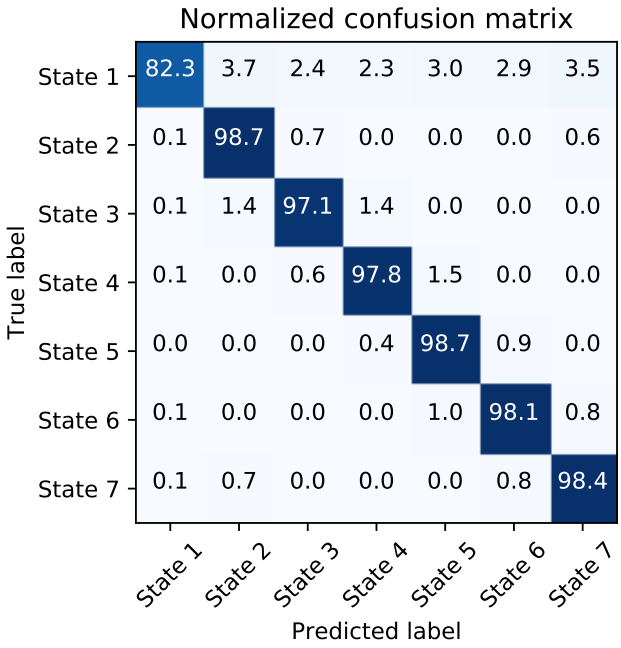

Figure 3.5: Normalized confusion matrix for 15 neurons in the middle layer. Average test accuracy of 98.05%

The tested network contains 15 neurons in the middle layer and it is trained with data from the 1 step prediction MPC model. Its confusion matrix, shown in Figure 3.5, shows a balanced error distribution where most of the wrong decisions for the active states lay in the neighbor states. The State 1 $[0, 0, 0]$ has a much lower accuracy than all the other states. This issue comes from the training data characteristics, since the training data was obtained with a fixed v_{ref} amplitude of $325V$ (corresponding to $v_{pp} = 650$ V) while the DC supply voltage is fixed to $700V$. This means that the three-phase voltage space vector is following a constant radius circle ($r = v_{ref}$) that is close to the maximum possible, as shown in Figure 2.3. This entails that there is a far lower amount of labels corresponding to State 1 among the training data than to other states, in fact, only 0.62% of the total labels correspond to State 1. The network is then not able to learn these decisions as accurately as the other states.

The ANNs for the 2 and 3 steps ahead MPC models were obtained in a similar fashion, the accuracies of those networks have been 97.1% and 97.57% respectively.

Chapter 4

Implementation and results

Validating the model through laboratory experiments is fundamental during an investigation process. Simulation is usually performed beforehand and it is a valid way for model troubleshooting and optimization. However, real case scenarios sometimes have unexpected performances that lead to unpredictable behaviour of the implemented model.

In this chapter, experimental validations of the ANN controllers are performed. The gathered data is compared to simulation results as well as to the equivalent FS-MPC controllers. The ANN models that have been tested during experimentation have been trained with 1, 2 and 3 steps ahead FS-MPC data, all of them contain one middle layer with 15 neurons. They are compared to the equivalent FS-MPC model, however, the 3 steps ahead prediction FS-MPC was not possible to implement since the computation time was higher than the sampling time.

4.1 Laboratory setup

The experimental work was carried out in a laboratory of the Energy Technology Department at Aalborg University. An electrical diagram of the setup is shown in Figure 4.2 and a picture of it is also presented in Figure 4.1.

The hardware consists on the devices shown in Figure 4.1. The model is processed through the *dSPACE* software and fed to the MicroLabBox DS1202. The inverter drivers receive a 15 V signal and the output of the MicroLabBox contain low voltage signals. To step up the voltage, a level shifter is installed. After the three-phase VSI (*Figure 2.1*) and the LC filter, a manual switch decides the output load, it can

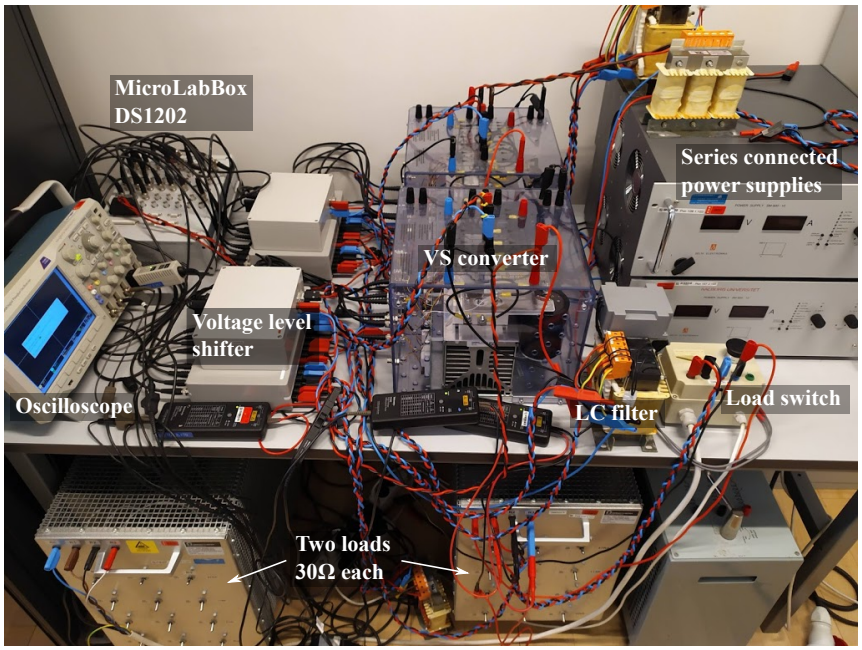

Figure 4.1: Laboratory setup.

vary the load resistance from 30 Ω to 60 Ω.

Experimental results are obtained from a four channel oscilloscope where the three line-to-line voltages (V_{ll}) and one phase current (I_{ph}) are displayed. On top of this, the dSPACE software measures data such as switching frequency, execution time or V_{DC}.

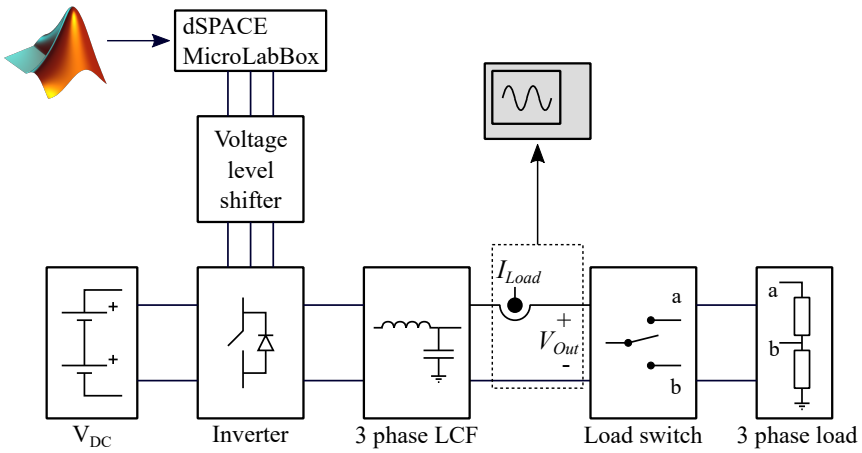

Figure 4.2: Electrical diagram of the laboratory setup.

4.2 Model validation

In order to ensure that the laboratory tests were performed correctly, a comparison between simulation and laboratory tests was carried out. The MPC and ANN models for 1 step prediction were tested and compared with simulations in steady state. The characteristics of the ANN imitator method that is used for the simulation and laboratory results are described in 3. Figures 4.3 and 4.4 compare one voltage phase during simulation and tests, for equal settings of $v_{ref} = 325$ V and $R_{load} = 60\ \Omega$. v_{ref} is the reference amplitude of all three phase voltages, and the load resistance is balanced and star-connected to the output of the low-pass filter. Throughout the experiments, THD value and switching frequencies were collected and are presented in Table 4.1.

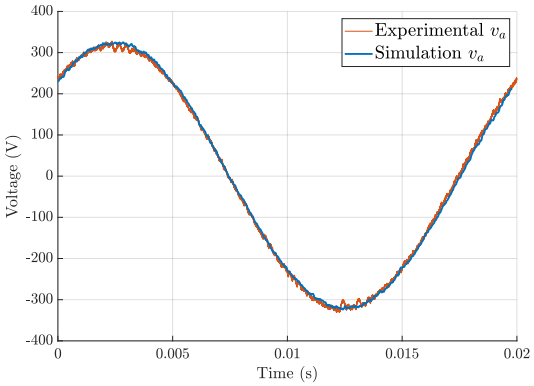

Figure 4.3: Steady-state performance for MPC algorithm for 1 step prediction, laboratory and simulation results for one voltage phase signal.

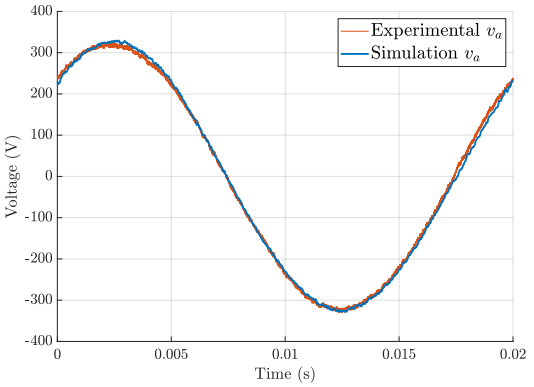

Figure 4.4: Steady-state performance for ANN method for 1 step prediction, laboratory and simulation results for one voltage phase signal.

The results show a satisfactory emulation of the MPC algorithm by the ANN method despite of showing a slight difference in the results. The variation between MPC and ANN simulation is 0.29% in the THD value and around 0.7% in switching frequency, which shows that the ANN was trained correctly and performs as well as the MPC. Between simulation and laboratory for both MPC and ANN the results validate the simulation because there is just a slight reduction in the switching frequency and variation in the THD value, but they are considered to come from assumptions and simplifications between laboratory and simulation.

Controller	Type	THD [%]	$f_{sw}[Hz]$
MPC	Simulation	1.075	8680
	Laboratory	1.654	8150
ANN	Simulation	1.364	8620
	Laboratory	1.356	8200

Table 4.1: Comparison between simulation and laboratory for both MPC and ANN method.

Since the model is assumed to be validated and is able to reproduce laboratory tests, the results shown in further sections will correspond to laboratory results, which are considered to be even more representative of the real performance of the algorithm.

4.3 Steady state performance

This section contains simulation and laboratory tests for steady state performance comparing prediction schemes for different steps for the FS-MPC algorithm and ANN method. The models used are FS-MPC with one and two steps horizon, shown in Figures 4.5 and 4.7. Results obtained from the ANN imitators for one, two and three steps horizon prediction can be seen in Figures 4.6, 4.8 and 4.9. The MPC algorithm for three steps ahead prediction could not be tested in the laboratory since its execution time exceeded the sampling time $T_s = 20$ µs and therefore the control scheme cannot be implemented in real time.

The tests were all performed with $V_{ref} = 325V$ and $R_{load} = 60\ \Omega$.

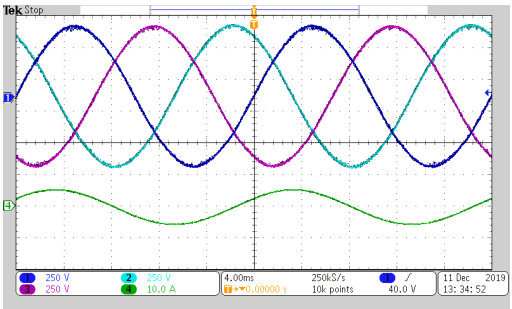

Figure 4.5: Steady state performance of MPC with one step ahead prediction.

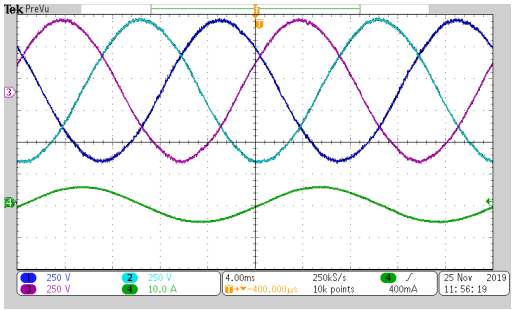

Figure 4.6: Steady state performance of ANN imitator method with one step ahead prediction.

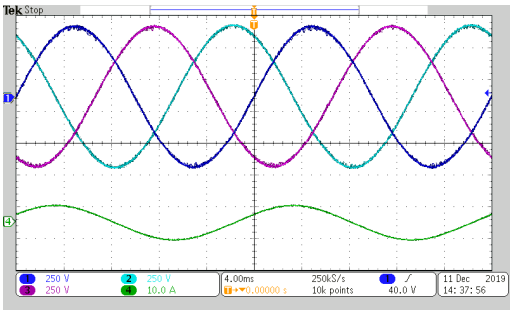

Figure 4.7: Steady state performance of MPC with two steps ahead prediction.

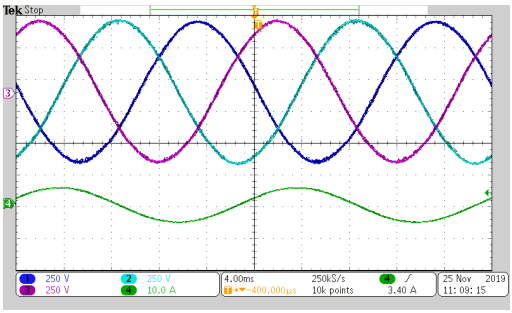

Figure 4.8: Steady state performance of ANN imitator method with two steps ahead prediction.

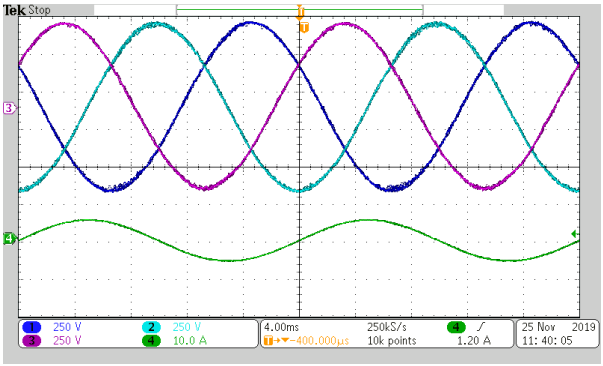

Figure 4.9: Steady state performance of ANN imitator method with three steps ahead prediction.

Table 4.2 shows the differences in performance between the models evaluated. It can be seen how the ANN models have equal execution time regardless of the increment in prediction horizon. MPC, however, has an execution time that increases exponentially with prediction horizon, such that for three steps the algorithm cannot be evaluated in real time and for two steps it takes a longer time to process than the equivalent ANN. This effect is one of the main advantages of the imitation learning model, since the execution time is expected to remain constant with even more steps ahead. Furthermore, for more complex converter topologies the execution time of their models could be largely reduced using the same method (e.g. multilevel converters).

The THD value and switching frequency presents similar performance in both methods and for different steps ahead. The ANN model starts having a better performance with 2-step ahead prediction.

Steps	Controller	THD [%]	$f_{sw}[Hz]$	Execution time [μs]
1	MPC	1.654	8150	10
	ANN	1.356	8200	11
2	MPC	1.662	8100	13
	ANN	1.547	8400	11
3	MPC	-	-	>20
	ANN	1.848	8300	11

Table 4.2: Comparison between different steps ahead prediction in steady state for MPC and imitation method using ANN.

4.4 Load step transient responses

One of the main advantages of FS-MPC over other control techniques is its ability to react quickly to unexpected perturbances or reference changes. During these experiments, a step on the load resistance is introduced and the performance of the different models is evaluated. The load is varied from 60 Ω to 30 Ω, and since v_{ref} is kept constant, the output current is doubled.

The load switching device introduces some noise in the current signal after it has performed the resistance change. This issue is purely hardware-related and therefore not considered as part of the transient response.

All the figures include a detailed window showing the transient response with the value of the response time obtained from the voltage graph. This time gives only an approximated idea of the response time. In order to obtain more detailed information about transient characteristics of the step response, the signals must be converted into dq-frame. Nevertheless, since the transients have a very short time span, the noise of the signals impeded the visualization of responses when dq-frame voltages and currents were measured using the dSpace tool.

The experiments show a comparatively poor transient response for the one step FS-MPC model compared to the equivalent ANN model, as shown in Figure 4.10. The figure shows a large spike in phase current while the voltage adjustment is slow, taking a about 320 μs until steady-state is reached with the new load.

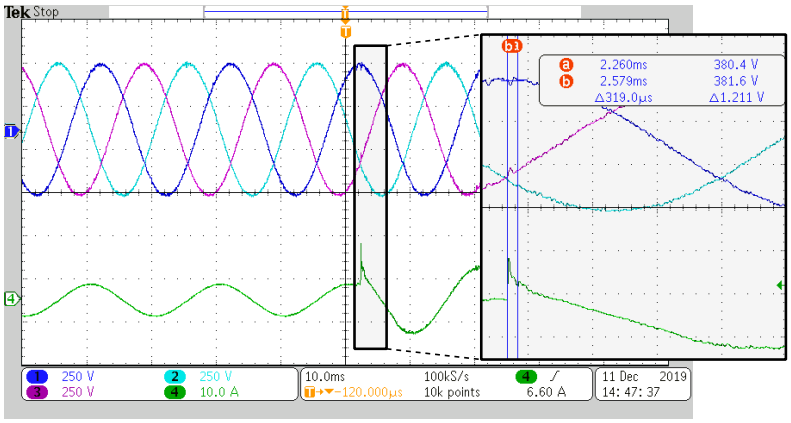

Figure 4.10: FS-MPC with one-step horizon transient performance. Load step from 60 Ω to 30 Ω.

On the other hand, the equivalent ANN model, shown in Figure 4.11, results in a better transient performance, with a much lower current spike and better reaction in the voltage waves.

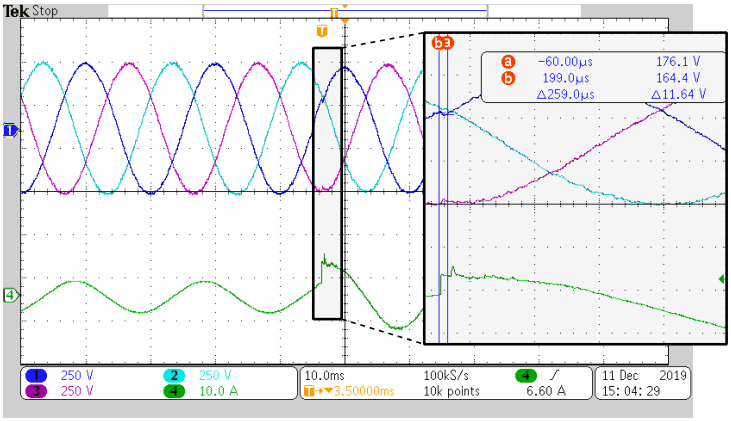

Figure 4.11: ANN with one-step horizon transient performance. Load step from 60 Ω to 30 Ω.

Increasing the prediction horizon, the performance of the FS-MPC is improved. This improvement in transients is to be expected since with a longer prediction window the model is able to adjust more accurately. Figure 4.12 shows how the two-step ahead FS-MPC model is now able to prevent large current spikes and has a reaction time of about 300 µs. In this image, the spikes caused by the load switching device are highly noticeable and can be seen in the current graph once the voltage is settled.

However, Figure 4.13 shows that the ANN scheme still has a better performance

with a settling time of 150 µs. It can again be seen how the voltage is adjusted more effectively with the ANN model.

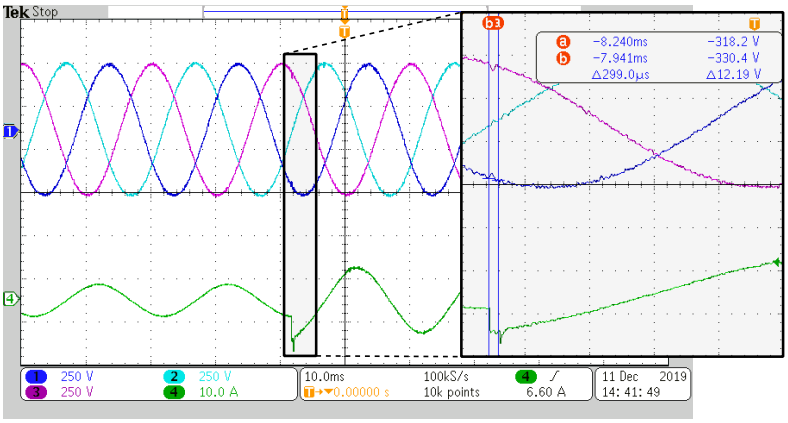

Figure 4.12: FS-MPC with two-step horizon transient performance. Load step from 60 Ω to 30 Ω.

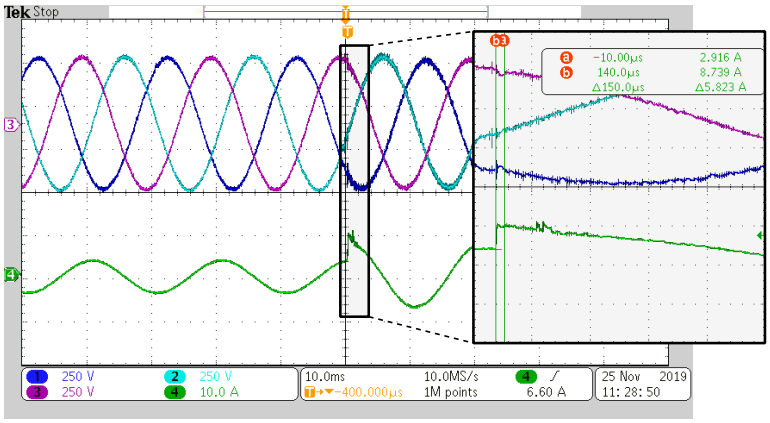

Figure 4.13: ANN with two-step horizon transient performance. Load step from 60 Ω to 30 Ω.

With the three-step ahead ANN model, shown in Figure 4.14, the differences are almost not notable.

In general, it is concluded that any improvement in transient performance with the increase of prediction steps in the algorithm comes as a trade-off with the lower steady state THD obtained for fewer steps prediction horizon.

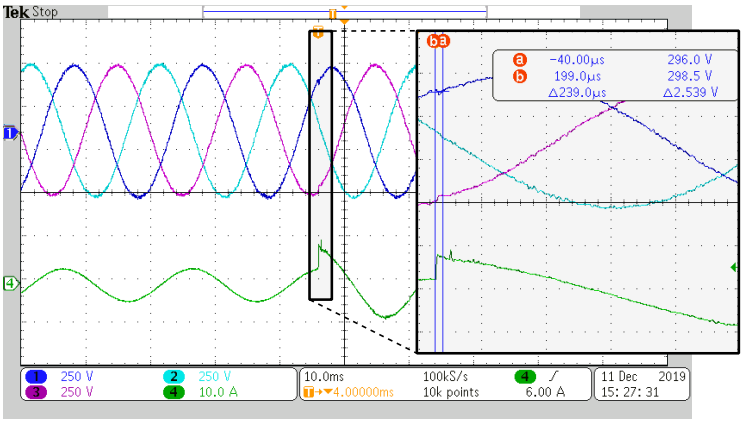

Figure 4.14: ANN with three-step horizon transient performance. Load step from 60 Ω to 30 Ω.

4.5 Robustness

Three-phase grids are not always perfectly stable neither in voltage nor in frequency. For this reason, it is important that a controller that aims to be suitable for grid-connected applications is able to match the grid characteristics even if some values are slightly different than expected.

Two different tests were performed in order to validate that the proposed model is robust enough to deal with grid inconsistencies. Firstly, the reference frequency is changed and secondly, so is the reference voltage. The performance is only evaluated for one step prediction ANN and compared with the equivalent MPC model. It is important to mention that the ANN model has only been trained for values of $f = 50$ Hz and $V_{ref} = 325$ V.

4.5.1 Robustness to frequency changes

Even though frequency inconsistencies in a grid are usually minor, the tests have been performed under highly varying frequencies to give a better idea of the performance of the model when it comes to frequency changes.

Steady state THD and f_{sw} values are presented in Table 4.3. It can be seen how the performance is very accurate, with a very low THD values even for frequencies that vary up to ± 30 Hz from the trained frequency. It is important to notice that the THD values for ANN are slightly lower than for MPC, despite MPC having the frequency reference as an input and the ANN not having been trained for different

frequencies than 50 Hz.

Frequency $[Hz]$	**Controller**	**THD** $[\%]$	$f_{sw}[Hz]$
20	MPC	1.679	8100
	ANN	1.418	8150
40	MPC	1.676	8100
	ANN	1.524	8200
50	MPC	1.640	8150
	ANN	1.356	8200
60	MPC	1.607	8100
	ANN	1.355	8300
80	MPC	1.523	8300
	ANN	1.447	8400

Table 4.3: Steady state performance of MPC and ANN models for different frequency values.

Transients between reference frequencies have also been evaluated and are displayed in Figures 4.15 and 4.16 for MPC and ANN models respectively. For this example, frequency follows a step change from $30Hz$ to $70Hz$. The results show a very fast and accurate transient that is barely noticeable in the figure.

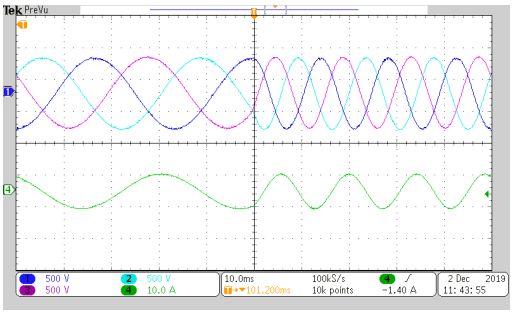

Figure 4.15: MPC frequency robustness, frequency step from 30 Hz to 70 Hz.

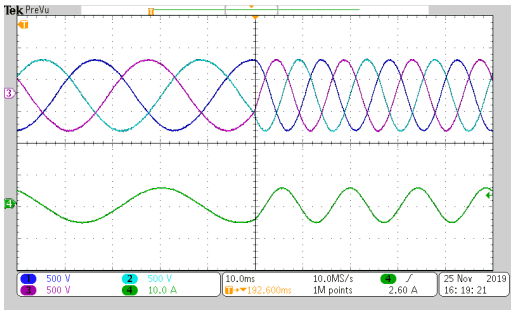

Figure 4.16: ANN frequency robustness, frequency step from 30 Hz to 70 Hz.

4.5.2 Robustness to changes in reference voltage

Similarly to the frequency behaviour, the grid might have variations in the voltage amplitude of the waves; for instance, if a restart procedure occurs (e.g. due to a breakdown event), the grid slowly increases the reference voltage until the required value is reached. For this reason, the model has to be able to adjust to different reference voltage values.

To prevent damage on the laboratory equipment, V_{ref} was never set to higher values than $325V$ so all the tests are performed at lower voltages.

Figure 4.17 and Table 4.4 show the performance of the ANN at different values of V_{ref}. Again, it is seen how the ANN model is able to accurately adjust to different voltage values.

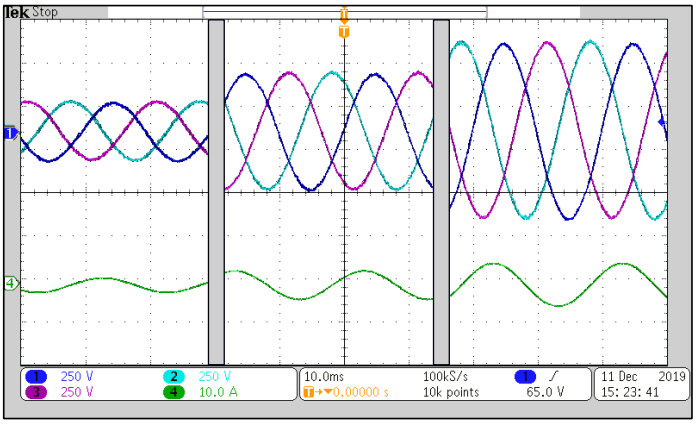

Figure 4.17: Comparison between $100V$, $200V$ and $300V$ behaviour of the ANN 1-step horizon model, the figure is an assembly of three separate oscilloscope screenshots.

$V_{ref}[V]$	THD [%]	$f_{sw}[Hz]$
100	3.124	13400
200	2.007	12500
300	1.564	9200

Table 4.4: Steady state performance when different voltage reference for ANN 1-step horizon model.

Table 4.4 shows how lower values of reference voltage entail also larger switching frequency. This is caused by the increasing amount of state 0 events, where switching is stopped.

It is important to point out that the satisfactory response of the FS-MPC robustness-wise was expected, since it has reference voltage amplitude and frequency as inputs. It is however remarkable to find such good results for the ANN controller. Thereby confirming the mapping ability of the network, since it has correctly extrapolated for untrained inputs.

Chapter 5

Discussion

This chapter aims to summarize the main issues encountered in the development of this project, discussing the reasoning behind the the decisions taken to overcome them. Furthermore, it is also intended to provide commentary on the obtained results and findings.

5.1 The MPC algorithm

The FS-MPC algorithms used in this project follow a conventional implementation: they are based on a state-space model of the converter's output filter and do not penalize switching actions. Although incorporating a switching term to the algorithm's cost function could have resulted in better performance, it was not considered crucial to the goals of the project, as the main objective was to prove that an ANN-based imitator could be developed for the system. However, an additional term was added to the cost function to penalize large voltage derivatives. This term can can greatly improve the performance of the algorithm, especially for short prediction horizons, at almost no additional computational cost.

The parameters used for the state-space model of the converter are taken from the experimental setup made available to the authors. The developed *Simulink* model also uses these same parameters to simulate the behavior of the setup using *Simscape* blocks, and includes the dead-time of the converter's switches.

The MPC algorithms of several steps ahead receive only the voltage references for the first step. Therefore, when implementing them, predictions need to be made on the future voltage references. This prediction is obtained using Lagrange poly-

nomials, as detailed in appendix B.2. The main advantage of using this procedure is that no information is required other than the references at previous sampling steps. Therefore, the predictions remain accurate without explicitly requiring any other information about the reference, be it phase, amplitude, or frequency. The approximations are very accurate, as the sampling frequency of the algorithm (50 kHz) is much higher than that of the reference voltages (at or around 50 Hz).

The algorithm also incorporates delay compensation, by updating the values of the system's states once before entering the optimization process (using the switching action chosen in the previous sample). This is meant to account for the delay to the application of switching actions that the experimental setup introduces.

5.2 The imitation method

The design and training of the neural network were performed using the *Python 3* implementation of the *TensorFlow* library, and making extensive use of the high-level *Keras* API. The main advantage of *TensorFlow* is that it is currently one of the most widely used frameworks for development of ANNs, and it is completely open-source. This entails that the library is very well-maintained and has a great amount of available documentation. It is also very compatible with other software: *Matlab*'s *Deep Learning Toolbox* includes functions to import trained *Keras* networks to it. This was the approach taken to implement the neural networks as *Simulink* controllers.

The data used to train the neural network was obtained by evaluating the MPC algorithms, using combinations of evenly spaced input parameters within the operating limits of the experimental setup. By setting the data points to be spaced in a grid-like fashion, the network is sure to be trained for all possible combinations of parameters, as there will not be any missing points in the training data. Validation and test data are instead obtained as random points in the operating ranges, to ensure that the network can generalize adequately inside these bounds.

Instead of using grid-like values of the α and β components of the reference voltage, a single time parameter is used as an input to the MPC algorithm, and it is then evaluated in sine functions to obtain the two components of the voltage. The neural network, however, still takes the two components of the reference voltage as inputs. This greatly reduces the number of data points to be evaluated using the MPC algorithms. This simplification is valid because the goal of the controller is to obtain a three-phase balanced voltage at a fixed frequency of $50Hz$ and an amplitude of 325 V, and the two components $v_{ref,\alpha}$ and $v_{ref,\beta}$ will therefore always

be coupled. On the other hand, this means that the neural network is only trained for this particular frequency and amplitude, as Lagrange polynomials are not used and sine functions are directly evaluated instead. However, as shown in chapter 4, the controller appears to still be very robust to changes in the reference voltages.

An alternative neural network controller was developed that was designed for changes in reference voltage amplitude and frequency. Curiously, however, it yielded worse THD at reference voltage amplitudes different from 325 V and frequencies different from 50 Hz than the networks trained for this single operating point. More testing should be performed to draw conclusions from this, as the worse THD may simply stem from the fact that this network showed a poorer accuracy than the original ones; and as such a poorer ability to replicate the MPC algorithm. Despite this, it is still indicative of the seemingly great ability of the implemented imitator to generalize to data for which it was not trained for.

The parameters of the neural network, such as choice of cost function, activation function and optimizer were mostly chosen by trial and error, although decisions were informed by literature and documentation. Recognizing the problem to be solved by the neural network as multi-class classification, the choice of the softmax function for the output layer of the network was obvious, as was the choice of categorical cross-entropy as the cost function. For the middle layer, tests were run using sigmoid, hyperbolic tangent and ReLU functions, all with and without dropout. ReLU without dropout was the option that took shorter times to converge and resulted in highest accuracies; and therefore it was chosen as the one to be implemented in the final networks. Similarly, the optimizers SGD, SGD with momentum, *Adagrad*, and *Adam* were all tested for training, as well as some others such as RMSProp. *Adam* was the algorithm most often used as it was the one that made the network's parameters converge the fastest and with good accuracy. This is one field where the *Keras* API excels: it is continuously updated to include newer optimization algorithms, and leaves the choice of both algorithm and algorithm hyper-parameters completely open to the user.

By trial and error, the training batch size was set to 100, and the number of epochs to 20. Due to the inherent randomness in the training of neural networks, training the same network using the same method does not mean that the network's parameters will converge to the same minimum: in fact, accuracy was observed to fluctuate noticeably when running the same code several times. Therefore, the training process was repeated multiple times until a network with a sufficiently high test accuracy could be obtained.

A single hidden layer was chosen for the network, as the added complexity introduced by adding one or several more layers did not result in significant accuracy

increases. The number of neurons in the single hidden layer was set to 15 after comparing the results obtained for a whole range of units. It should be kept in mind, however, that these results may not be completely conclusive. As mentioned, the stochasticity of the training process means that training the same network again could result in better results. The comparison was done by training each network twice and for 10 epochs. A linear neural network was also trained to compare its performance with the non-linear ones, resulting in a validation accuracy of 82.89%, noticeably lower than that of the final networks, which is close to 98%.

The neural network was made computationally cheaper by removing the softmax activation function from the outer layer during simulation and laboratory implementation. Although this function is required to train the network, it does not change its performance when evaluating it, at least in this application: only the position of the largest value in the output layer is used, and applying the softmax function to the layer does not modify this position.

5.3 Experimental tests

Relatively few issues were encountered during the collection of data from the laboratory setup. The mechanical switch used to realize a step load change, going from 60 Ω to 30 Ω, resulted in some undesired current peaks after the initial step, but this should be only a minor issue when analyzing the system's transient response.

Transient responses with changes in load resulted in over-currents. Therefore, the reference voltage amplitude was lowered from 325 V to 225 V for the corresponding tests. Voltage transients are almost completely instantaneous for all the control schemes tested, and therefore their analysis in the *dq* reference frame was not included in this report, as the inherent measurement noise makes it hard to distinguish the transient behavior.

As both the experimental setup and its *dSpace* control scheme had already been prepared and validated when they were made available to the project's authors, implementation of the developed controllers was straightforward.

Chapter 6

Conclusion

In this chapter, the main conclusions drawn from the development and results of this project are discussed. As stated in chapter 1, the main goal of the project was to develop a neural network-based controller that is able to imitate the behavior of finite-set model predictive control for use in a three-phase, six-switch, dc-to-ac converter.

This report documents the steps taken to obtain and verify such controllers. The state-space model of the six-switch, three-phase converter was obtained and implemented in FS-MPC algorithms for one, two, and three-step prediction horizons. After verifying the algorithms, imitator neural networks were trained using *TensorFlow* and verified both in simulation and experimentally. Experimental results are documented in Chapter 4.

Having developed such imitators, and shown that the obtained results from both simulation and laboratory implementation closely match those obtained with FS-MPC algorithms, it can be concluded that this main objective has been fulfilled. It is the opinion of the authors that machine learning and neural networks have a wide range of possible applications in the fields of power electronics and control engineering, as demonstrated by the results shown in this report.

This project shows that control schemes with an unfeasible real-time implementation due to computational limitations may be successfully imitated by a feasible neural network-based controller trained using adequate data. This was shown in the present report only for an FS-MPC algorithm using a 3-step ahead prediction horizon. However, the implications of such a method being successful are much broader: any arbitrarily complex control scheme can be sufficiently well approximated by a neural network imitator, in principle reducing the weight of limitations

imposed by computational requirements.

Nevertheless, a neural network that is able to approximate a very complex control scheme with a high enough accuracy may be so large as to still be unfeasible for a given sampling time. However, the neural network-based imitation method should in almost all cases result in lower computational requirements.

The imitators implemented in this project do not present a significant improvement over the more conventional MPC algorithms, as the setting in which they are implemented does not make much use of the advantages provided by several steps-ahead predictions. However, the possibility of training neural networks offline by making use of simulated controller data is a powerful method that may yield very promising results for replacing other control schemes. This project can then be interpreted as proof of concept.

Chapter 7

Future work

This chapter aims to catalog some possible modifications and improvements that may be implemented to the processes detailed in this project.

As detailed in previous chapters, the main goal of this project was to obtain a neural-network based imitator of model predictive control for application in a six-switch, three-phase DC-AC converter. This main objective has been fulfilled, but the scope of the project may be broadened and some parts of the design process have room for improvement. Some of the authors' suggestions for future work that could be based on this project are listed as follows:

- Expand the operating ranges of the controllers. Both the MPC and ANN controllers can be designed for wider operating ranges and different setups. For example, non-resistive loads or different filter setups.
- Design imitators for MPC acting on more than a three step prediction horizon.
- Modify the MPC algorithm's cost function to account for switching losses or other performance-influencing metrics. In this way, the MPC algorithm would be able to choose between the two zero states of the converter, selecting the one that requires fewer switching actions. Imitator neural networks could then also be trained to select the optimal zero state. Instead of including this choice in the control algorithm, it could also be selected after the controller outputs a zero state action.
- Substitute the state-space model of the inverter in the MPC algorithm for an ANN-based model. Compare results and performance with the conven-

tional state-space based method. This converter model could be trained using experimental data, to ensure that the modelling is as accurate as possible. Imitators could also be designed for such an MPC algorithm. This would mean that even if the ANN model of the converter was very complex (and thus computationally expensive), the algorithm could still be implemented for real-time execution.

- Implement MPC and ANN imitators for a multilevel converter. The advantages provided by ANN imitators should be most noticeable when used for the control of multilevel converters, as MPC may prove impossible to implement in real-time for prediction windows as short as two or even one steps ahead.

- Analyze the performance of different architectures of neural networks that differ from the three layer-deep, fully-connected ANN implemented in this project. Some promising architectures could be convolutional or recurrent neural networks.

- Implement other types of machine learning algorithms such as ensemble learning or search trees as MPC imitators, and compare their performance to that of ANN-based algorithms.

- Study the possible uses of neural networks for use as estimators in drives applications. The ability of neural networks to accurately map complex relationships in data could prove very useful for commonly estimated parameters such as rotor position and speed.

- Design arbitrarily complex controllers for motor drive applications and attempt to obtain adequate ANN-based imitators for them.

- Analyze in-depth the performance of the network when inputs are added, removed, or measurements are replaced by estimators (potentially also neural network-based). Methods such as principal component analysis may prove useful in such a case.

- Train the neural network on-line, by adapting its weights and biases in real time using experimental data. This could result in significantly improved performance.

- Collect experimental data for training and/or validation of ANNs. A network trained with experimental data would, in principle, result in better performance of the control algorithm, as system would be more adequately modelled by the network.

- Implement modifications to the training process of the neural network to ensure that stability is guaranteed. This could be done by modifying the back-propagation algorithm to ensure that at every training step, the network's predictions will satisfy Lyapunov's stability criteria [17].
- Study other applications of neural networks to the field of power electronics, such as reliability analysis, system modelling and identification, or cybersecurity.

Bibliography

[1] Oludare Isaac Abiodun et al. *State-of-the-art in artificial neural network applications: A survey*. Nov. 2018. DOI: 10.1016/j.heliyon.2018.e00938.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[3] José (Professor) Rodríguez and Patricio Cortes. Estay. *Predictive control of power converters and electrical drives*. IEEE/Wiley, 2012. ISBN: 9781119941446.

[4] Fumio Harashima et al. "Application of neural networks to power converter control". In: *Conference Record - IAS Annual Meeting (IEEE Industry Applications Society)*. pt 1. Publ by IEEE, 1989, pp. 1086–1091. DOI: 10.1109/ias.1989.96777.

[5] K. J. Hunt et al. "Neural networks for control systems-A survey". In: *Automatica* 28.6 (1992), pp. 1083–1112. ISSN: 00051098. DOI: 10.1016/0005-1098(92)90053-I.

[6] Xingang Fu et al. "Training Recurrent Neural Networks with the Levenberg-Marquardt Algorithm for Optimal Control of a Grid-Connected Converter". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.9 (Sept. 2015), pp. 1900–1912. ISSN: 21622388. DOI: 10.1109/TNNLS.2014.2361267.

[7] Neeraj Priyadarshi et al. "An AN-GA controlled SEPIC converter for photovoltaic grid integration". In: *Proceedings - 2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering, CPE-POWERENG 2019*. Institute of Electrical and Electronics Engineers Inc., Apr. 2019. ISBN: 9781728132020. DOI: 10.1109/CPE.2019.8862395.

[8] Yang Sun et al. "Artificial Neural Network for Control and Grid Integration of Residential Solar Photovoltaic Systems". In: *IEEE Transactions on Sustainable Energy* 8.4 (Oct. 2017), pp. 1484–1495. ISSN: 19493029. DOI: 10.1109/TSTE.2017.2691669.

[9] Daming Wang et al. "A Deep Neural Network Based Predictive Control Strategy for High Frequency Multilevel Converters". In: *2018 IEEE Energy Conversion Congress and Exposition, ECCE 2018*. Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 2988–2992. ISBN: 9781479973118. DOI: 10.1109/ECCE.2018.8558293.

[10] Flavius Alexandru Luntrasu et al. *Modelling and Control of a Three-Switch , Three-Phase DC to AC Converter*. Tech. rep. 2019.

[11] Yingve Soldbakken. *Space Vector PWM Intro*. 2017. URL: https://www.switchcraft.org/learning/2017/3/15/space-vector-pwm-intro (visited on 12/04/2019).

[12] Muhammad Rashid, Narendra Kumar, and Ashish Kulkarni. *Power Electronics Devices, Circuits and Applications*. 4th. Pearson, 2014, p. 1027. ISBN: 978-0-273-76908-8.

[13] Alex Chumbley and João Areias. *Linear Time-Invariant Systems*. URL: https://brilliant.org/wiki/linear-time-invariant-systems/ (visited on 12/08/2019).

[14] Samir Kouro et al. "Model predictive control - A simple and powerful Method to control power converters". In: *IEEE Transactions on Industrial Electronics* 56.6 (2009), pp. 1826–1838. ISSN: 02780046. DOI: 10.1109/TIE.2008.2008349.

[15] Niklas Panten, Nils Hoffmann, and Friedrich Wilhelm Fuchs. "Finite Control Set Model Predictive Current Control for Grid-Connected Voltage-Source Converters with LCL Filters: A Study Based on Different State Feedbacks". In: *IEEE Transactions on Power Electronics* 31.7 (2016), pp. 5189–5200. ISSN: 08858993. DOI: 10.1109/TPEL.2015.2478862.

[16] Tomislav Dragicevic. "Model Predictive Control of Power Converters for Robust and Fast Operation of AC Microgrids". In: *IEEE Transactions on Power Electronics* 33.7 (2018), pp. 6304–6317. ISSN: 08858993. DOI: 10.1109/TPEL.2017.2744986.

[17] Zhihong Man et al. "A new adaptive backpropagation algorithm based on Lyapunov stability theory for neural networks". In: *IEEE Transactions on Neural Networks* 17.6 (Nov. 2006), pp. 1580–1591. ISSN: 10459227. DOI: 10.1109/TNN.2006.880360.

[18] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. Vol. 521. 7553. 2017, p. 785. ISBN: 3540620583, 9783540620587. DOI: 10.1016/B978-0-12-391420-0.09987-X. arXiv: arXiv:1011.1669v3.

[19] Moshe Leshno et al. *Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function*. Tech. rep. 1993, pp. 861–867.

[20] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. *Deep Sparse Rectifier Neural Networks*. Tech. rep. 2011.

[21] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0.

[22] Erik Meijering. *A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing*. Tech. rep. 2002.

[23] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015. arXiv: 1412.6980.

[24] John Duchi and Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization * Elad Hazan*. Tech. rep. 2011, pp. 2121–2159.

[25] Ashia C. Wilson et al. "The Marginal Value of Adaptive Gradient Methods in Machine Learning". In: (May 2017). arXiv: 1705.08292. URL: http://arxiv.org/abs/1705.08292.

Appendices

Appendix A

Deep neural networks

Artificial neural networks can be considered to be universal function approximators. In principle, a neural network is able to map any arbitrarily complex relationship between a set of given inputs and a desired set of outputs. This makes neural networks, with its many different variants, extremely powerful tools for tasks where this relationship is either hard to describe algorithmically or no such algorithm even exists. For example, they have found many uses in image recognition, a task that would often be almost impossible to solve with regular coding tools.

For a classifier neural network, such as the one presented in this project, some function $y = f^*(\mathbf{x})$ maps a set of inputs $\mathbf{x}$ to a category y. A neural network approximating this relationship would then be defined as $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, ideally learning the set of network parameters $\boldsymbol{\theta}$ that results in the best function approximation. Note that the output category y becomes a vector in the neural network approximator, as the output is often represented using one-hot encoding. A one-hot encoded vector contains a value of 1 in one of its elements and 0 in all the others. In an ideal classifier, where the correct output is chosen every time, y would be the category corresponding to the largest value of $\mathbf{y}$.

Neural networks obtain their name from the fact that they are loosely based on neuroscience, mirroring the behavior of biological neurons and their connection patterns. They are called networks because they result from the composition of several functions. For example, a neural network could be described as the composition of three functions, forming $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. In such a case, the network would be said to have three **layers**, where $f^{(1)}$ would be called the first layer, $f^{(2)}$ the second, and so on. The final layer of the network is called the **output**

layer, in this case $f^{(3)}$. Since the data provided to the network does not specify the desired output of the intermediate layers, these are called **hidden layers**. The dimensionality of each of these hidden layers, which corresponds to the number of units (or neurons) that form it, determines the **width** of the model.

A.1 Neurons and layers

The behavior of each unit, or artificial neuron in a network, is almost always defined by a linear transformation [18]. That is, for a vector of inputs $\mathbf{x}$, the unit output z is obtained as:

$$z = \mathbf{w}\mathbf{x} + b \tag{A.1}$$

This can be interpreted as the unit being connected to all the units in the previous layer (or input data), with their values scaled by the corresponding element of the vector $\mathbf{w}$ and added up together. A bias constant b is then added to the neuron value. Extrapolating to a whole layer, the layer output $\mathbf{z}$ becomes:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{A.2}$$

The matrix $\mathbf{W}$ is called the **weight** matrix, and the vector $\mathbf{b}$ is called the **bias** vector of the unit. Such a model has, however, an important limitation. As each of the layers is simply a linear mapping of the previous one, only linear input-output relationships can be learnt by the neural network. A simple way to overcome this is to apply an element-wise nonlinear function to the output of each layer. This function is commonly called **activation function**. In this way, a well-designed neural network can approximate any function, be it linear or nonlinear. However, this also means that the training process of the neural network almost always becomes a non-convex optimization problem, so obtaining the network parameters is not a problem that can be solved explicitly. Therefore, gradient descent methods must be used, where convergence to the global optimum point is not guaranteed.

Activation functions

Any function can be applied to the output of a layer with the goal of improving the behavior of the neural network, and as long as the function is differentiable, many can yield good results. The simplest case would be to not apply any activation function at all, or equivalently, choose the activation as the identity function. In such a case, the network would become a linear function. Comparing the results obtained using such a network with a non-linear one can be useful to assess the need for non-linearity in a particular application.

More commonly, activation functions are chosen from a set of functions that have been found to perform well in most situations, with research in this area being an active field. Activation functions should be non-polynomial, since this will guarantee that the neural network possesses universal function approximator properties, as proven in [19].

A.1.1 The sigmoid and hyperbolic tangent functions

Two similar and commonly used activation functions are the sigmoid function:

$$g(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{A.3}$$

And the hyperbolic tangent function:

$$g(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{A.4}$$

Graphs of these two functions are shown in figures A.1 and A.2. The two functions

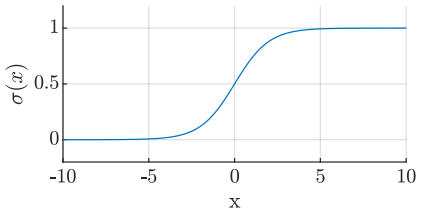

Figure A.1: The sigmoid function.

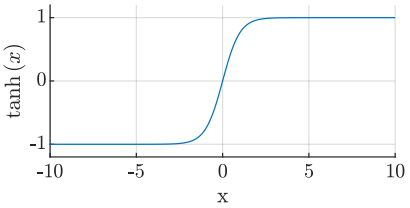

Figure A.2: The hyperbolic tangent function.

have often been favored because their saturating behavior is biologically very plausible, especially in the case of the sigmoid function. In artificial neural networks, the hyperbolic tangent is more often preferred over the sigmoid, as its behavior near the zero point very closely resembles the identity function.

However, both functions present an issue commonly referred to as the vanishing gradient problem. This arises due to the fact that both functions strongly saturate at values of x that deviate from zero, yielding a gradient at these points that becomes vanishingly small. This can make gradient-based learning very difficult.

A.1.2 The rectified linear unit function

In order to overcome the limitations posed by the vanishing gradient problem, the rectified linear unit function, often shortened to ReLU, is becoming more widely used. There also exists research showing that the ReLU function might mimic the behavior of biological neurons more accurately than either the sigmoid or the hyperbolic tangent functions, further justifying the use of this activation function [20]. The ReLU function can be simply defined as:

$$g(x) = \text{ReLU}(x) = \max(0, x) \tag{A.5}$$

This can be visualized as a function with value 0 for all negative values of x, and the identity function for all positive values of x, as shown in figure A.3.

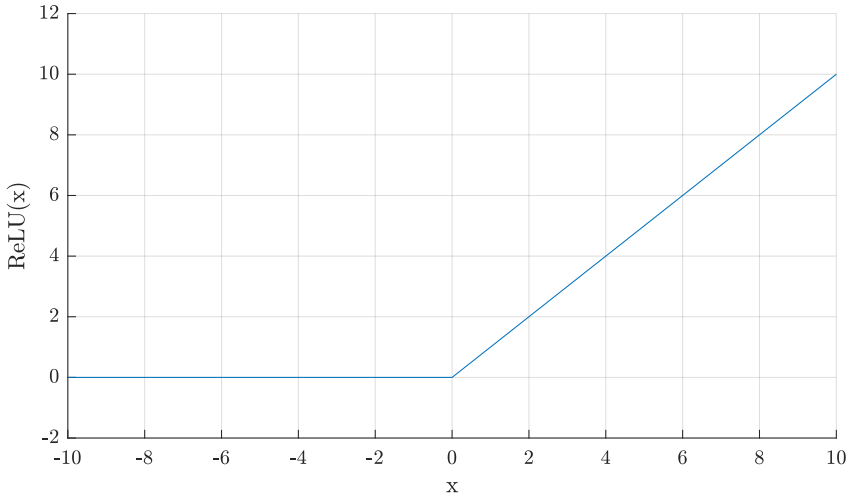

Figure A.3: The rectified linear unit (ReLU) function.

The similarity between ReLU units and linear units makes them easy to optimize compared to other non-linear activation functions. In practice, this often results in shorter training time and faster convergence to optimum points [18].

Due to this reasons, the ReLU activation function is the one used in the intermediate layer of the neural network topology presented in this project.

A.1.3 The softmax function

The choice of the activation function for the output layer usually differs from one chosen for other units. For a classification neural network with an output following a Bernoulli distribution (either 0 or 1), a sigmoid function is often chosen for the output unit, as it maps any input value to a range between 0 and 1. Its output can then be interpreted as how confident the neural network is on the prediction, with an output of 0.5 representing no confidence in either an output of 0 or an output of 1.

In cases where the output of the network is not a single binary variable, but a discrete variable with n possible values, the Bernoulli distribution generalizes to a categorical distribution (also simply called generalized Bernoulli distribution). In such a case, the corresponding generalization of the sigmoid function is the softmax function, which can be defined as follows, for the i-th unit of the layer:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{A.6}$$

The softmax function is not applied strictly element-wise, as it requires the values of the layer's other units to obtain each of unit's output. The function simultaneously ensures that each unit's output has a value between 1 and 0 and that all the outputs of the layer add up to 1, so that the layer represents a valid probability distribution.

The softmax function can pose some of the same training issues as the sigmoid function, as values similarly saturate when one of the values of the layer is much higher than the rest. However, as this function is applied to the output layer, the cost function associated to it can be more easily designed to compensate for the saturation.

The neural network presented in this project uses a softmax layer as its output, as it is solving a multiple category classification problem.

A.2 The back-propagation algorithm

So far, this chapter has been concerned with forward-propagation, i.e. how a deep neural network obtains output values from a given set of inputs. During training, outputs obtained by the neural network produce a scalar cost when compared to their expected values. Back-propagation [21] is an algorithm that allows this information to flow backwards from the outputs, computing the gradients attached

to each parameter of the network that minimize this cost, or equivalently, that minimize the deviation of the network predictions with respect to their labels.

More formally, the goal of back-propagation is to obtain the gradient of the cost function J with respect to the network parameters $\boldsymbol{\theta}$, which in turn include the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$. This gradient can thus be written as $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Back-propagation makes use of the chain rule of calculus to obtain the values of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

A.2.1 The cost function

The first step to do so is to define the cost function used in the network training. The choice of the cost function is an important part of the design of a neural network, and selecting a good cost function depends on the activation function used in the output layer and on the type of problem at hand. In this project, a softmax layer is used as the output, as the problem at hand is multi-class classification, where data points can only belong to a single possible output.

When training a neural network input data is often referred to as features, the desired outputs for each example (which are required in advance) are called labels, and the network's obtained outputs are simply called predictions.

The chosen cost function used for training is categorical cross-entropy, which fits best to this classification problem. Categorical cross-entropy results from applying the softmax function to the final layer's outputs, and then computing its cross-entropy loss. Cross-entropy loss is defined as:

$$CE = -\sum_{j=1}^{n} t_j \log\left(s_j\right) \tag{A.7}$$

Where t_j is the correct label attached to the prediction of the network s_j. As expected, this expression yields a value of 0 when all the predictions match their labels, and larger values the more they do not. In a multi-class classification problem, only one of the elements of the label vector $\mathbf{t}$ is not zero, as the category is encoded in a one-hot vector. Taking this into consideration, and using the softmax function's expression, cross-entropy loss becomes:

$$CCE = -\log\left(\frac{e^{s_p}}{\sum_{j=1}^{n} e^{s_j}}\right) \tag{A.8}$$

Where s_p is the network's prediction for the positive class of the label vector, before applying the softmax function to it. This is the function commonly known as cat-

egorical cross-entropy. Although at first glance it might seem as if this expression does not make use of training labels, they are still required to obtain the value of the sub-index p. Using this function, predictions of zero everywhere except for a one at the position of the label class results in a cross-entropy loss of zero, with loss increasing proportionally with deviations from this ideal case.

A.2.2 Gradient calculation

As introduced above, the goal of back-propagation is to obtain the value of the gradient of the cost function with respect to each of the parameters of the neural network. This gradient corresponds to the rate of change that should be applied to each of the parameters in order to maximize the cost function. This gradient will then be used in an optimization algorithm with the goal of minimizing the value of the cost function, but it must be kept in mind that this is a separate process from back-propagation, which is only concerned with the calculation of gradients.

In order to simplify the analysis, the cost will be assumed to be attached to a single training example. Let J denote the cost function, and w_{ij} denote the weight attached to the connection between the i-th node in layer $L-1$ and the j-th node in layer L. The gradient of the cost function with respect to this weight can be found using the chain rule of calculus:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial a_j}{\partial z_j} \frac{\partial J}{\partial a_j} \tag{A.9}$$

Where $z_j = (\sum_{k=1}^{n} w_{kj} a_k) + b_j$, with n denoting the number of input units to the j-th unit, and $a_j = g(z_j)$, with g being the activation function of the neuron. The problem is then reduced to obtain each of the individual gradients on the right-hand side of equation A.9. Therefore:

$$\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \Big(\sum_{k=1}^{n} w_{kj} a_k \Big) = \frac{\partial}{\partial w_{ij}} (w_{ij} a_i) = a_i \tag{A.10}$$

The next term depends on the chosen activation function of the neuron. In the case of a ReLU activation function:

$$\frac{\partial a_j}{\partial z_j} = g'(z_j) = \text{ReLU}'(z_j) = \begin{cases} 1 & z_j > 0 \\ 0 & z_j < 0 \end{cases} \tag{A.11}$$

The derivative of the ReLU function is undefined at $z_j = 0$. However, most practical implementations of this derivative simply set it to have value zero at this point.

The other activation function used in the networks presented in this project is the softmax function. When using this function, the term becomes:

$$\mathbf{J}(\mathbf{z}) = \mathbf{g}'(\mathbf{z}) = \text{softmax}'(\mathbf{z}) = \begin{cases} a_k(1 - a_l) & k = l \\ -a_k a_j & k \neq l \end{cases} \tag{A.12}$$

Since softmax is a vector function, the equivalent of the gradient $\frac{\partial a_j}{\partial z_j}$ is the Jacobian matrix $\mathbf{J}(\mathbf{z})$. The parameters k and l are the indices of the Jacobian matrix, ranging from 1 to the number of units in the layer (just as the index j).

The partial derivative of the cost function with respect to its input depends, obviously, on the choice of cost function. Assume first that j is in the last layer of the network, and consider a case where square error is used as cost function:

$$\frac{\partial J}{\partial a_j} = \frac{\partial}{\partial a_j}\frac{1}{2}(t - a_j)^2 = a_j - t \tag{A.13}$$

Where t is the correct label attached to the neuron's output.

The networks designed in this project use cross-entropy as their loss function. From equation A.7, the gradient of the function is:

$$\frac{\partial J}{\partial a_j} = -\sum_{k=1}^{m} t_k \frac{\partial \log(a_k)}{\partial a_k} = -\sum_{k=1}^{m} t_k \frac{1}{a_k} \tag{A.14}$$

Where m is the number of neurons in the output layer. This expression can be greatly simplified by combining it with the derivative of the softmax function, shown in A.12. Using the row of the Jacobian matrix corresponding to the neuron j, the gradient $\frac{\partial J}{\partial z_j}$ can be computed directly, as follows:

$$\frac{\partial J}{\partial z_j} = -\sum_{k=1}^{m} \left(t_k \frac{1}{a_k} \frac{\partial a_k}{\partial z_j} \right) = a_j \Big(y_j + \sum_{k=1, k\neq j}^{m} y_k \Big) - y_j \tag{A.15}$$

The target vector $\mathbf{t}$ can be replaced by the one-hot label vector $\mathbf{y}$. Since $\mathbf{y}$ is a one-hot vector, the term $y_j + \sum_{k=1,k\neq j}^{m} y_k$ always adds up to 1. Therefore, this gradient becomes, simply:

$$\frac{\partial J}{\partial z_j} = a_j - y_j \tag{A.16}$$

These expressions assume that the neuron j belongs to the output layer of the neural network. If the unit j is instead in an arbitrary layer, this derivation is less straightforward. Consider the cost J as a function of neurons in a set $L = \{u, v, \cdots, w\}$ receiving input from neuron j. The partial derivative then becomes:

$$\frac{\partial J(a_j)}{\partial a_j} = \frac{\partial J(z_u, z_v, \cdots, z_w)}{\partial a_j} \tag{A.17}$$

Taking the total derivative with respect to a_j, the expression becomes:

$$\frac{\partial J}{\partial a_j} = \sum_{l \in L} \left(\frac{\partial J}{\partial z_l} \frac{\partial z_l}{\partial a_j} \right) = \sum_{l \in L} \left(\frac{\partial J}{\partial a_l} \frac{\partial a_l}{\partial z_l} \frac{\partial z_l}{\partial a_j} \right) = \sum_{l \in L} \left(\frac{\partial J}{\partial a_l} \frac{\partial a_l}{\partial z_l} w_{jl} \right) \tag{A.18}$$

Therefore, this gradient can be computed if all derivatives in the layer L (the layer immediately closer to the output layer) are known. Computations using back-propagation then begin obtaining the gradients for the output layer, and then continue towards the input layer. This is where the algorithm gets its name from, as cost function gradients propagate backwards from the output to the input layers.

Once all three partial derivatives have been obtained, using them in equation A.9 results in the partial derivative $\frac{\partial J}{\partial w_{ij}}$, the goal of the algorithm.

To obtain the gradient of the cost function with respect to a bias instead of a weight, only a simple change must be applied. The basic chain rule expression shown in equation A.9 becomes:

$$\frac{\partial J}{\partial b_j} = \frac{\partial z_j}{\partial b_j} \frac{\partial a_j}{\partial z_j} \frac{\partial J}{\partial a_j} \tag{A.19}$$

Therefore, the only gradient that changes on the right-hand side is the first one, which may be found to be:

$$\frac{\partial z_j}{\partial b_j} = \frac{\partial}{\partial b_j} ((\sum_{k=1}^{n} w_{kj} a_k) + b_j) = 1 \tag{A.20}$$

With the expressions presented in this section, the complete gradient vector $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ can be obtained. In the simplest case, using a steepest descent algorithm, the weights and biases are then updated as:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \nabla_{\boldsymbol{\theta}}^k J(\boldsymbol{\theta}) \tag{A.21}$$

Where η is a scalar constant often referred to as learning rate. In most applications, however, more advanced optimization algorithms are preferred in order to improve training speed and convergence.

Appendix B

Lagrange polynomials

Lagrange polynomials can be used as an interpolation tool. The Lagrange polynomial corresponding to a set of points $(x_{0...k}, y_{0...k})$ is defined as the polynomial of lowest degree that when evaluated at each point x, yields its corresponding value of y. More formally, a Lagrange polynomial can be defined as the polynomial $L(x)$ that satisfies the following with the lowest degree possible:

$$L(x_i) = y_i \quad \forall i \in \{0, 1, \ldots, k\} \tag{B.1}$$

The Lagrange polynomial corresponding to a set of $k+1$ data points can be calculated as [22]:

$$L(x) = \sum_{i=0}^{k} y_i l_i(x) \tag{B.2}$$

Where the Lagrange basis polynomials $l_i(x)$ are defined as:

$$l_i(x) = \prod_{\substack{0 \leq m \leq k \\ m \neq i}} \frac{x - x_m}{x_j - x_0} \tag{B.3}$$

In this project, interpolation is used in the MPC algorithm to obtain estimates of reference voltages for two and three steps ahead, from the reference voltages in the current step and the previous two periods. The sampling periods are also evenly spaced. Let T_s denote the sampling time, and v denote the reference voltages corresponding to the periods, as shown in table B.1.

t	0	T_s	$2T_s$	$3T_s$	$4T_s$
v	v_0	v_1	v_2	v_3	v_4

Table B.1: Definition of voltages for each time period.

Where T_s, v_0, v_1, and v_2 are known, and the values v_3 and v_4 are to be estimated from the corresponding Lagrange polynomial. Using these definitions and equation B.3, the Lagrange basis polynomials corresponding to this problem can be obtained as such:

$$l_0(t) = \frac{(t - T_s)(t - 2T_s)}{2T_s^2} \tag{B.4}$$

$$l_1(t) = \frac{t(t - 2T_s)}{-T_s^2} \tag{B.5}$$

$$l_2(t) = \frac{t(t - T_s)}{2T_s^2} \tag{B.6}$$

Therefore, from equation B.2 the Lagrange polynomial is found to be:

$$L(t) = \sum_{i=0}^{2} y_i l_i(t) = v_0 \frac{(t - T_s)(t - 2T_s)}{2T_s^2} + v_1 \frac{t(t - 2T_s)}{-T_s^2} + v_2 \frac{t(t - T_s)}{2T_s^2} \tag{B.7}$$

Evaluating $L(t)$ at $t = 3T_s$ (which corresponds to two-steps ahead prediction) yields the following expression:

$$v_3 \approx L(3T_s) = v_0 \frac{(2T_s)(T_s)}{2T_s^2} + v_1 \frac{3T_s(T_s)}{-T_s^2} + v_2 \frac{3T_s(2T_s)}{2T_s^2} = v_0 - 3v_1 + 3v_2 \tag{B.8}$$

Similarly, evaluating the Lagrange polynomial at $t = 4T_s$, which would correspond to two steps ahead to the current time step, results in:

$$v_4 \approx L(4T_s) = v_0 \frac{(3T_s)(2T_s)}{2T_s^2} + v_1 \frac{4T_s(2T_s)}{-T_s^2} + v_2 \frac{4T_s(3T_s)}{2T_s^2} = 3v_0 - 8v_1 + 6v_2 \tag{B.9}$$

These expressions can then be implemented in the MPC algorithm, with the following form:

$$v(k+1) = v(k-2) - 3v(k-1) + 3v(k) \tag{B.10}$$

$$v(k+2) = 3v(k-2) - 8v(k-1) + 6v(k) \tag{B.11}$$

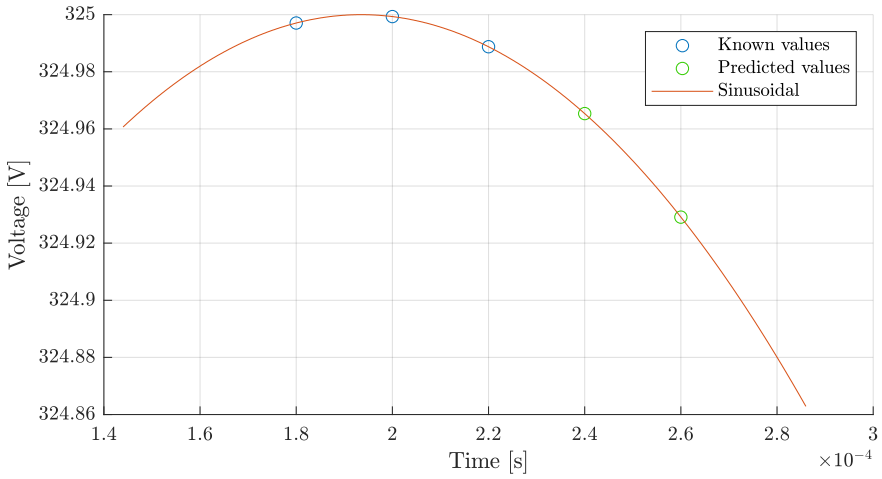

Figure B.1: Visualization of Lagrange polynomial interpolation applied to the control reference generalization for the current project.

Where k is the current sampling instant. To visualize the accuracy of this method, consider a sinusoidal reference with an amplitude of 325 V and a frequency of 50 Hz, sampled with $Ts = 20\ \mu s$, as shown in figure B.1.

Figure B.1 shows that for such a small sampling time with respect to the sinusoidal reference frequency, the approximations provided by Lagrange polynomial interpolation, shown in light blue, are almost indistinguishable from the real values.

Lagrange polynomials are a good match for the interpolation used in this project, as interpolation is only required for a small number of evenly spaced sampling times, so the interpolant does not need to be recalculated and is computationally very cheap. If the points were not evenly spaced, interpolation using Newton polynomials would be better suited to the task.

Appendix C

The *Adam* optimizer

As introduced in appendix A, the training of neural networks is a non-convex and non-linear optimization problem. Therefore, it cannot be solved explicitly and a gradient-based method must be applied. Many such methods have been proposed in recent years for application in this field, as different methods can yield very different results depending on the application.

The *Adam* algorithm was introduced in 2015 by Kingma and Ba [23]. The algorithm aims to improve another commonly used optimization algorithm, *Adagrad*, which tends to result in good performance in settings with sparse gradients (which occurs often for most commonly used activation functions, as explained in appendix A); however, it often struggles to converge in non-convex settings, which is the case for most neural network training problems.

Adam is an adaptive learning rate optimization algorithm. This means that, when training a neural network, its parameters are updated by adding to their values their gradient times a tuneable scalar learning rate. It is the computation of this learning rate with which adaptive learning rate algorithms are mostly concerned about. In order to adequately understand the *Adam* algorithm, some simpler algorithms will be presented first to introduce nomenclature and justify the improvements that *Adam* provides.

One of the simplest and most commonly used optimization algorithms is stochastic gradient descent (SGD), shown in algorithm 1 [18].

The learning rate ϵ is a constant scalar parameter. For the algorithm to converge it must be positive, as the network's parameters must be updated in the opposite direction to the one given by the gradient. Moreover, it should be a value smaller

Algorithm 1: Stochastic gradient descent (SGD)

Require: Learning rate ϵ
Require: Initial parameters $\boldsymbol{\theta}$
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with their corresponding targets $\mathbf{y}^{(i)}$
  Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\mathbf{g}}$
end

than one, to prevent gradients from becoming increasingly large. $\boldsymbol{\theta}$ is a vector containing the values of all the weights and biases of the network at the current iteration.

Before the first step of back-propagation, the parameter vector is often randomly initialized to small non-zero values. This introduces a certain degree of randomness to the optimization of the network, as slightly different initial parameters often result in their convergence to very different local minima. This is true of most neural network training methods, and not unique to SGD.

The concept of batches will be introduced here. When a neural network is trained, instead of computing the value of the cost function and its gradients with respect to the parameters for each of the training examples, several examples are taken at once and the cost function is averaged over the number of examples. This is more computationally efficient than applying the back-propagation algorithm to compute all of the network's gradients for each single training example. The larger the batch size m, the faster the training, as fewer steps of back-propagation need to be computed. On the other hand, smaller batch sizes introduce more stochasticity to the training, which can result in convergence to better cost function minima. This is why this gradient descent algorithm becomes stochastic. Batches should be randomly sampled from the training data to ensure that the gradient estimate is as accurate as possible, while verifying that every data point is used during training.

The gradient estimate $\hat{\mathbf{g}}$ is obtained by averaging the gradient with respect to the parameters of the network, which are obtained by making use of the back-propagation algorithm as detailed in appendix A. $L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$ represents the value of the cost function L for a single training example $\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$, by comparing the prediction of the neural network $f(\mathbf{x}^{(i)}; \boldsymbol{\theta})$ to the label of the training example $\mathbf{y}^{(i)}$.

The stopping criterion for the algorithm can be challenging to design for a non-

convex optimization problem such as the training of a non-linear neural network. Moreover, the inherent stochasticity of the optimization algorithm may cause gradients to abruptly change even after seeming to converge, potentially resulting in the algorithm finding a better minimum point. Common convergence criteria are therefore not very suitable for this application. Usually, the algorithm is made to stop after a certain amount of computational steps. This is often specified as number of epochs, which represent the number of times that the optimization algorithm is applied using all of the training data points.

A common modification to stochastic gradient descent consists on introducing the method of momentum to the algorithm. This can accelerate learning, especially for large gradients, small but consistent gradients, and noisy gradients. The modified algorithm with momentum introduces a variable $\mathbf{v}$ that plays the role of velocity: it is an exponentially decaying average of the negative gradient. This velocity is then used in place of the gradient estimate when updating the parameters of the network. In this way, the update of the parameters has a tendency to continue in the same direction and magnitude as the gradients computed in previous iterations, making the algorithm less prone to becoming stuck in poor local minima as well as accelerating learning. Stochastic gradient descent with momentum is shown in algorithm 2 [18].

Algorithm 2: Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameters $\boldsymbol{\theta}$, initial velocity $\mathbf{v}$
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with their corresponding targets $\mathbf{y}^{(i)}$
  Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$
end

Besides velocity $\mathbf{v}$, only a single parameter is introduced when comparing this algorithm with the more basic SGD: the momentum parameter α. It should be noted that for $\alpha = 0$, the algorithm is completely equivalent to the original SGD. The parameter α should be a scalar with a value between zero and one, just as the learning rate. It determines how quickly the contributions of previous gradients exponentially decay: the larger α is relative to ϵ, the more the previous gradients affect the current direction. Both of these parameters can be adaptively modified to improve performance.

One such approach is the *Adagrad* algorithm [24]. Its pseudo-code is shown in algorithm 3.

Algorithm 3: The *Adagrad* algorithm

Require: Learning rate ϵ
Require: Initial parameters $\boldsymbol{\theta}$
Require: Small constant δ, for numerical stability
Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with their corresponding targets $\mathbf{y}^{(i)}$
  Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ (element-wise operations)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

The *Adagrad* algorithm introduces several new parameters. The constant δ is simply required to prevent division by zero when computing the update $\Delta\boldsymbol{\theta}$, and should be given a very small value. Instead of a velocity $\mathbf{v}$, *Adagrad* uses a gradient accumulation variable $\mathbf{r}$, which stores squared gradient instead of negative gradient. If $\mathbf{v}$ could be thought of as a velocity or momentum term, $\mathbf{r}$ could then be interpreted as kinetic energy (calculated element-wise). The operation of squaring each element of the gradient estimate vector is represented by $\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$.

The parameter update vector $\Delta\boldsymbol{\theta}$ is then computed by scaling the gradient estimate element-wise by the adaptive learning rate $-\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}}$. As $\mathbf{r}$ is initialized to $\mathbf{0}$, this scaling is initially very large but never infinite, thanks to the constant δ. Each component of the gradient vector is scaled by a different value, which translates to a different learning rate being applied to each of the parameters of the network. This learning rate decreases proportionally to the increase in the corresponding component of the accumulated squared gradient vector. This means that the parameters whose gradient is larger have a correspondingly fast decrease in their learning rate, while parameters with smaller gradients maintain a larger learning rate for a longer time. Therefore, optimization progresses towards minima that rely less on larger changes in parameters.

The *Adam* algorithm can be understood as a combination of SGD with momentum and the *Adagrad* algorithm, as it makes use of both accumulated gradient and accumulated squared gradient. These are more generally referred to as first and second moments. Pseudo-code for the *Adam* algorithm [23] is written in algorithm

4.

Algorithm 4: The *Adam* algorithm

Require: Learning rate ϵ (suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 (suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ, for numerical stability
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\mathbf{s} = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$
Initialize time step $t = 0$
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with their corresponding targets $\mathbf{y}^{(i)}$
  Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$
  Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$
  Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$
  Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\epsilon \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}}$ (element-wise operations)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end

The *Adam* algorithm contains three parameters than can be tuned to modify its performance. The parameters ρ_1 and ρ_2 determine the decay rates of the first and second moments, respectively, in an analogous manner to the α parameter found in SGD with momentum. The default values of the two parameters, which result in good performance in most cases and thus are seldom modified in practice, have values close to one. This means that accumulated gradients tend to give more weight to their previous values than to current gradients, or equivalently, that moments decay slowly. As the initial values of both $\mathbf{s}$ and $\mathbf{r}$ are $\mathbf{0}$, the moments always hold gradient values scaled by $(1 - \rho^t)$. This bias is corrected before computing the parameter update by simply dividing the moments by their respective biasing term.

The update rule of the *Adam* algorithm can be understood as being equivalent to the one used in the *Adagrad* algorithm but substituting the gradient vector by the first moment estimate. Therefore *Adam* not only adaptively modifies the scaling of the gradient for each of the parameters of the network, but also takes into account previous gradient directions and magnitudes when computing parameter updates.

The original paper proposing the *Adam* algorithm [23] showed very promising results, obtaining faster and better convergence than all the other algorithms presented in this appendix and than some other commonly used ones. Since it was published, however, there has been research showing that stochastic gradient descent with momentum can, in some settings, find better minimum points more accurately, although often with much larger convergence times [25].

After performing tests with several different optimizers, *Adam* appeared to be the one that provided the best results for the application of this project, showing significantly faster convergence and reaching good although somewhat inconsistent optimum points. Moreover, *Adam* requires a minimum amount of tuning of parameters, as its defaults perform very well in most situations, including the training of network presented in this project. *Adam* was therefore the optimizer used to train the neural networks presented in this report.