ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCHOOL OF ENGINEERING

-Forlì Campus-

SECOND CYCLE MASTER'S DEGREE in
AEROSPACE ENGINEERING
Class LM-20

GRADUATION THESIS IN:

Automatic Flight Control

Data Dependent Convergence Guarantees for Regression Problems in Neural Networks

CANDIDATE:

Jacopo Piccini

SUPERVISORS:

Prof. Paolo Castaldi

Prof. Balázs Kulcsár

EXAMINER:

Prof. Matteo Zanzi

Academic Year 2020/2021

Abstract

It has been recently demonstrated that the artificial neural networks' (ANN) learning under gradient descent method, can be studied using neural tangent kernel (NTK). This thesis' goal is to show how techniques related to control theory, can be applied to model and improve the hyperparameters training dynamics. Moreover, it will be proven how by using methods from linear parameter varying (LPV) theory can allow the exact representation of the learning dynamics over its whole domain. The first part of the thesis is dedicated to the modelling and analysis of the system. The modelling of simple ANNs is hereby suggested and a method to expand this approach to larger networks is proposed. After the first part, the LPV system model's different properties are analysed using different methods. After the modelling and analysis phase, the focus will be shifted on how to improve the neural network both in terms of stability and performances. This improvement is achieved by using state feedback on the LPV system. After setting up the control architecture, controllers based on different methods, such as optimal control and robust control, are then synthesized and their performances are compared.

Keywords: Artificial Neural Networks, Control Theory, Neural Tangent Kernel, Optimal Control, $\mathcal{H}_\infty$ Control, Linear Fractional Representation.

Contents

List of Figures

List of Tables

1 Introduction

Following the works presented in [1] and [2], it was shown how the hyperparameters training in artificial neural networks (ANNs) could be analyzed by means of neural tangent kernel (NTK), i.e. by using methods related to linear time invariant (LTI) systems. Nonetheless, in order to capture the whole training dynamics it would be necessary to perform an interpolation amongst different LTI systems. The idea behind this thesis is to avoid performing the interpolation by using linear parameter varying (LPV) system theory.
The application to machine learning (ML) problems, such as ANNs, of methods from control theory can possibly allow the improvement of both stability properties and performances levels. Furthermore, if adequate stability properties were to be reached, ML methods could start to be used in applications where safety and guaranteed convergence are required.
The path towards the completion of this thesis was not linear. This was mainly due to the lack of bibliography results w.r.t. the application of control theory to ML problems. The initial idea was to model and synthesize a controller using LPV theory. As it will be later shown this approach could not be further investigated because of the difficult in extend and automate the approach to more complex networks, more similar to what are used in reality. Because of this difficult and the presence of software tools, it was chosen to use linear fractional representations to perform the modeling phase. This approach is more promising as is suitable to be automated. The thesis then shifts its goal to the synthesis of a controller to improve the networks' stability and performances. This part is only a proof of concept as the controller is synthesized for a single neuron. Nonetheless, as it will be proven in Chapter 3, a single neuron and a single layer made of n-many neurons are very similar, hence similar results for the closed loop systems can be expected.

2

Theory

2.1 Neural Network Training Dynamics

In this section, the procedure for deriving the training dynamics equation for a neural network will be shown. For sake of simplicity, the equations for a single neuron will be derived. In Chapter 3 the approach will be expanded to more complex networks.

2.1.1 Single Neuron Training Dynamics Equation

As shown in [1] the training dynamics in parameter space of a wide Neural Network can be seen as a linear model with as input the data signal d and as output the estimated label $\hat{y}$. As a first step the equations governing the dynamics of a single neuron learning under gradient descent will be derived.

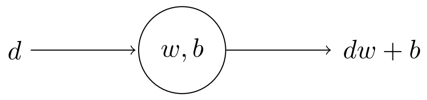

Figure 2.1: Single Neuron Single Layer Network

As shown in [3] a system representing the network parameters dynamics can be obtained under certain assumptions. These being the network under gradient flow training, and a mean squared error (MSE) loss function to be used:

$$\mathcal{L}(t) = (y - \hat{y}(t))^2 \tag{2.1}$$

where the estimated label is denoted as:

$$\hat{y}(t) = dw(t) + b(t) \tag{2.2}$$

Then, the equations for the state vector $\mathbf{x} = [w,\ b]^T$ can be obtained by expanding the following equation, where $\nabla\mathcal{L}$ is the vector with as components the single derivatives of $\mathcal{L}$ take w.r.t. the different hyperparameters of the system.

$$\dot{\mathbf{x}} = -\alpha\nabla\mathcal{L} \tag{2.3}$$

By specializing it for a single neuron single layer network:

$$\begin{cases} \dot{w} = -\alpha\frac{\partial}{\partial w}(y - \hat{y})^2 \\ \dot{b} = -\alpha\frac{\partial}{\partial b}(y - \hat{y})^2 \end{cases} \tag{2.4}$$

Using Eq.(2.2) and expanding Eq.(2.4) the following system is obtained:

$$\begin{aligned}\dot{w}(t) &= -2\alpha d^2 w(t) - 2\alpha d b(t) + 2\alpha d y \\ \dot{b}(t) &= -2\alpha d w(t) - 2\alpha b(t) + 2\alpha y\end{aligned} \tag{2.5}$$

The latter system, which represents the neuron hyperparameters training dynamics, can be re-written as:

$$\begin{aligned}\begin{bmatrix}\dot{w}(t) \\ \dot{b}(t)\end{bmatrix} &= \begin{bmatrix}-2\alpha d^2 & -2\alpha d \\ -2\alpha d & -2\alpha\end{bmatrix}\begin{bmatrix}w(t) \\ b(t)\end{bmatrix} + \begin{bmatrix}2\alpha d \\ 2\alpha\end{bmatrix} y \\ \hat{y}(t) &= \begin{bmatrix}d & 1\end{bmatrix}\begin{bmatrix}w(t) \\ b(t)\end{bmatrix}\end{aligned} \tag{2.6}$$

Where:

- y is the label
- w is the weight of the neuron
- b is the bias
- d is the current data value
- α is the learning rate

From the previous equation it may be seen that the system model cannot be represented by a linear time invariant system (LTI) since there are dependencies on both d, α which may vary in time, but the whole learning dynamics could be represented by sampling the system for different values of d, α. Furthermore, this system has no manipulable variable only a target that behaves as a reference signal or disturbance. In order to model these nonlinearities and cover with a single linear parameter varying (LPV) system model the whole training domain, techniques from linear parameter varying (LPV) system models theory will be used.
It can be noticed that in the output equation an activation function was not used. Its lack is mainly due to its high nonlinearity. Moreover if an MSE loss function is assumed, it is not possible to eliminate them. In fact, we make an implicit assumption here. Only the unsaturated region of the ANN is used. Some saturation function has a smooth analytical form and therefore can be incorporated. However this is left for future work. This holds both for a single neuron training dynamics and for more complex networks. On a final note, a solution to this would be to use Zames-Falb multipliers[9] and integral quadratic constraints (IQC) [10], but this goes over the aim of this thesis.

2.2 Linear Parameter Varying System Models

The reason why LPV system models are useful is that they allow to transform nonlinear systems in linear systems by collecting nonlinearities inside specifically defined parameters. Then, similar methods to the ones used for LTI system might be used. The single neuron case will now be used to present some of the LPV techniques that will be used.
The state space form for a LPV system:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A(\mathbf{p}(t))\mathbf{x}(t) + B(\mathbf{p}(t))\mathbf{u}(t) \\ \hat{y}(t) &= C(\mathbf{p}(t))\mathbf{x}(t) + D(\mathbf{p}(t))\mathbf{u}(t)\end{aligned} \tag{2.7}$$

In this thesis, and for sake of computational simplicity, it will be attempted to obtain an affine (or linear) in the parameter $\mathbf{p}(t)$ model. This means that the system described in Eq.(2.7) can be re-written as:

$$\begin{bmatrix} A(\mathbf{p}(t)) & B(\mathbf{p}(t)) \\ C(\mathbf{p}(t)) & D(\mathbf{p}(t)) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \sum_{1}^{n_p}{}_j\, p_j(t) \begin{bmatrix} A_j & B_j \\ C_j & D_j \end{bmatrix} \tag{2.8}$$

Before setting up the parameters vector a consideration on α must be made. In general it is true that the learning rate value may vary but throughout the learning iteration, i.e. the learning process for a certain value of d, is kept constant and it is known. Because of this, the learning rate will be considered as a fixed value. This hypothesis will then be loosen when the GSS Library and LFR Toolbox [4] will be used, this because of computational issue. The parameters vector $\mathbf{p}(t)$ will then become:

$$\mathbf{p}(d(t)) = \begin{bmatrix} d(t) \\ d^2(t) \end{bmatrix} = \begin{bmatrix} p_1(t) \\ p_2(t) \end{bmatrix} \tag{2.9}$$

Where n_p is the number of parameters with affine parameterization and p_j is the j-th component of $\mathbf{p}(t)$. Although an affine LPV is the simplest LPV, especially compared to rational or polynomial LPVs, it is still needed to identify $4 \times (n_p + 1)$ matrices to be able to construct the system presented in (2.8).

A few considerations must be made on the parameters vector classification. A parameter vector of this kind is said to be exogenous as its component do not depend on the system's signals, i.e. $\mathbf{x}, u, y$. The system matrices can now be rewritten w.r.t. the parameters vector:

$$\begin{aligned} A(\alpha, \mathbf{p}) &= \begin{bmatrix} -2\alpha p_2 & -2\alpha p_1 \\ -2\alpha p_1 & -2\alpha \end{bmatrix} \\ B(\alpha, \mathbf{p}) &= \begin{bmatrix} 2\alpha p_1 \\ 2\alpha \end{bmatrix} \\ C(\alpha, \mathbf{p}) &= \begin{bmatrix} p_1 & 1 \end{bmatrix} \\ D(\alpha, \mathbf{p}) &= 0 \end{aligned} \tag{2.10}$$

Note, the original GD flow and the LPV model in 2.10 are equivalent, in other words, the parameterization results in an exact reformulation of the learning dynamics. For sake of notation the time-dependency of $\mathbf{p}$ is omitted.

For sake of completion, if α was to be taken as a varying parameter of the system, the parameter vector $\mathbf{p}$, would then be the following:

$$\mathbf{p} = \begin{bmatrix} \alpha d^2 \\ \alpha d \\ \alpha \\ d \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{2.11}$$

and the resulting matrices will be:

$$\begin{aligned} A(\mathbf{p}) &= \begin{bmatrix} -2p_1 & -2p_2 \\ -2p_2 & -2p_3 \end{bmatrix} \\ B(\mathbf{p}) &= \begin{bmatrix} 2p_2 \\ 2p_3 \end{bmatrix} \\ C(\mathbf{p}) &= \begin{bmatrix} p_4 & 1 \end{bmatrix} \\ D(\mathbf{p}) &= 0 \end{aligned} \tag{2.12}$$

A small comment on the parameters vector must be now made. Since, for the single neuron case, the data and learning rate value, if not taken as a constant, are given as input, and both their minimum and maximum values are known. This is enough to conduct an analysis using LPV techniques. Furthermore, it is often plausible to work with normalized data. This clearly states that **p** can not be arbitrary large and belongs to a range. This property is beneficial and exploited in LPV analysis and synthesis.

2.2.1 Stability

For a LTI system, the stability analysis can be performed by determining if there exists a symmetric positive definite matrix $P \succ 0$ and a positive definite matrix $Q \succeq 0$ such that the following relation holds:

$$A^T P + PA = -Q \tag{2.13}$$

Verifying this relation is equivalent to checking that the eigenvalues of A belongs to the left half plane. thus guaranteeing the system's asymptotic stability if $c > 0$. The property $\mathrm{Re}(\lambda_i) < 0$, namely all the eigenvalues' real parts belong are strictly less than zero, can be revisited by solving the following inequality for a real scalar $c \geq 0$:

$$A^T P + PA + cP \prec 0 \tag{2.14}$$

While $P = P^T \succ 0$. If this inequality is verified, then it is guaranteed that $\mathrm{Re}(\lambda_i) < c$.

In order to asses a LPV system model Eq.(2.14) will be used. Moeover, the symmetric positive definite matrix P will be used to construct a parameter independent Lyapunov function V:

$$\begin{aligned} &V(\mathbf{x}) = \mathbf{x}^T P \mathbf{x} \\ &\text{with } V(\mathbf{x}) > 0 \text{ and } V(\mathbf{0}) = 0 \end{aligned}$$

The scalar c is still real and non-negative. Then, the LPV system's quadratic stability is guaranteed iff:

$$M(\mathbf{p}) = P(A(\mathbf{p}) + A^T(\mathbf{p})P + cP \prec 0 \tag{2.15}$$

The latter inequality must be verified over the whole range of **p**. The inequality can then be said to be satisfied if:

$$\begin{aligned} \lambda(P) &\in \text{RHP} \\ \lambda(M) &\in \text{LHP} \end{aligned} \tag{2.16}$$

The inequality presented in Eq.(2.15) can be assessed by assuming that the parameter polytope $\mathcal{P} \subset \mathbb{R}_p^n$ in which $\mathbf{p}$ belongs is convex. This assumption is of fundamental importance, because if the inequality is satisfied at the polytope's vertices then it is always satisfied.

2.2.2 Structural Reachability

The reachability, also called (input-output) controllability, is the property of a system to achieve acceptable control performances as stated in [7].
Following what is said in [8], the Kálmán full rank condition can be used for a LPV system model with a slight modification.
The LPV system model is said to be reachable iff the matrix $\mathcal{R}$ is full rank, i.e. $rank(\mathcal{R}) = n = dim(x)$, $\mathcal{R}_1 = B(\mathbf{p})$ and $\mathcal{R}_{i+1} = \dot{\mathcal{R}}_i - A(\mathbf{p})\mathcal{R}_i$.

$$\mathcal{R} = \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 & \dots & \mathcal{R}_n \end{bmatrix} \tag{2.17}$$

In the following chapter, two different analyses will be run, the first one neglecting the dependence w.r.t. time of the parameter vector, i.e. $\dot{\mathbf{p}} = \mathbf{0}$. While the second one will consider a possible time-dependency of $\mathbf{p}$. Although the second simulation is far more general, it will be shown how the two different simulations converge to the same result. The reachability matrix for the system defined in Eq.(2.10) will be:

$$\begin{aligned} \mathcal{R} &= \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{R}_1 & \dot{\mathcal{R}}_1 - A(\mathbf{p})\mathcal{R}_1 \end{bmatrix} \\ &= \begin{bmatrix} B(\mathbf{p}) & \dot{B}(\mathbf{p}) - A(\mathbf{p})B(\mathbf{p}) \end{bmatrix} \end{aligned} \tag{2.18}$$

As it will be shown in the following chapter, depending on the trajectory of $\mathbf{p}$ we can have a $\mathbf{p}(t)$ related actual reachability problem as the Kálmán condition may not be fulfilled.

2.2.3 Structural Observability

In the same fashion as it has been done for the structural reachability, an analysis of the structural observability will be carried on.
Following the definition given in [5], a LPV system is said to be structurally observable iff:

$$\begin{aligned} rank(\mathcal{O}) &= rank(\begin{bmatrix} \mathcal{O}_1^T & \mathcal{O}_2^T & \dots & \mathcal{O}_n^T \end{bmatrix}^T) \\ &= n = dim(\mathbf{x}) \end{aligned} \tag{2.19}$$

Namely meaning that the Kálmán full-rank condition must be fulfilled for the matrix $\mathcal{O}$ which is the observability matrix which is built in the following way:

$$\begin{aligned} \mathcal{O}_1 &= C(\mathbf{p}(t)) \\ \mathcal{O}_{i+1} &= \dot{\mathcal{O}}_i + \mathcal{O}_i A(\mathbf{p}(t)), \qquad i > 1 \end{aligned} \tag{2.20}$$

As done for the structural reachability two different analyses will be made, these being the time-independent and the time-dependent to see how the structural observability property is impacted by the time dependency.

2.3 Linear Fractional Representation

One possible way to represent LPV systems is to use a transformation that separates the scheduling parameters from the system model parameters. As such, we use linear fractional transformations (LFT) to do so.
The LFR is a method of dealing with uncertain system by extracting the uncertainty in a sub-system. The overall system will then be characterized by the following two sub-systems:

- M, which is the nominal plant.
- P, the sub-system capturing all the uncertain part.

The two sub-systems blocks are then interconnected in the following way:

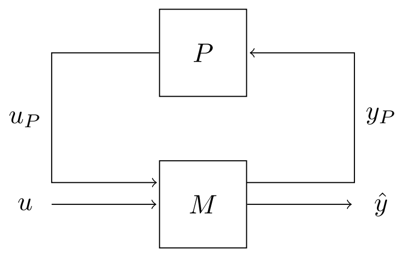

Figure 2.2: Blocks interconnection of the systems obtained with the results of the LFR

In order to go back from the nominal and uncertain sub-systems to the original uncertain plant, an upper linear fractional transformation (LFT) is applied.

2.4 Control Theory

The control of the neural network under analysis will be achieved by following two different approaches. The first one involves the use optimal control while the second one involves the use of robust control.

2.4.1 Optimal Control

Linear Quadratic Regulator

Optimal control theory envisages the control of a LTI system expressed in its state-space form, i.e.:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \hat{y}(t) &= C\mathbf{x}(t) \end{aligned} \tag{2.21}$$

By using a control input defined in the following way:

$$\mathbf{u}(t) = -K\mathbf{x}(t) \tag{2.22}$$

The closed loop system will then become:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= (A - BK)\mathbf{x}(t) \\ \hat{y}(t) &= C\mathbf{x}(t) \end{aligned} \tag{2.23}$$

The matrix K is a constant matrix such that the cost function $J(t)$ is minimized. The cost function is defined by introducing the weighting matrices Q, R. These two matrices are used to penalize respectively the state vector or the control input action.

$$J = \frac{1}{2}\int_0^\infty \Big(\mathbf{x}(t)^T Q\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t)\Big)\, dt \tag{2.24}$$

The optimal cost function will then be given by:

$$J^* = \frac{1}{2}\mathbf{x}_0^T P\mathbf{x}_0 \tag{2.25}$$

Where $\mathbf{x}_0$ is the initial state vector, and P is the solution to the Control Algebraic Riccati Equation (CARE):

$$PA + A^T P + Q = PBR^{-1}B^T P \tag{2.26}$$

The control matrix can then be defined by using the Riccati solution P:

$$K = R^{-1}B^T P \tag{2.27}$$

The solution to the CARE will be found by using built-in functions in Matlab®.
The use of a linear quadratic regulator (LQR) causes two main issues:

- the controller is able to stabilize the system only for a certain combination of α, d
- a label different than zero causes an effect equivalent to a disturbance or to a reference signal

The first issue can be overcome by gain-scheduling, i.e. properly discretizing the surface on which the parameters α, d vary, synthesizing a controller for each point on the grid and then performing an interpolation between the controllers.
The second issue cannot be addressed in a generic way. In order to overcome this difficulty, a different kind of controller, described in the following section, will be used.

Linear Quadratic Integral Regulator

The synthesis process is the same, i.e. a quadratic cost function analogous to the one described in Eq.(2.24) is defined and by solving the optimization process the control matrix K is found. Although the optimization problem is similar, the control architecture is substantially different.

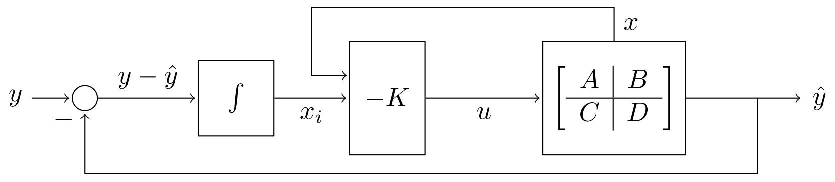
y
$-$
$y - \hat{y}$
$\int$
x_i
$-K$
u
x
$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$
$\hat{y}$

Figure 2.3: Control architecture of the linear quadratic integral controller. The matrices A, B, C, D are augmented in dimensions because of the control architecture.

This time the controller is a $[1 \times 3]$ matrix. The third state x_i is the integrated difference between the input label and the system's estimated label.
The closed loop system matrices must then be reformulated as, w.r.t. the standard LQR case, there is an additional state. The third equation of the system is defined in the following way:

$$\dot{x}_i(t) = y - \hat{y}(t) = y - C\mathbf{x}(t) \tag{2.28}$$

Now, the complete system dynamics can be reconstructed in its state-space formulation:

$$\begin{aligned} \begin{bmatrix} \dot{\mathbf{x}}(t) \\ \dot{x}_i(t) \end{bmatrix} &= \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ x_i(t) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} B \\ 0 \end{bmatrix} y \\ \hat{y}(t) &= \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ x_i(t) \end{bmatrix} \end{aligned} \tag{2.29}$$

Once again, the LQI controller will be synthesized only for a specific combination of α, d. In order to cover the whole domain on which α, d may vary, a gain-scheduled LQI controller will be synthesized following the same procedure described for the case of a LQR.
As previously said, going from the use of a LQR to a LQI regulator, has caused a change in the system's dimensions. Moreover, the new state vector has been augmented with the new state x_i. Because of this, the optimal cost function's state weighting matrix, i.e. Q, must be augmented as well. The new state penalizing matrix will be called Q_i and it is defined in the following way:

$$Q_i = \left[\begin{array}{c|c} Q & 0 \\ \hline 0 & q_i \end{array}\right] \tag{2.30}$$

In our case, q_i is a scalar constant and is the weight on the new state. While the new state weighting matrix needed to be augmented in dimensions, the control weighting matrix remains the same since the dimensions of u do not change. Having defined Q_i R, the new optimal cost function is defined as follows:

$$J = \frac{1}{2}\int_0^\infty \left([\mathbf{x}(t) \quad x_i(t)]\, Q_i\, [\mathbf{x}(t) \quad x_i(t)]^T + \mathbf{u}^T(t) R \mathbf{u}(t) \right) dt \tag{2.31}$$

2.4.2 Robust Control

In the previous sections, the controller was found by performing an interpolation amongst different controllers. Although, as it will be proven in Chapter 3, this methods provides good performances, it causes a great computational burden as a LQR (or a LQI controller) must be synthesized for $n \times m$ different conditions, where n is the amount of points required to properly discretize the range of d and m is the number of different values of α which discretize its range. This process may be affordable if both ranges are small, on the other hand if the ranges get larger, creating a dense-enough grid might lead to a significant computational burden. In order to avoid the high computational cost, another branch of control theory must be used: robust control.

Robust control deals with the synthesis of controllers for uncertain systems. The synthesis process is based on the minimization of the $\mathcal{H}_\infty$ norm of an interconnected system. The optimization process is performed on an interconnected system where the input and output signals of the original plant systems are combined in order to reach certain performances levels. In addition to the combination of different signals, they can be weighted in order to give more importance to certain signals' behaviours. The optimization process gives as a result a controller which will then be set in feedback with the augmented system. The $\mathcal{H}_\infty$ norm of the closed loop system transfer function $T(s)$ is defined as follows and it is named γ.

$$\gamma = ||T(s)||_\infty = ||T(j\omega)||_\infty = \max_{\omega} \bar{\sigma}(G(j\omega)) \tag{2.32}$$

The objective of the controller is to minimize the value of γ. The $\mathcal{H}_\infty$ norm minimization problem could then be set in the following way:

$$\min_{K,\gamma} \max_{w\neq 0} \left(||\mathbf{z}(t)|| - \gamma||\mathbf{w}(t)||\right) \tag{2.33}$$

In the latter equation $\mathbf{z}(t)$ are the output performance signals of the augmented plant while $\mathbf{w}(t)$ are the non-zero disturbances vector.

3
Method and Results

3.1 Single Neuron LPV Analysis

3.1.1 Parameterization

The LPV formulation of an affine model can be given as:

$$A(p) = A_0 + \sum_{j=1}^{n_p} p_j A_j \tag{3.1}$$

Where with $\mathbf{p}(t)$ is a parameter vector. For the single neuron training case:

$$\mathbf{p}(t) = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} d \\ d^2 \end{bmatrix} \tag{3.2}$$

Resulting in:

$$A_0 = \begin{bmatrix} 0 & 0 \\ 0 & -2\alpha \end{bmatrix},\ A_1 = \begin{bmatrix} 0 & -2\alpha \\ -2\alpha & 0 \end{bmatrix},\ A_2 = \begin{bmatrix} -2\alpha & 0 \\ 0 & 0 \end{bmatrix} \tag{3.3}$$

By applying the same approach to the matrix B, still considering the first two states only:

$$B(\mathbf{p}(t)) = B_0 + B_1 p_1 + B_2 p_2 = \begin{bmatrix} 0 \\ 2\alpha \end{bmatrix} + \begin{bmatrix} -2 \\ 0 \end{bmatrix} d + \begin{bmatrix} 0 \\ 0 \end{bmatrix} d^2 \tag{3.4}$$

A very important property of this LPV system model is that so far no approximations were made. This means that an exact LPV model is obtained, which matches the original nonlinear behaviour.

3.1.2 Stability Analysis

The stability analysis both asymptotic st. and quadratic st. will be conducted with the help of [6] following the procedures explained in [5], i.e. the quadratic stability of the LPV system is defined by:

$$M(A_i) = PA(\mathbf{p}_i) + A^T(\mathbf{p}_i)P + cP \prec 0 \tag{3.5}$$

The previous equation needs to be solved numerically. But, it would require a definite programming tool which does not exist. In order to overcome this issue, the latter equation will be set in a semi-definite form:

$$M(A_i) = PA(\mathbf{p}_i) + A^T(\mathbf{p}_i)P + cP \preceq 0 \tag{3.6}$$

In this case: $i = 1, 2, 3, 4$. Where $\mathbf{p}_i$ is denoting the vertices of the assumed convex parameter polytope $\mathcal{P}$.

i	$\mathbf{p}_i$	p_1	p_2
1	$\mathbf{p}_1$	p_{1min}	p_{2min}
2	$\mathbf{p}_2$	p_{1min}	p_{2max}
3	$\mathbf{p}_3$	p_{1max}	p_{2min}
4	$\mathbf{p}_4$	p_{1max}	p_{2max}

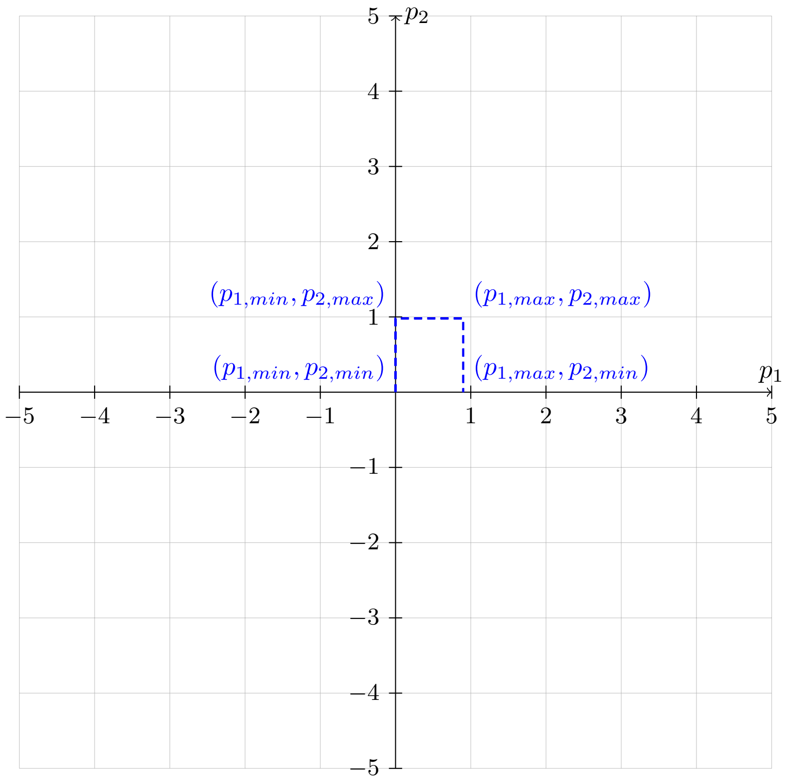

Figure 3.1: Vertices of the convex parameter polytope $\mathcal{P}$. The two components of the parameters are assumed independent from each other. In this way a more robust stability analysis is obtained.

Where $P = P^T \succ 0$, $V(\mathbf{x}) = \mathbf{x}^T P \mathbf{x}$, and real scalar $c \geq 0$. For this first analysis: $\alpha = 2.2e - 1$. The choice of α, at least for the initial simulations, is related to the SGD optimized learning rate presented in [3]. The analysis of Eq.(3.6) for different values of c in terms of eigenvalues will be later presented. As parameters ranges a normalized data batch results into:

$$\begin{aligned} 0 \leq p_1 < 1 \\ 0 \leq p_2 < 1 \end{aligned} \tag{3.7}$$

In the first range the strict inequality on the upper bound is set to prevent A from becoming rank deficient, i.e. having one zero eigenvalue. Moreover the value is set

to 0.99. The second range is derived from the relation $p_2 = p_1^2$.
Changing the value of c the following eigenvalues are obtained:

$\lambda(P)$	$\lambda(M(A(\mathbf{p}_1))$	$\lambda(M(A(\mathbf{p}_2)))$	$\lambda(M(A(\mathbf{p}_3)))$	$\lambda(M(A(\mathbf{p}_4)))$
1.097e-11	-1.963e-10	-2.359e-10	-3.983e-10	-4.662e-10
3.040e-10	1.558e-11	-8.403e-12	8.016e-12	1.226e-11

Table 3.1: Eigenvalues analysis while using LMILAB, $c = 0.1$, $\alpha = 2.2e - 1$.

$\lambda(P)$	$\lambda(M(A(\mathbf{p}_1))$	$\lambda(M(A(\mathbf{p}_2)))$	$\lambda(M(A(\mathbf{p}_3)))$	$\lambda(M(A(\mathbf{p}_4)))$
0.002e-9	-0.443e-9	-0.535e-9	-0.954e-9	-0.112e-8
0.791-12	0.044e-9	-0.002e-9	0.007e-9	0.004e-8

Table 3.2: Eigenvalues analysis while using LMILAB, $c = 0.2$, $\alpha = 2.2e - 1$.

$\lambda(P)$	$\lambda(M(A(\mathbf{p}_1))$	$\lambda(M(A(\mathbf{p}_2)))$	$\lambda(M(A(\mathbf{p}_3)))$	$\lambda(M(A(\mathbf{p}_4)))$
-2.168e-12	-5.103e-11	-5.062e-11	-5.103e-11	-5.062e-11
6.326e-11	1.369e-11	1.519e-11	1.369e-11	1.519e-11

Table 3.3: Eigenvalues analysis while using LMILAB, $c = 0.3$, $\alpha = 2.2e - 1$.

A LPV system model is said to be quadratically stable (QS) if all the eigenvalues of P lay in the right half plane (RHP), i.e. $\lambda(P) \in RHP$ and all the eigenvalues of the A matrices evaluated at the domain vertices lay in the left hand plane (LHP), i.e. $\lambda(M(A_i)) \in LHP$. As it may be seen by looking at the previous tables, the QS stability condition is never satisfied although the eigenvalues magnitude is close to zero, although the eigenvalues may reflect numerical errors and hence further refinements have to be done. Thus, it is not possible to conclude anything about quadratic stability.
Finally, what can be noticed that from this analysis is that the pairs of points $\mathbf{p}_1,\ \mathbf{p}_3$ and $\mathbf{p}_2,\ \mathbf{p}_4$ lead to the same eigenvalues $\lambda_{1,2}$.

3.1.3 Structural Reachability Analysis

Following what is said in Chapter 2, the Kálmán full rank condition can be used for a LPV system model with a slight modification.

$$\mathcal{R} = \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 & \dots & \mathcal{R}_n \end{bmatrix} \tag{3.8}$$

The first analysis will be run neglecting the time dependency of the parameter vecctor, i.e. the time derivative will not be taken into account.
For the single neuron training dynamics:

$$\begin{aligned} \mathcal{R} &= \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{R}_1 & \dot{\mathcal{R}}_1 - A(\mathbf{p})\mathcal{R}_1 \end{bmatrix} \\ &= \begin{bmatrix} B(\mathbf{p}) & \dot{B}(\mathbf{p}) - A(\mathbf{p})B(\mathbf{p}) \end{bmatrix} \end{aligned} \tag{3.9}$$

Combining the equations stated in Section 3.1.1, the needed expressions of $A(\mathbf{p})$ and $B(\mathbf{p})$ can be found:

$$A = \begin{bmatrix} -2\alpha p_2 & -2\alpha p_1 \\ -2\alpha p_1 & -2\alpha \end{bmatrix} \tag{3.10}$$

$$B = \begin{bmatrix} 2\alpha p_1 \\ 2\alpha \end{bmatrix} \tag{3.11}$$

Because of what has been previously stated and since $\alpha = const.$, it can be concluded that $\dot{B}(\mathbf{p}) = 0$.

$$\begin{aligned} A(\mathbf{p})B(\mathbf{p}) &= -2\alpha \begin{bmatrix} p_2 & p_1 \\ p_1 & 1 \end{bmatrix} \begin{bmatrix} 2\alpha p_1 \\ 2\alpha \end{bmatrix} \\ &= -4\alpha^2 \begin{bmatrix} p_2 & p_1 \\ p_1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ 1 \end{bmatrix} \\ &= -4\alpha^2 \begin{bmatrix} p_1 p_2 + p_1 \\ p_1^2 + 1 \end{bmatrix} \end{aligned} \tag{3.12}$$

With the previous results, the reachability matrix can be written:

$$\mathcal{R} = 2\alpha \begin{bmatrix} p_1 & -2\alpha p_1(1+p_2) \\ 1 & -2\alpha(1+p_1^2) \end{bmatrix} \tag{3.13}$$

By expanding the matrix determinant:

$$\begin{aligned} |\mathcal{R}| &= -2\alpha p_1(1+p_1^2) + 2\alpha p_1(1+p_2) \\ &= 2\alpha(-p_1 - p_1^3 + p_1 + p_1 p_2) \\ &= 2\alpha(-p_1^3 + p_1 p_2) \end{aligned} \tag{3.14}$$

By plugging in the relationship between the parameters, i.e. $p_2 = p_1^2$, it can be proved that, no matter the values of α, p_1, p_2 the system is not structurally reachable. Looking back at Eq. (3.13) it can be seen how the first row is actually the second row multiplied by p_1, i.e. the two rows are not linearly independent, hence the matrix cannot be expected to be full rank. By computing the controllability matrix for some values of d and a fixed α, it can be proved that the rank of the reachability matrix is 1, thus it the Kálmán condition is not fulfilled.

3.1.4 Time-Dependent Structural Reachability Analysis

It might be significant to better understand the reachability property when the time-dependency of the parameter vector is considered, hence the assumption $\dot{\mathbf{p}} = 0$ is loosen while still keeping $\alpha = const.$ The time derivative of the parameter vector will be:

$$\begin{aligned} \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \end{bmatrix} &= \begin{bmatrix} \dot{d} \\ 2d\dot{d} \end{bmatrix}, \\ &= \begin{bmatrix} \dot{p}_1 \\ 2p_1\dot{p}_1 \end{bmatrix} \end{aligned}$$

Now, the structural reachability matrix $\mathcal{R}$ can be recomputed following Eq.(3.9).

$$\begin{aligned}\mathcal{R} &= \begin{bmatrix}\mathcal{R}_1 & \dot{\mathcal{R}}_1 - A\mathcal{R}_1\end{bmatrix}, \\ &= \begin{bmatrix}B & \dot{B} - AB\end{bmatrix}, \\ &= \begin{bmatrix}2\alpha p_1 & 2\alpha\dot{p}_1 + 4\alpha^2(p_1p_2 + p_1) \\ 2\alpha & 4\alpha^2p_1^2 + 4\alpha^2\end{bmatrix}, \\ &= 2\alpha\begin{bmatrix}p_1 & \dot{p}_1 + 2\alpha p_1(p_2+1) \\ 1 & 2\alpha(p_1^2+1)\end{bmatrix}\end{aligned} \tag{3.15}$$

By computing the determinant of the new reachability matrix:

$$|\mathcal{R}| = -\dot{p}_1 = -\dot{d} \tag{3.16}$$

This result is consistent to what obtained for the time independent case. This also means that the system is always reachable if the data provided to the system is monotonic. Alternatively the system is reachable for monotonic piece-wise data values.
A geometrical explanation of what is happening in the time-dependent case will be now offered. The determinant of the $\mathcal{R}$ matrix can be seen as the area spanned by the vectors $(2\alpha p_1, 2\alpha)$, $(2\alpha\dot{p}_1 + 4\alpha^2 p_1(p_1^2+1), 4\alpha^2(p_1^2+1))$ which are the columns of $\mathcal{R}$. As previously stated if $\dot{p}_1 = 0$, the two columns are linearly dependent, i.e. they are aligned if seen in the Cartesian plane. In this case the determinant would be zero, while for $\dot{p}_1 = \dot{d} \neq 0$, the area does not shrink to zero. Finally, it can be concluded that for $\dot{p}_1 = \dot{d} \to 0$, the area of the spanned parallelogram shrinks to zero.

3.1.5 LPV Structural Observability Analysis

An analysis of the structural observability will be now carried on in the same fashion as done for the study of the structural reachability.
A LPV system is said to be structurally observable iff the Kálmán full-rank condition is fulfilled for the matrix $\mathcal{O}$, this being:

$$\begin{aligned}\mathcal{O}_1 &= C(p(t)) \\ \mathcal{O}_{i+1} &= \dot{\mathcal{O}}_i + \mathcal{O}_i A(p(t)), \qquad i > 1\end{aligned} \tag{3.17}$$

For the current network, using the parameterization presented in Section 2.2, and keeping the assumption that $\dot{\mathbf{p}} = 0$, and recalling that the two parameters are related such that $p_1^2 = p_2$, the structural observability matrix becomes:

$$\begin{aligned}\mathcal{O} &= \begin{bmatrix}\mathcal{O}_1 \\ \mathcal{O}_2\end{bmatrix} \\ &= \begin{bmatrix}C(p(t)) \\ C(p(t))A(p(t))\end{bmatrix} \\ &= \begin{bmatrix}p_1 & 1 \\ -2\alpha p_1(p_2+1) & -2\alpha(p_1^2+1)\end{bmatrix} \\ &= \begin{bmatrix}p_1 & 1 \\ -2\alpha p_1(p_2+1) & -2\alpha(p_2+1)\end{bmatrix}\end{aligned} \tag{3.18}$$

In the last line of the latter equation, it can be seen that the two rows are linearly dependent, hence $\mathcal{O}$ is rank deficient no matter the value of α. It can be concluded that the system, while using $\mathbf{p} = \begin{bmatrix} d & d^2 \end{bmatrix}^T$ as parameter vector is not structurally observable.
Another important observation can be made while looking at the last step in Eq.(3.18). What can be concluded is that the assumption $p_1^2 = p_2$ is what makes the system structurally non-observable. If the latter property were not true, the $\mathcal{O}$ matrix would fulfill the Kálmán rank condition resulting in a structurally observable system. Furthermore what is being observed is that there might be a parameterization that makes the system structurally observable. On the other hand, it must also be pointed out that it is possible that such a parameterization is not able to describe the training dynamics.

3.1.6 Time-Dependent Structural Observability Analysis

In the same way as Section 3.1.4, the structural observability of the system will now be investigated while having time-dependent parameters. The structural observability matrix $\mathcal{O}$ will now be:

$$\begin{aligned}
\mathcal{O} &= \begin{bmatrix} \mathcal{O}_1 \\ \dot{\mathcal{O}}_1 + \mathcal{O}_1 A \end{bmatrix}, \\
&= \begin{bmatrix} C \\ \dot{C} + CA \end{bmatrix}, \\
&= \begin{bmatrix} p_1 & 1 \\ \dot{p}_1 - 2\alpha(p_1 + p_1 p_2) & -2\alpha(1 + p_1^2) \end{bmatrix}, \\
&= \begin{bmatrix} p_1 & 1 \\ \dot{p}_1 - 2\alpha p_1(1 + p_1^2) & -2\alpha(1 + p_1^2) \end{bmatrix}
\end{aligned}$$

Computing the determinant of $\mathcal{O}$:

$$|\mathcal{O}| = -\dot{p}_1 = -\dot{d}$$

This is the same result found in the structural reachability analysis with time-dependent parameter. In the same way, this means that, with the following parameter choice, the system is structurally observable if the data signal is monotonic or it can be piece-wise observable if only some portions of the data signal are monotonic. The exact same geometric comment written for the time-dependent structural reachability analysis can now be made for the time-dependent structural observability.

3.2 Single Neuron Linear Fractional Representation

Using the MATLAB® library GSS from the SMAC toolbox [4], it is possible to model the neuron with LFT.

By using the GSS library setting as input the nonlinear system, the matrices M, P can be obtained automatically. This second approach is used to check if a different parameterization, w.r.t. what has been manually computed and presented in Section 3.1.1, is proposed. Then the system stability is studied. Moreover this approach will be more suitable when larger ANNs are considered. After obtaining the matrices M, P by using the GSS library, the two systems are combined by using the MATLAB® built-in function *lft.m*, which performs the LFT of the two systems. Notice that an upper LFT must be performed, hence the order in which the systems are given as input must be switched, first P, then M.
Now the different parameterizations and their stability results will be presented.

3.2.1 $\alpha - d$ parameterized learning dynamics

This parameterization is the most general as both α, d are taken as uncertain parameters. According to the script, the M matrix, this being the matrix representing the system stripped of all the terms containing α, d, is:

$$M = \begin{bmatrix} M_{3,3} & M_{3,12} \\ M_{12,3} & M_{12,12} \end{bmatrix} \tag{3.19}$$

The matrices having the following dimensions:

Matrix	Rows Number	Columns Number
$M_{3,3}$	6	6
$M_{3,12}$	6	3
$M_{12,3}$	3	6
$M_{12,12}$	3	3

Table 3.4: $M_{i,i}$ matrices dimensions with d, α as parameters.

Analyzing the matrices' contents, in particular $M_{12,12}$, it turns out that the input vector of the matrix M is:

$$\zeta = \begin{bmatrix} u_{P,1} & u_{P,2} & u_{P,3} & u_{P,4} & u_{P,5} & u_{P,6} & w & b & y \end{bmatrix}^T \tag{3.20}$$

While the output vector is:

$$z = \begin{bmatrix} y_{P,1} & y_{P,2} & y_{P,3} & y_{P,4} & y_{P,5} & y_{P,6} & \dot{w} & \dot{b} & \hat{y} \end{bmatrix}^T \tag{3.21}$$

The relation between $u_{P,i}$ and $y_{P,i}$ is:

$$u_P = P y_P$$

The reason why, it was possible to identify the last state of the input vector is the position of the 1 in the matrix $M_{12,12}$. Furthermore, the structure of $M_{12,12}$ is the following:

$$M_{12,12} = \left[\begin{array}{c|c} A_{2\times2} & B_{2\times1} \\ \hline C_{1\times2} & D_{1\times1} \end{array}\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Thus, while using α, d as parameters, the nominal system A matrix is:

$$M.A|_{\alpha,d} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.22}$$

It can be concluded that the nominal plant has two identically zero eigenvalues. The other matrices are hereby reported for completion.

$$M_{3,3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -0.3961 & 0 & 0 \\ 0 & 0 & 0 & -0.0990 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1715 & -0.6860 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.4082 & 0 & 0 \end{bmatrix} \tag{3.23}$$

$$M_{3,12} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.6860 & -0.6860 \\ 0 & 0.1715 & 2.7440 \\ -1.7321 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.7071 & 0 \end{bmatrix} \tag{3.24}$$

$$M_{12,3} = \begin{bmatrix} -2.8284 & 0 & 0 & 0 & -1 & 0 \\ 0 & -2.9155 & -4.4409\text{e}-16 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5774 & 0 & 0 \end{bmatrix} \tag{3.25}$$

The matrices $M_{3,12}$ represents the state influence on the uncertainty and equivalently, $M_{12,3}$ represents how the uncertainty affect the state behaviour.

3.2.2 d parameterized learning dynamics

Setting d as parameter and a fixed value of α, i.e. $\alpha = 1e - 3$, while keeping the notation of Eq.(3.19), the dimensions of the system are:

Matrix	Rows Number	Columns Number
$M_{3,3}$	2	2
$M_{3,12}$	2	3
$M_{12,3}$	3	2
$M_{12,12}$	3	3

Table 3.5: $M_{i,i}$ matrices dimensions with d as parameter.

Ant the nominal A matrix is:

$$M.A|_{d} = \begin{bmatrix} 0 & 0 \\ 0 & -0.0020 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & -2\alpha \end{bmatrix} \tag{3.26}$$

The open loop dynamics will be now characterized by the eigenvalues:

$$\begin{aligned} \lambda_1 &= 0, \\ \lambda_2 &= -2\alpha \end{aligned} \tag{3.27}$$

Since α represents the learning rate, it is strictly positive. Hence the system has a stable eigenvalue and a marginally stable eigenvalue. Although for decreasing values of α, which typically guarantee convergence although at a slower rate, the system tends to have two zero eigenvalues. The other $M_{i,i}$ are reported for this different parameterization.

$$M_{3,3} = \begin{bmatrix} 0 & -0.0012 \\ 0 & 0 \end{bmatrix} \tag{3.28}$$

$$M_{3,12} = \begin{bmatrix} 0 & 0.0020 & -0.0020 \\ -1.7321 & 0 & 0 \end{bmatrix} \tag{3.29}$$

$$M_{12,3} = \begin{bmatrix} -1 & 0 \\ 0 & 0.0012 \\ 0 & -0.5774 \end{bmatrix} \tag{3.30}$$

In the previous sections, an investigation of how a single neuron could be modeled and analysed using tools from control theory was performed. This was done in order to both obtain results easier to interpret and to understand which tool would be the most appropriate. In industrial applications much more complex networks are used. In the next sections it will be shown how to model and then study the stability of such networks. In order to identify the individual issues caused by the increasing complexity, the following steps will be made:

- Study a two neurons - single layer network
- Study a single neuron - two layers network
- Study a two neurons - two layers network

It is worth to remind that saturation functions are omitted throughout this thesis.

3.3 Two Neurons Single Layer Network

The network can be depicted as follows:

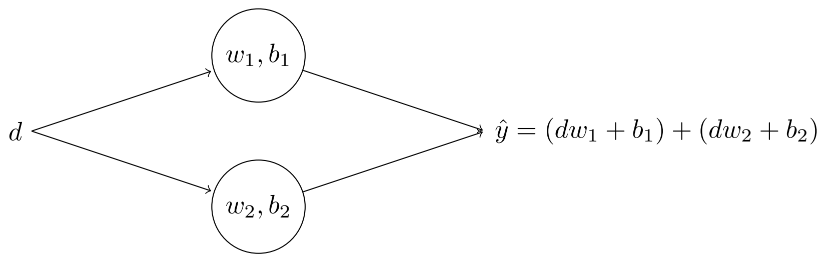

Figure 3.2: Two neurons one layer network.

The equation governing the previous networks can be obtained using Eq.(2.3), which results into:

$$\begin{aligned} \dot{w}_1 &= -2\alpha(d^2w_1 + d^2w_2 + db_1 + db_2 - dy) \\ \dot{w}_2 &= -2\alpha(d^2w_1 + d^2w_2 + db_1 + db_2 - dy) \\ \dot{b}_1 &= -2\alpha(dw_1 + dw_2 + b_1 + b_2 - y) \\ \dot{b}_2 &= -2\alpha(dw_1 + dw_2 + b_1 + b_2 - y) \end{aligned} \tag{3.31}$$

Setting the state vector as:

$$\mathbf{x} = \begin{bmatrix} w_1 & w_2 & b_1 & b_2 \end{bmatrix}^T \tag{3.32}$$

The training dynamics system can be transformed in state-space form:

$$\begin{aligned} A &= -2\alpha \begin{bmatrix} d^2 & d^2 & d & d \\ d^2 & d^2 & d & d \\ d & d & 1 & 1 \\ d & d & 1 & 1 \end{bmatrix}, \qquad B = 2\alpha \begin{bmatrix} d \\ d \\ 1 \\ 1 \end{bmatrix}, \\ C &= \begin{bmatrix} d & d & 1 & 1 \end{bmatrix} \end{aligned} \tag{3.33}$$

What can be concluded is that adding a neuron to a one layer network does not introduce new nonlinearities.

3.3.1 $\alpha - d$ parameterized learning dynamics

The results obtained while using d, α as parameters are the same with obtained in Subsection 3.2.1.The LFR dimensions are:

Matrix	Row Number	Column Number
$M_{3,3}$	4	4
$M_{3,12}$	4	5
$M_{12,3}$	5	5
$M_{12,12}$	5	5
P	4	4

Table 3.6: $M_{i,i}$ matrices dimensions with d, α as parameters.

Where the uncertainty matrix is:

$$P = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & d \end{bmatrix} \tag{3.34}$$

It can be seen that the although the nonlinearities are the same, the P block synthesized for this network is different from the previous one. This should not raise any concerns as infinitely-many linear fractional representations may exist.
Finally, the A matrix of the LTI system, i.e. $M.A$, is identically zero, hence its eigenvalues are all zero. Although $M.A$ has a higher dimensions w.r.t. the single neuron case, the results found show consistency. The matrices $M_{i,i}$ will now be reported.

$$M_{12,12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.35}$$

$$M_{3,3} = \begin{bmatrix} 0 & 0 & 0.6325 & -1.5543\text{e}-16 \\ 0 & 0 & -3.0355\text{e}-16 & -1.5811 \\ 0 & 0 & 1.0757\text{e}-16 & -1.9178\text{e}-32 \\ 0 & 0 & 0.4000 & -7.1312\text{e}-17 \end{bmatrix} \tag{3.36}$$

$$M_{3,12} = \begin{bmatrix} 0 & 0 & 1.5811 & 2.2551\text{e}-16 \\ 1 & -1.7234\text{e}-16 & 1.7009\text{e}-16 & 0.6325 \\ 0 & 0 & 1.5811 & -2.2551\text{e}-16 \\ 1 & -1.7234\text{e}-16 & 1.7009\text{e}-16 & 0.6325 \\ -1 & -3.4468\text{e}-16 & 2.9123\text{e}-16 & -0.6325 \end{bmatrix}^T \tag{3.37}$$

$$M_{12,3} = \begin{bmatrix} 2.6999\text{e}-16 & 2 & 0 & 0 \\ -2 & -3.2935\text{e}-16 & 0 & 0 \\ 2.6999\text{e}-16 & 2 & 0 & 0 \\ -2 & -1.5329\text{e}-17 & 0 & 0 \\ 0 & 0 & 0.6325 & 0 \end{bmatrix} \tag{3.38}$$

3.3.2 *d* parameterized learning dynamics

Analogously to what is done for the parameterization which uses both α and d, a similarity with Subsection 3.2.2 is found. The system has the following dimensions:

Matrix	Rows Number	Columns Number
$M_{3,3}$	2	2
$M_{3,12}$	2	5
$M_{12,3}$	5	2
$M_{12,12}$	5	5
P	2	2

Table 3.7: $M_{i,i}$ matrices dimensions with d as parameter.

The P block being:

$$P = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix} \tag{3.39}$$

Thanks to how the state vector was ordered an interesting property can be observed from looking at the $M.A$ matrix;

$$M.A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -0.0020 & -0.0020 \\ 0 & 0 & -0.0020 & -0.0020 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -2\alpha & -2\alpha \\ 0 & 0 & -2\alpha & -2\alpha \end{bmatrix} \tag{3.40}$$

For this formulation the LTI system's A matrix is block diagonal. Moreover, it is constituted of a $0_{2\times2}$ matrix and of a $[2 \times 2]$ matrix having all entries equal to -2α, which is rank deficient. The eigenvalues are then, the following:

$$\begin{aligned} \lambda_1 &= -0.0040 = -4\alpha \\ \lambda_2 &= 0 \\ \lambda_3 &= 0 \\ \lambda_4 &= 0 \end{aligned} \tag{3.41}$$

Finally, the trend shown in this section can be extended to the case of a single layer with n-many neurons. Since adding neurons to a single line does not introduce new nonlinearities the results obtained are the same with the only exception of the increased dimensions. Hence, eigenvalues shows Lyapunov stable learning dynamics.

$$M_{12,12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -0.0020 & 0 & -0.0020 & 0.0020 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -0.0020 & 0 & -0.0020 & 0.0020 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.42}$$

$$M_{3,3} = \begin{bmatrix} 0 & -0.4000 \\ 0 & 0 \end{bmatrix} \tag{3.43}$$

$$M_{3,12} = \begin{bmatrix} 0 & 0.6325 & 0 & 0.6325 & -0.63258 \\ -1.5811 & 0 & -1.5811 & 0 & 0 \end{bmatrix} \tag{3.44}$$

$$M_{12,3} = \begin{bmatrix} -0.0032 & 0 \\ 0 & 0.0013 \\ -0.0032 & 0 \\ 0 & 0.0013 \\ 0 & -0.6325 \end{bmatrix} \tag{3.45}$$

3.4 Single Neuron Two Layers Network.

Adding layers to a network causes one main issue. The output signal from the first perceptor is carries informations about the weight and the bias, which are the system states in the NTK formulation. This causes an interaction between different states, which is something not desirable. We want to avoid this as much as possible because we would want to have a non endogenous, i.e. not depending on the system variables, parameter vector.

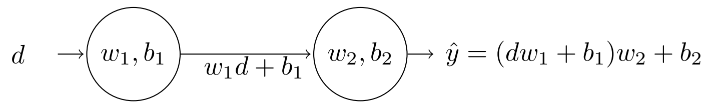

Figure 3.3: Single neuron two layers network.

One possible solution would come from the fact that the exact value of the weights and biases of the entire network for a certain training iteration are known. The idea is to see the output from the first perceptor, or layer as it will be later explained, $\hat{y}_1 = dw_1 + b_1$ as a real uncertain parameter. This is possible because, as previously said, the values of network hyperparameters are known. The parameter $\hat{y}_1$ would then have the following properties:

Parameter	Type	Nominal Value	Minimum Value	Maximum Value
$\hat{y}_1$	Real	$d_{nom}w_1 + b_1$	$d_{min}w_1 + b_1$	$d_{max}w_1 + b_1$

Table 3.8: Second layer uncertain input.

For the current network, the LFR would then be similar to the analysis presented in Subsection 3.2.1, as the only difference would be the nominal value and the range of the uncertainty. It must be pointed out that the results obtained in Subsection 3.2.1 held for different ranges of d and different values of α. In order to further clarify this point, the next figure, representing the second layer, which is constituted only from a neuron in this case, is presented:

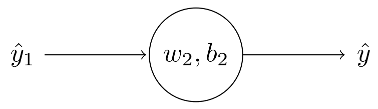

Figure 3.4: Reformulation of the second layer.

This approach is promising as if we were to add more neurons to each layer, each neurons' output would not add any nonlinearity to the next layer while being linearly combined to the other outputs coming from the precedent layer. The issue rising in this approach is that the output signal of the i-th layer, $\hat{y}_i$, must be updated with the current weight and bias of the i-th layer following the equations presented in Tab. 3.8. The presented algorithm is able to solve this problem for a generic single neuron multi layer network.

Algorithm 1 LFR.

```
1: procedure LINEAR FRACTIONAL REPRESENTATION
2:     x0 = CURRENT VALUES OF THE NETWORK WEIGHTS AND BIASES
3:     $N$ = NUMBER OF LAYERS
4:     $\hat{y}_1 = d$, $d$ IS THE DATA GIVEN TO THE NETWORK
5:     for $i = 1 : N - 1$ do
6:         COMPUTE LFR FOR A SINGLE LAYER WITH INPUT DATA $y_i$
7:         $\hat{y}_{i+1} = w_i \hat{y}_i + b_i$, UPDATE OUTPUT SIGNAL
8:         SAVE DIFFERENT SIGNALS
```

Now that all the different input signals are available, the entire network can be re-built using a LFR by building i-many A matrices and then by using the function *append.m* which has been rebuilt in the SMAC Toolbox [4]. The reason why the previous algorithm was developed was to avoid having endogenous parameters. What was done, is exploiting the fact that, for the current learning iteration, the entire network's weights and biases are known. And since the training dynamics of the i-th layer neuron is not influenced by the (i+1)-th layer neuron, when going from i-th layer to the (i+1)-th layer it is possible to "update" the signal using the following relation:

$$\begin{aligned} &\hat{y}_{i+1} = \hat{y}_i w_i + b_i, \quad \text{for i = 1 : N - 1} \\ &\hat{y}_1 = d \\ &\hat{y}_N = \hat{y}_{N-1} w_{N-1} + b_{N-1} = \hat{y} \end{aligned} \tag{3.46}$$

If specialized for the two layers case:

$$\hat{y} = (d w_1 + b_1) w_2 + b_2 \tag{3.47}$$

After this, the stability analysis can be brought back to the results presented in Section 3.3.

3.5 Two Neurons Two Layers Network

In order to achieve a full LFR of the neural network some slight modifications w.r.t. what was presented in Section 3.3 must be performed. These are namely a change of the state vector. For the i-th layer it will now be:

$$\mathbf{x}_i = \begin{bmatrix} w_{i,1} \\ b_{i,1} \\ w_{i,2} \\ b_{i,2} \\ \vdots \\ w_{i,M} \\ b_{i,M} \end{bmatrix} \tag{3.48}$$

Then, the estimated label equation will be computed using a different solution. Taking into account the i-th layer, what only matters is the sum of each neurons' output. Hence, instead of computing M-many outputs, where M is the number of neurons in each layer which is known, only the sum of them will be computed. This also imply a significant reduction in the dimension of the C_i matrix that becomes $[1 \times 2M]$ while the original matrix dimensions were $[M \times 2M]$. The equation which enables the computing the new signal coming out of each layer is:

$$\begin{aligned} \hat{y}_i &= \sum_{1}^{M}{}_{j}(dw_{i,j} + b_{i,j}) \\ &= (dw_{i,1} + b_{i,1}) + (dw_{i,2} + b_{i,2}) + \ldots + (dw_{i,M} + b_{i,M}) \\ &= \begin{bmatrix} d & 1 & d & 1 & \ldots & d & 1 \end{bmatrix} \mathbf{x}_i \end{aligned} \tag{3.49}$$

The latter equation computes the sum of the signals going out of all the neurons belonging to the j-th layer and going into each neuron of the following neuron. The approach used to model more complex networks will be to obtain each layer's linear fractional representation and then combining them by using the GSS library. This way of modeling the problem has two main advantages. The first one is that, given the number of neurons per layer and assuming it is kept constant throughout the layers, the equations only have to be defined one time. Secondly, only one LFR needs to be explicitly computed. This is because, as seen in Section 3.3, the LFR structure is always the same and the only changing quantities are the signals $\hat{y}_i$, defined as in Eq.(3.49). Because of these two reasons, the modelling problem becomes suitable and trivial for automation. This, given the dimensions of a real ANN, is an extremely important property. One last comment must be made on the parameterization choice. The previously described approach can be exploited even while changing the parameterization choice layer by layer. Moreover, the code developed is able to take as input a certain parameterization choice vector and then construct the appropriate LFR. Given the new parameterization, some of the stability results presented in Section 3.3 need to be reformulated because of the new state vector.

3.5.1 Stability Analyses

The equations for the i-th layer with the new specified vector becomes:

$$\begin{aligned}
\dot{w}_{i,1} &= -2\alpha(\hat{y}_{i-1}^2 w_{i,1} + \hat{y}_{i-1} b_{i,1} + \hat{y}_{i-1}^2 w_{i,2} + \hat{y}_{i-1} b_{i,2} - \hat{y}_{i-1} y_i) \\
\dot{b}_{i,1} &= -2\alpha(\hat{y}_{i-1} w_{i,1} + b_{i,1} + \hat{y}_{i-1} w_{i,2} + b_{i,2} - y_i) \\
\dot{w}_{i,2} &= -2\alpha(\hat{y}_{i-1} w_{i,1} + \hat{y}_{i-1} b_{i,1} + \hat{y}_{i-1} w_{i,2} + \hat{y}_{i-1} b_{i,2} - \hat{y}_{i-1} y_i) \\
\dot{b}_{i,2} &= -2\alpha(\hat{y}_{i-1} w_{i,1} + b_{i,1} + \hat{y}_{i-1} w_{i,2} + b_{i,2} - y_i) \\
\hat{y}_i &= (d_i w_{i,1} + b_{i,1} + d_i w_{i,2} + b_{i,2}) \\
&\text{for } i = 1 : N \quad \text{with } \hat{y}_0 = d
\end{aligned} \tag{3.50}$$

A slight notation modification has been performed in order to be able to consider any layer inside a network, not only the first one. This is possible because, since for the current learning iteration all weight and biases are known, all signals between different layers can be constructed.
The linear fractional representation of the layer dynamics can now be built. The GSS-object representing the layer can be turned into an USS-object which can be used to perform stability analyses accordingly to different approaches. Using the Robust Control Toolbox^{TM} the USS-object can be split into its $M-P$ components as specified in Fig. 2.2. The system can be represented as:

$$\left[\begin{array}{c} y_P \\ \hline \dot{\mathbf{x}} \\ \hat{y} \end{array}\right] = \left[\begin{array}{c|c} M_{1,1} & M_{1,2} \\ \hline M_{2,1} & M_{2,2} \end{array}\right] \left[\begin{array}{c} u_P \\ \hline \mathbf{x} \\ y \end{array}\right]$$

Where:

$$u_P = P y_P \tag{3.51}$$

By using this representation, it is possible to see how the uncertainties affect both the state and the estimation of the label through the layers. The matrix $M_{2,2}$ is constant for all layers as the training dynamics through the network's layers is always the same. What changes from layer to layer is the data going into each layer. This signal will change both in terms of nominal value and range accordingly to Eq.(3.46).

3.6 Controlled Learning Dynamics

3.6.1 Control Structure

The control of the network will be achieved by injecting fake label as inspired by [3]. The fake label will be called y_f. The injection must happen on the estimation signal as, if it was done before a neuron, the same results obtained for an open loop single neuron single layer network would be obtained. This would be totally useless since it was proven that there is not a stable region. Hence, the controlled network would have a structure like this:

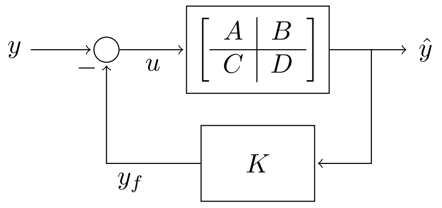

Figure 3.5: Block diagram of the control scheme for a single neuron.

The training equations must now be derived while using the updated loss function:

$$\begin{aligned}\mathcal{L}_c(t) &= (y - y_c(t))^2 \\ &= (y - \hat{y}(t) - y_f(t))^2 \\ &= (y - (dw(t) + b(t)) - y_f(t))^2\end{aligned} \tag{3.52}$$

Computing the gradient of the new loss function, i.e. ∇L_c:

$$\begin{aligned}\frac{\partial \mathcal{L}_c}{\partial w}(t) &= \frac{\partial}{\partial w}(y - y_f(t) - dw(t) - b(t))^2 \\ &= -2d(y - y_f(t) - dw(t) - b(t)), \\ \frac{\partial \mathcal{L}_c}{\partial b} &= \frac{\partial}{\partial b}(y - y_f(t) - dw(t) - b(t))^2 \\ &= -2(y - y_f(t) - dw(t) - b(t))\end{aligned} \tag{3.53}$$

The controlled training equations can thus be written as $\dot{x} = -\alpha \nabla L_c x$.

$$\begin{aligned}\dot{w}(t) &= 2\alpha d(-dw(t) - b(t) + y - y_f(t)) \\ \dot{b}(t) &= 2\alpha(-dw(t) - b(t) + y - y_f(t))\end{aligned} \tag{3.54}$$

The system can be re-formulated in the state-space as:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \begin{bmatrix} B & B_c \end{bmatrix} \begin{bmatrix} y \\ y_f(t) \end{bmatrix} \tag{3.55}$$

From Eq.(3.53) and Eq.(3.54) it can be seen that $B = -B_c$. Because the label y is constant and cannot be manipulated to enhance the system performances, it shall not be used as a control signal. Then, the system will only be controlled by the fake label signal, i.e.:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B_c y_f(t) \tag{3.56}$$

The control input matrix will be taken as $B_c = B$, by assuming the loss function to be $\mathcal{L}_c = (y - \hat{y}(t) + y_f(t))^2$.

3.6.2 Gain Scheduling Linear Quadratic Regulator

The first attempt at controlling a single neuron training dynamics will be made by using optimal control to synthesize a gain scheduling LQR. For what was previously said, the assumption $\alpha = const.$ will be made. In order to account for the different values of the parameter d in the training dynamics equations, the system in Eq.(3.55) is sampled at different values within the range of d and then an attempt at synthesizing a linear quadratic regulator for each value of d is made. Because of the dimensions of the problem, each controller K has dimensions $[1 \times 2]$. Then, an interpolation amongst each K component is made to obtain a gain scheduling controller, i.e. a controller which has as components polynomial functions with d as variable. In this thesis the interpolating polynomial will have order 4.

$$K(d) = \begin{bmatrix} a_0 + a_1 d + a_2 d^2 + a_3 d^3 + a_4 d^4 \\ b_0 + b_1 d + b_2 d^2 + b_3 d^3 + b_4 d^4 \end{bmatrix}^T \tag{3.57}$$

In order to be able to use the MATLAB® built-in function *lqr.m*, which uses optimal control theory to synthesize a LQR for the given system, it is necessary to remove the identically zero pole of the A matrix. This pole will always be in zero since the A matrix is non full rank. The pole is then moved in $p_{new} = -10 \times \epsilon$, where $\epsilon \approx 2.204e - 16$ is the MATLAB® machine precision. Although this small change does not impact the system dynamics, it overcome the numerical issue within the software. It must be noticed that the ϵ multiplying factor might change if a different MATLAB® version is used. In this thesis the MATLAB® version used was R2020b Update 5. By using the approach previously described, the following gain scheduling optimal controller is synthesized:

$$K(d)|_{N=4} = \begin{bmatrix} d^4 \\ d^3 \\ d^2 \\ d \\ 1 \end{bmatrix}^T \begin{bmatrix} 8.0728\text{e}-06 & 0.1248 \\ -0.1711 & -3.1915\text{e}-06 \\ -5.6174\text{e}-06 & -0.3073 \\ 0.3837 & 1.3731\text{e}-06 \\ 4.8062\text{e}-07 & 0.4123 \end{bmatrix} \tag{3.58}$$

Now the plot of the exact values of the synthesized controllers' components against the interpolated values will be presented in order to prove the applicability of the method.

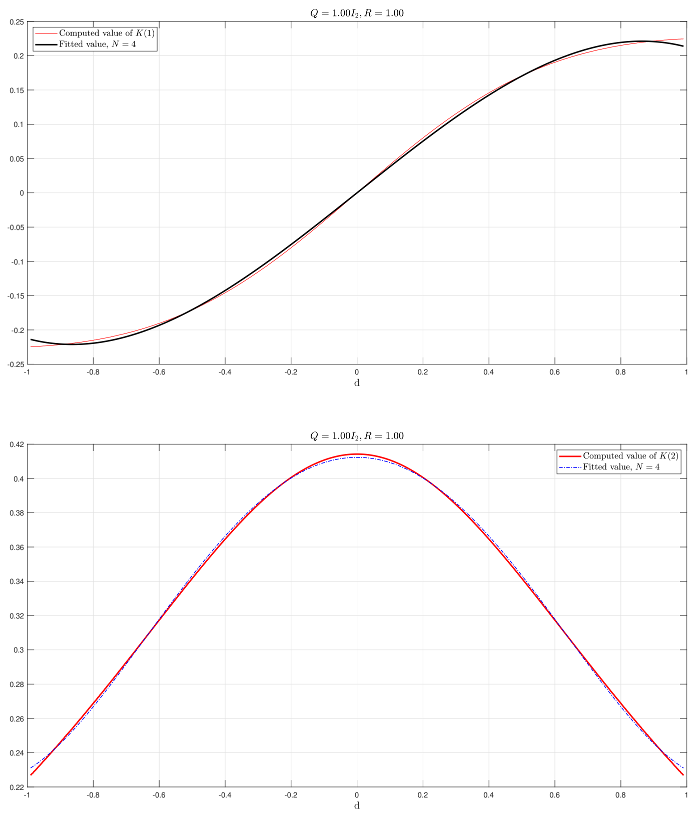

Figure 3.6: Comparison between the interpolated polynomials for the first component of the gain scheduled LQR and the actual synthesized values.

To obtain the gain scheduling controller presented in Eq.(3.58), the state and control input were assumed to be:

$$
\begin{aligned}
Q &= I_2 \\
R &= 1
\end{aligned}
\tag{3.59}
$$

In the next sections both the effect of changing the weights value is analyzed. The study is first carried out for $K(1)$ and then for $K(2)$.

Effects on $K(1)$

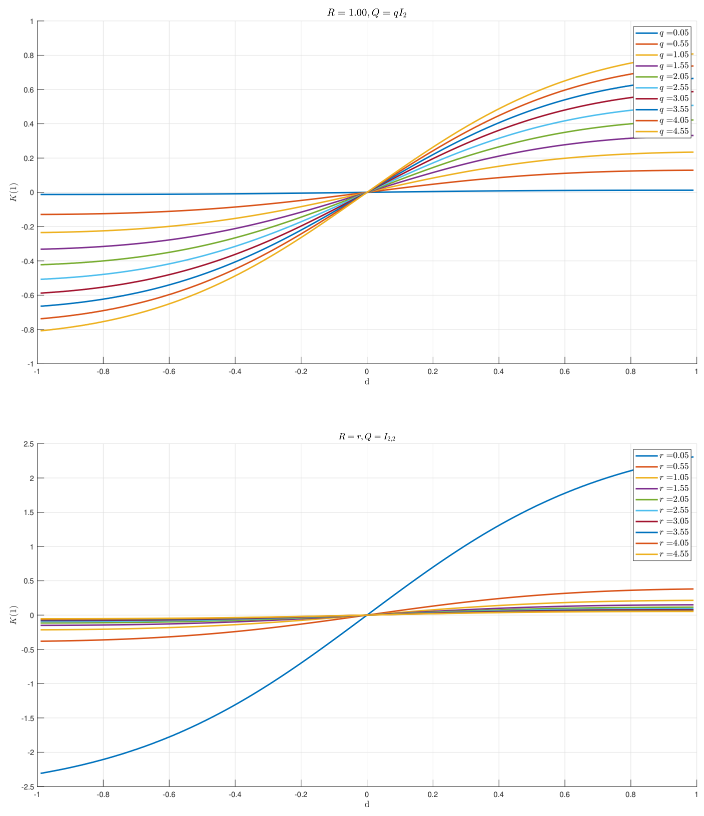

Figure 3.7: Behaviour of the first component of gain scheduling controllers synthesized for different values of both Q and R.

Increasing the penalisation on the states while keeping R constant has the same effect of decreasing R, which means that an inexpensive controller is being synthesized, hence a high gain behaviour is mimicked. Increasing values of q accentuates the effects of the nonlinear terms in Eq.(3.58). What can be observed is that, no matter the penalisation, for negative values d the first term of the controller will be negative. A conclusion that can be drawn from this study is that, controller synthesized for different conditions of Q and R are not interchangeable. This is because for the first term of the controller there is a significant change in the value of $K(1)$ for different weights.

Effects on $K(2)$

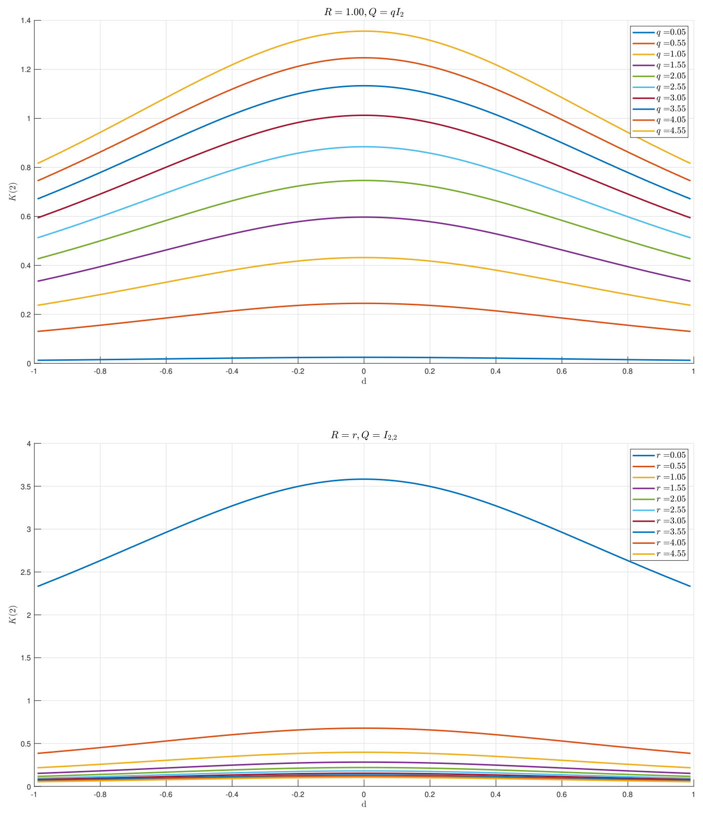

Figure 3.8: Behaviour of the second component of gain scheduling controllers synthesized for different values of both Q and R.

Increasing the penalization on the states, i.e. designing an inexpensive controller, causes a change in the curve shape. As it can be seen in the figure, the bell of $K_2(d)$ gets more accentuated as q increases. Finally, the distance between different curves decreases as q increases. A similar conclusion can be drawn while varying R. Less expensive controllers, e.g. the controller synthesized for $R = 0.05$, shows an accentuated bell shape w.r.t. other controllers. However, the curves get flatter quite fast, as an increase of the control input weight to $R = 0.55$, makes the curve significantly flatter.

3.6.3 Closed Loop Simulation

It shall now be showed how the system performances are affected by the gain scheduling controller.

First the plots of system controlled by a gain scheduling LQR synthesized for $Q = I_2,\ R = 1$ will be shown. Then the effect of $Q,\ R$ will be investigated.

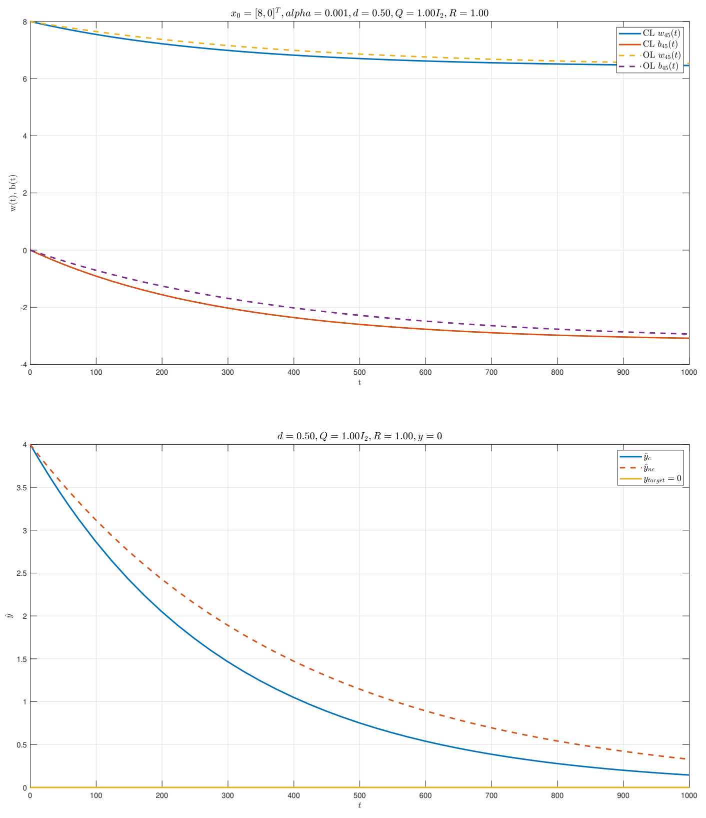

Figure 3.9: States and estimated label behaviour w.r.t. time.

As it may be seen the system converges faster but there are no significant differences, this holds for different values and combinations of d, α, $\mathbf{x}_0$. The main issue of this controller is that the control input, i.e. $u(t)$, is being restricted by the weight R. If the penalization on the input is lowered, an equivalent effect can be achieved by increasing the penalization on the state vector although it is less intuitive, much better performances can be achieved.

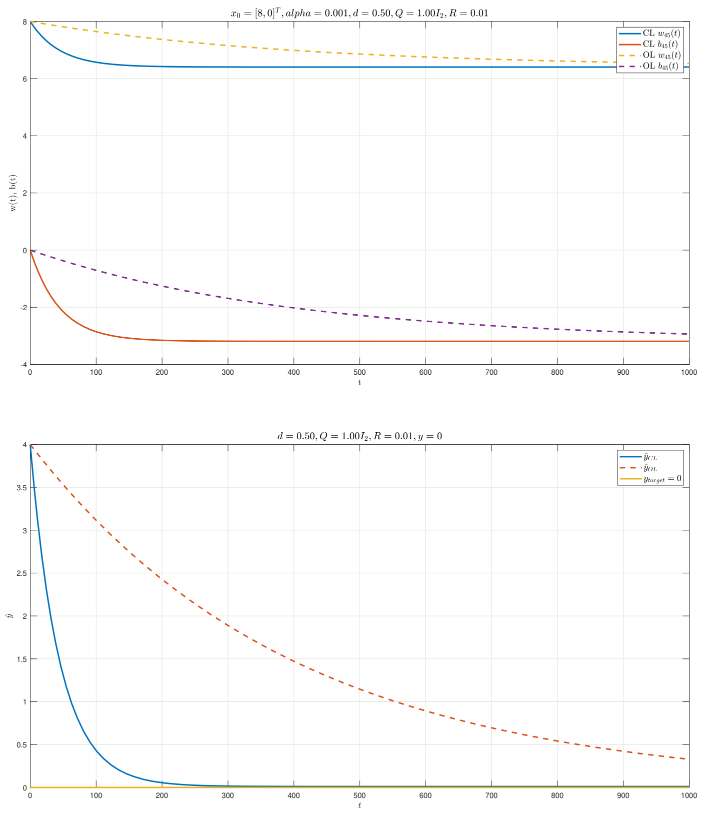

Figure 3.10: States and estimated label behaviour w.r.t. time.

It can be easily noticed that performances are much improved by lowering the value of R, i.e. synthesizing less expensive controllers.

A few words must be spent on how the weights have been chosen so far, and how they might be chosen in future applications. Usually, the weights Q, R are used to prevent the system states vector and control inputs from reaching values for which the system presents an undesirable behaviour, i.e. they might be used for constraining the states trajectory and control input magnitude. This especially applies for mechanical systems where large control inputs may be undesirable because of the actuator's dynamics. The problem studied in this thesis does not present physical limitations, neither in terms of states nor as control inputs. Because of this, the controller can be given a higher authority by setting a lower value of R. Finally, although lowering the values of R provide better performances, it might be helpful to find an optimum operation point for which the system's performances are enhanced by the controller but the computational burden does not get too large.

Finally, it will be attempted to use an inexpensive controller to show the impossibility of tracking the label.

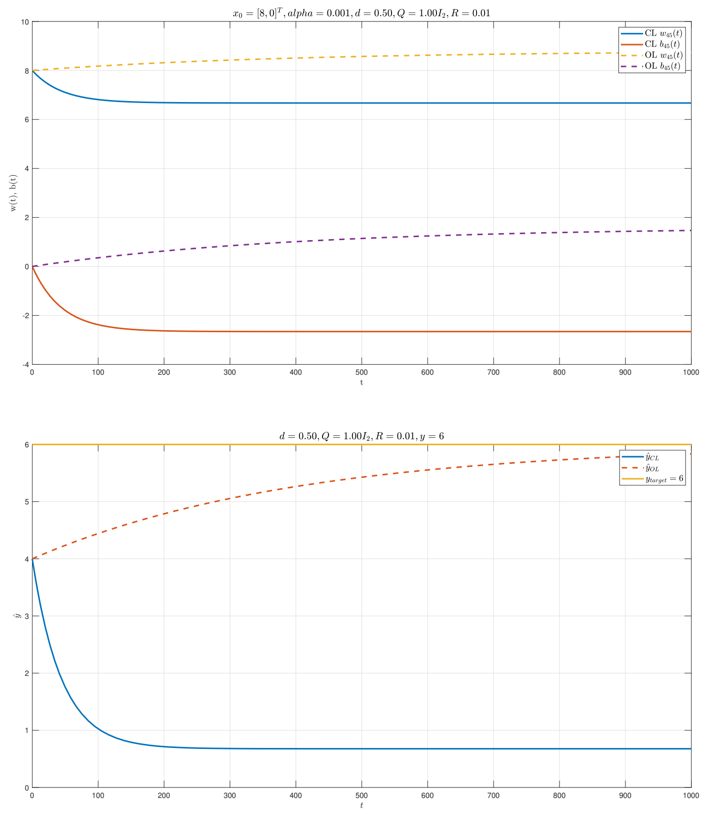

Figure 3.11: States and estimated label behaviour w.r.t. time using an inexpensive controller, $y = 6$.

The non-robustness of the LQR can be overcome in two different ways.

The first one is to account for the offset by modifying the fake label in the following way:

$$y_{f,new}(t) = -K(x(t) - x_{ss}) \tag{3.60}$$

Where x_{ss} is the states' steady-state value, x_{ss} can be calculated as:

$$x_{ss} = C^{+}y \tag{3.61}$$

The symbol C^{+} representes the pseudo-inverse of the matrix C. This is required since C is not square, hence the pseudo inverse must be used. This operation is performed in MATLAB® by using the command \. The closed loop with $y \neq 0$ then becomes:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B(-K(x(t) - x_{ss})) + By, \\ &= (A - BK)x(t) + B(y + Kx_{ss}) \end{aligned} \tag{3.62}$$

As it will now be shown by the plots, this slight modifications solve the offset issue.

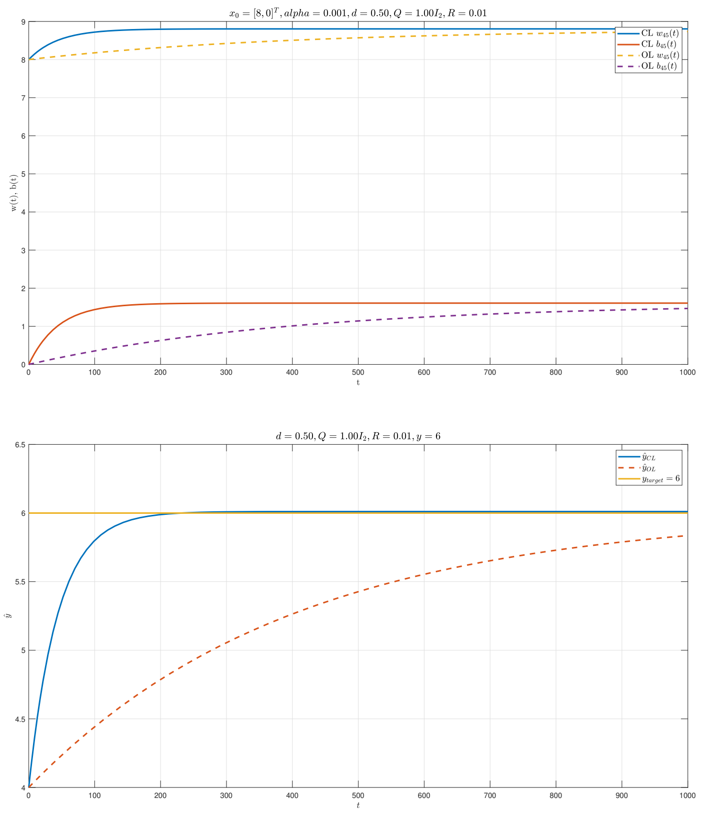

Figure 3.12: States and estimated label behaviour w.r.t. time using an inexpensive controller and accounting for non-zero label, $y = 6$.

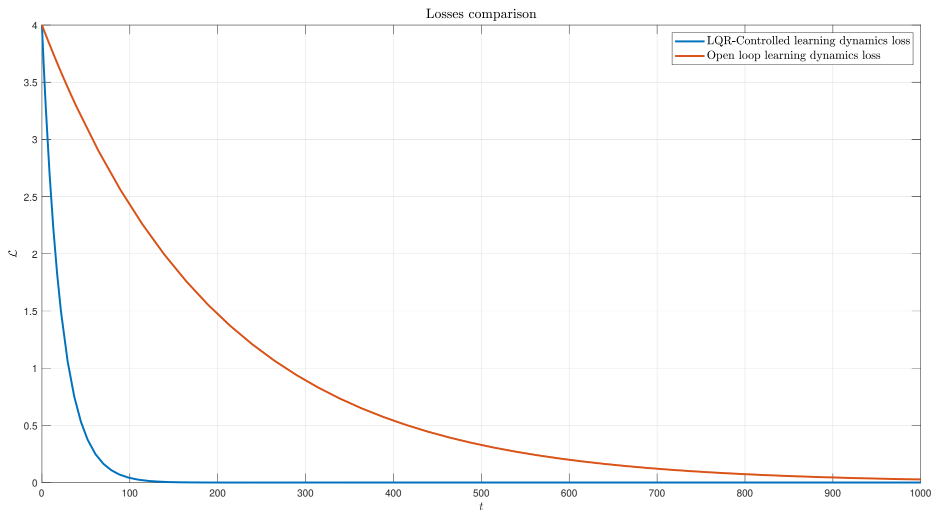

Figure 3.13: Loss behaviour w.r.t. time of the LQR-controlled learning dynamics compared to the behaviour of the non-controlled learning dynamics, $y = 6$.

A brief discussion is needed in order to fully explain what price has been paid to solve the offset problem for the LQR. Eq.(3.61) was used to obtain the asymptotic values of the states. Because of the dimensions of C, the matrices could not be simply inverted, instead a pseudo-inversion was performed. Moreover, the problem set was undetermined, this meaning the problem was having one equation and two variables. Because this is a non well-posed problem it might be desirable to avoid it. In the next section an alternative control structure, the LQI regulator which is a robust controller obtained from the optimal control theory, will be presented in order to avoid this issue.

3.6.4 Gain Scheduling Linear Quadratic Integral Regulator

The gain scheduling LQI controller is synthesized in an analogous way to the LQR version. The following matrix $K_i(d)$ is found:

$$K_i(d) = \begin{bmatrix} d^4 \\ d^3 \\ d^2 \\ d \\ 1 \end{bmatrix}^T \begin{bmatrix} -3.8303\text{e}-05 & 9.0797 & 4.3789\text{e}-06 \\ -15.1933 & -6.2800\text{e}-06 & 4.6710\text{e}-07 \\ 3.7191\text{e}-05 & -25.1009 & -3.2519\text{e}-06 \\ 53.1207 & 1.0235\text{e}-05 & -7.2872\text{e}-07 \\ -2.6472\text{e}-06 & 55.2057 & -3.1623 \end{bmatrix} \tag{3.63}$$

Two comments can be made on the latter results. The first one is that, similarly to what has been seen before, the element $K_i(1)$ is mostly influenced by the polynomial terms with even exponent, while $K_i(2)$ is mostly influenced by the odd-exponent terms. The second observation is that the controller term related to the integrative variable does not depend on the d value as the terms related to the polynomial terms with non-zero exponent are in the order of $\approx 10^{-6}$. The following plots describe the closed loop simulations. The case where $y = 6$ is immediately studied. Similarly to the LQR case, a less expensive controller guarantees better performances. In this thesis the LQI gain scheduling controller for $R = 0.01$ is shown.

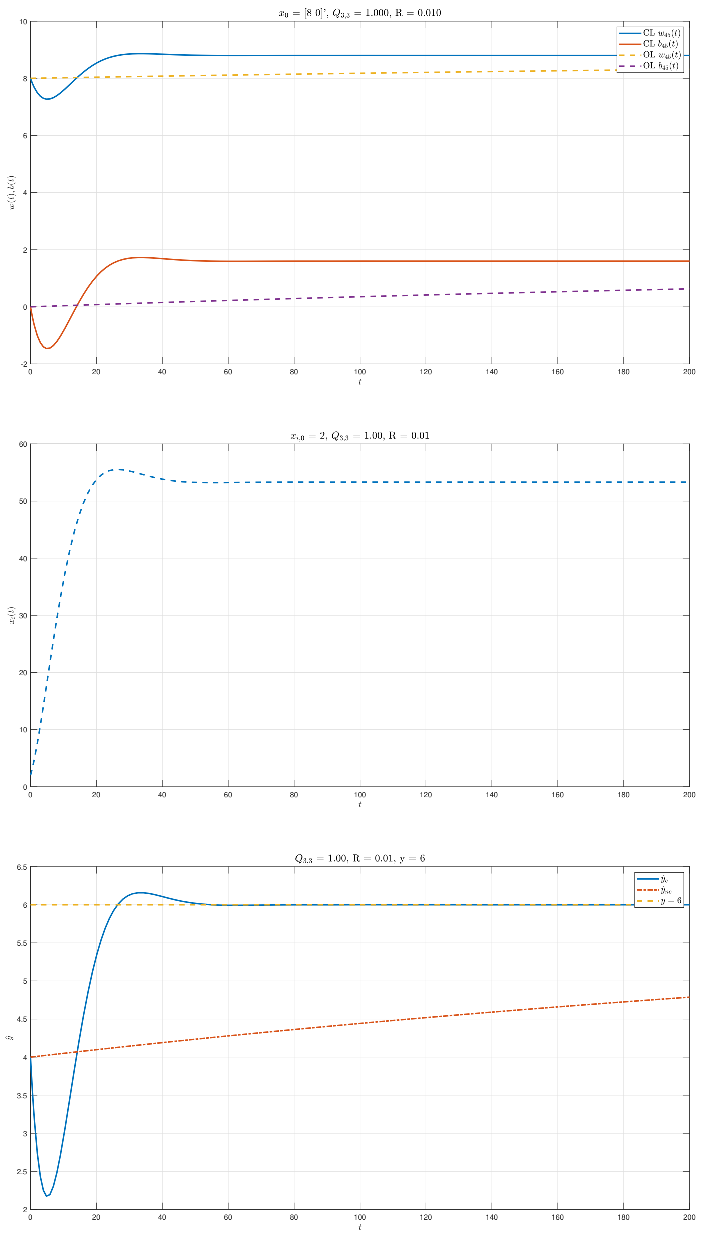

Figure 3.14: States and estimated label behaviour w.r.t. time using an inexpensive LQI controller, $y = 6$.

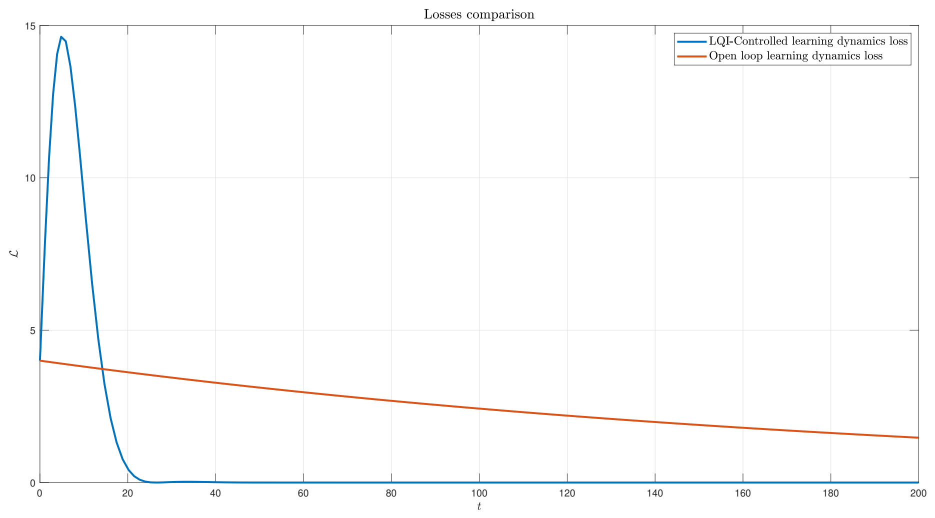

Figure 3.15: Loss behaviour w.r.t. time of the LQI-controlled learning dynamics compared to the behaviour of the non-controlled learning dynamics, $y = 6$.

From the plots it can be concluded that the LQI gain scheduling controller guarantees even better performances while accounting for the disturbance-like effect caused by a non-zero label. Finally, from the loss plot, it can be seen how the loss value also exhibits a peak due to the initial oscillations.

3.6.5 On the influence of initial conditions

Before concluding the analysis of the system controlled by an optimal regulator, the influence of the initial conditions must be considered. This final investigation will be run only for the LQI regulator. The reason for this is because, in order to perform this analysis, the original LPV system will be reduced to a LTI sytem for an arbitrary combination of α, d. During the development of this analysis different combinations of values were tested and similar results were found. This was done in order to check the consistency of the analysis over the whole domain of the learning dynamics.
During the testing of the synthesis script different combinations of $\mathbf{x}_0$ and y have been tested in order to avoid any malfunctioning. In addition to a greater speed of convergence, for certain values of $\mathbf{x}_0$ and y an initial unexpected oscillation for the system controlled by the LQI regulator was observed. .
The unexpected oscillation is similar to those exhibited by non-minimum phase systems, i.e. systems with at least one zero in the RHP. In order to verify this property, it is necessary to find the transfer function of the system. For the closed loop system presented in Eq. (2.29) the transfer function is:

$$\frac{\hat{y}}{y}(s) = \begin{bmatrix} C & 0 \end{bmatrix} \left(sI_3 - \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} B \\ 1 \end{bmatrix} \tag{3.64}$$

For the controller synthesized for $Q = I_3,\ R = 0.1,\ d_{nom} = 0.5$, the transfer function has the following zeros:

Zero ID	Zero values
1	-3.1623
2	-0.0631 + j0.0630
3	-0.0631 - j0.0630
4	-0.0634 + j0.0628
5	-0.0634 - j0.0628
6	-0.0628 + j0.0628
7	-0.0628 - j0.0628
8	-0.0633 + j0.0624
9	-0.0633 - j0.0624
10	-0.0628 + j0.0624
11	-0.0629 - j0.0624
12	3.3171e-15
13	3.3176e-15
14	-1.7246e-16
15	-1.6834e-16
16	3.7922e-17
17	3.4937e-17

Table 3.9: Transmission zeros.

As it can be seen in the previous table, the transmission zeros 12,13,16,17 belongs to the RHP. Thisnmeans that the closed loop system is either non minimum phase or the transmission zeroes are illposed. But, the last two, i.e. z_{16}, z_{17} cannot be considered as they are smaller than MATLAB® machine precision, i.e. $\epsilon \simeq 2.2e{-}16$. As seen in the previous table there are RHP zeros which may cause the oscillations, nevertheless this may not be the reason behind the undesirable behaviour since the magnitude of the RHP zeros is very close to the MATLAB® machine precision.

Using a gain-scheduled LQI controller for the previously stated parameters, some time simulations for different initial conditions will now be run. Since the initial oscillations is present both in the $w(t)$, $b(t)$ and $\hat{y}(t)$ only the estimated label will be reported for clarity reasons. Different labels will also be tested.

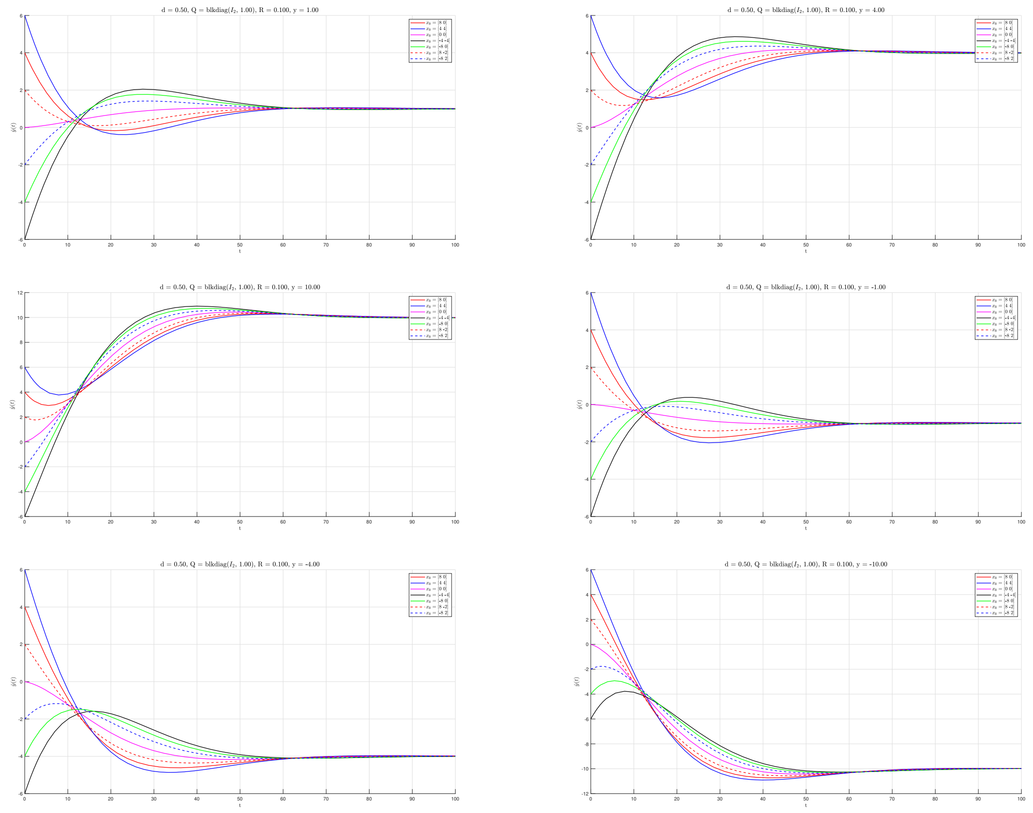

Figure 3.16: $\hat{y}(t)$ behaviour for the different labels.

What can be learnt from this plots is that the system will experience a larger initial oscillation, that may be due to the system being non-minimum phase, if the initial guess of the weight and bias are closer to the asymptotic value. The same applies to the estimated label.

3.6.6 Conclusions

As it was proven in this section, a gain scheduling LQI controller guarantees a good level of performances both in label tracking and in states training. Nonetheless, the computational cost of this method is high, and it still returns an approximated result because of the interpolation.

The computational cost of this method is high since, at each sampled value of d the following operations must be done:

- Change the position of the identically zero pole
- Solve an optimization problem

The controllers presented in this thesis were synthesized by discretizing the range of d in 1000 points. Then, after this n iterations are finished an interpolation problem must be solved. To reduce the computational cost it will now be used robust control, which allows to synthesize a controller for an uncertain plant while solving a single optimization problem.

3.7 Robust Control

In order to synthesize a $\mathcal{H}_\infty$ based controller it is necessary to create appropriate performance signals and weights to guarantee suitable performances by the controlled system.

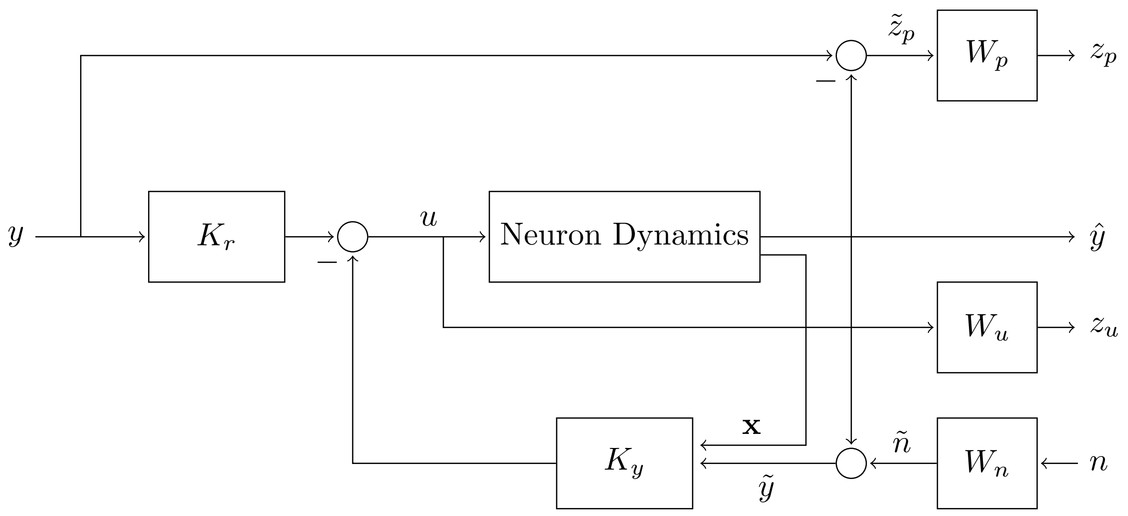

Figure 3.17: Structure used for the $\mathcal{H}_\infty$ controller synthesis.

The weights used for the $\mathcal{H}_\infty$ synthesis are W_p, W_u and W_n. In the robust control framework, weights are used to augment the system and set specifications for the synthesis.

The most complex weight is W_p.

$$W_p(s) = A\frac{s + z_p}{s + p_p} \tag{3.65}$$

Where A is the magnitude, z_p is the zero and p_p is the pole of the transfer function. Its goal is to shape the difference between the label and the estimated label. Its behaviour in the frequency domain can be seen in the following plot.

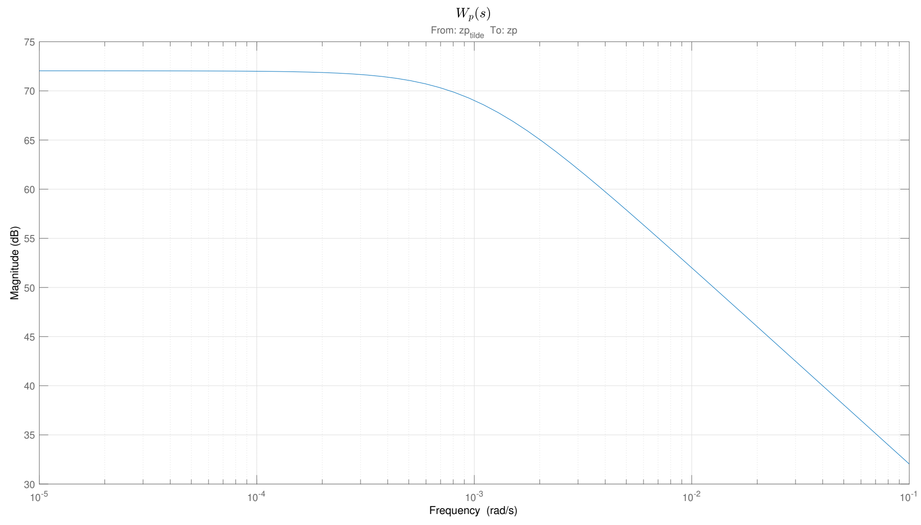

Figure 3.18: Bode magnitude plot of the weight $W_p(s)$.

By choosing the weight in this way, the error is going to decrease as the frequency increases.
The two other weights, i.e. W_u and W_n, are more simple. W_u is a static gain defined in such a way to act as an upper bound on the control input. On the other hand, W_n is used to simulate a noise on the estimated label. Although this weight is not strictly necessary, it can add some generality to the synthesis problem as numerical issues might happen. This can be represented by a static gain having magnitude 1% of y but it will be later shown how the systems can sustain higher levels of noise.
A helpful property of the system can now be exploited. It can be assumed that the controller has access to both the state vector and to the disturbances. Since the disturbances, i.e.y, n are both caused by the label, this assumption can be made. In this way, it is possible to use the MATLAB® built-in function *hinffi.m* which takes advantages of this assumption to synthesize a $\mathcal{H}_\infty$ based feedback controller. The control input u is then computed in the following way:

$$u = K \begin{bmatrix} \mathbf{x} \\ \mathbf{w} \end{bmatrix} \tag{3.66}$$

Where $\mathbf{w}$ is the disturbances vector, i.e. $\mathbf{w} = [r \quad n]$. It must be pointed out that $\mathbf{x}$ is now the state vector of the augmented plant shown in Fig. 3.17. Its dimensions are now $[3 \times 1]$. The controller for the plant shown in Fig.3.17 is:

$$K = 1\text{e}04 * \begin{bmatrix} -2.4276 & -4.8551 & 0.6043 & 0 & 0 \end{bmatrix} \tag{3.67}$$

Now, some closed loop simulations can be run. The loop is closed by performing a lower LFT between the augmented plant and the controller. The simulations are run for a sequence of different labels, the noise on the feedback branch is set to the 10% of the largest label injected in the system and by the use of *randn.m* is then made random while keeping a normal distribution. This increase is made to make the oscillations in the system clearer.

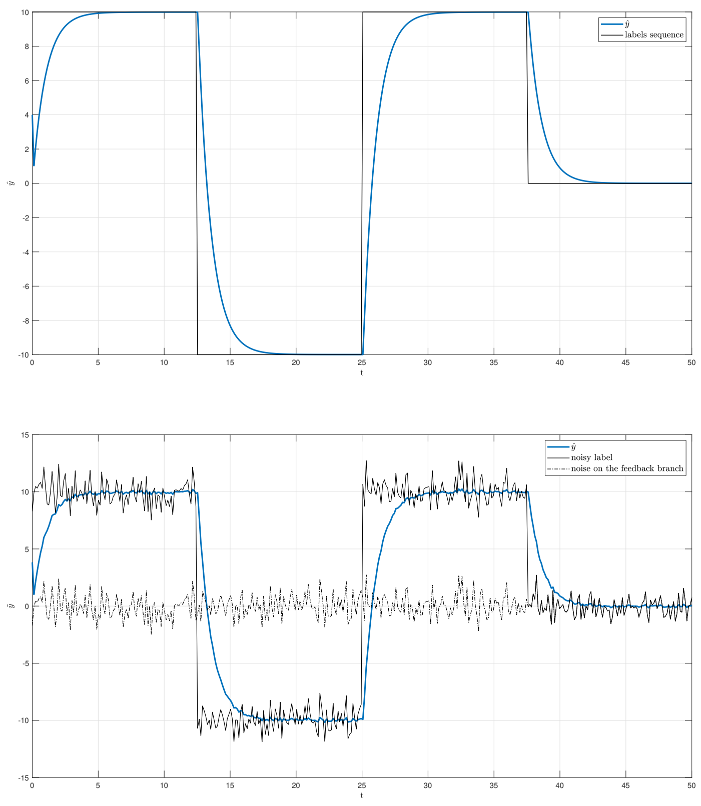

Figure 3.19: Closed loop simulation for a sequence of labels with a $\mathcal{H}_\infty$ based controller.

As shown in the previous plots, the estimated labels performance level, both the regular and the noisy one, is much higher than what was achieved with a gain scheduling optimal controller. Moreover to synthesize this robust controller, no assumptions or sampling on the range of d were made. This means that the controller presented in Eq.(3.67) holds for the entire range of d.
Although the convergence of the label is guaranteed, with the plant presented in Fig. 3.17 it is impossible to know the value of the trained hyperparameters. This issue can be easily overcome by augmenting the output equation of the neuron training dynamics in the following way:

$$C_{new} = \begin{bmatrix} C \\ I_2 \end{bmatrix} \tag{3.68}$$

In this way, the states current value is also part of the system output vector. The augmented plant will remain the same, with the only difference that C_{new}, and a new D matrix in order to be coherent w.r.t. systems' dimensions, is now used. The synthesis gives as result the following controller:

$$K_{aug} = 1e04 * \begin{bmatrix} -2.4367 & -4.8734 & 0.6066 \end{bmatrix} \tag{3.69}$$

As it may had been expected since the weights are kept the same, K_{aug} and K are very similar. A simulation similar to the one previously run showing the hyperparameters' training is now presented:

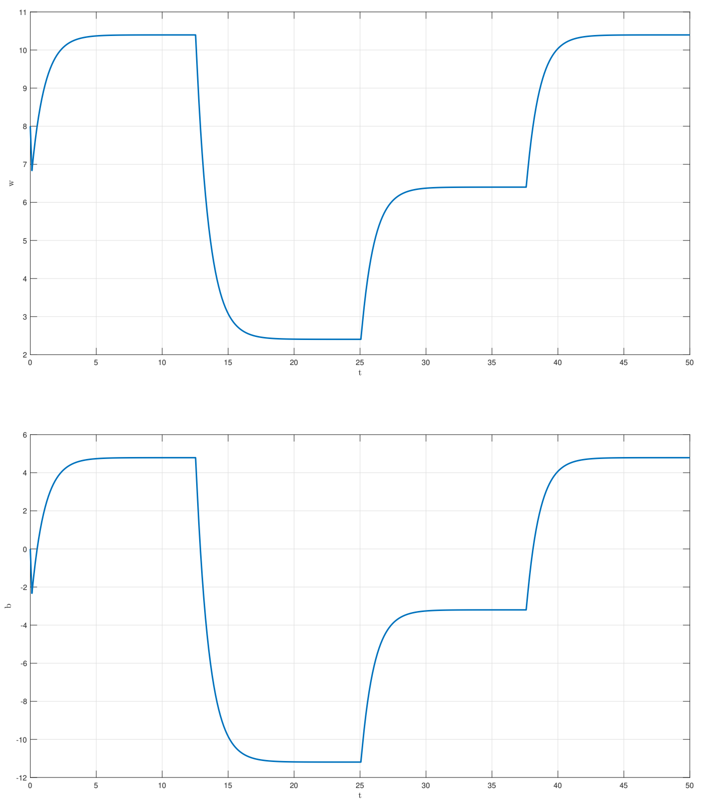

Figure 3.20: Training behaviour of the hyperparameters while controlled by a $\mathcal{H}_\infty$ based controller.

If compared to the gain scheduling LQI controller it can be proven that the hyperparameters converge to the same values. However, the convergence speed is much faster. The robust controller allows a convergence in approximately 5 time iterations while the system controlled by the optimal controller takes approximately 20 time iterations. The performances of the controller can be further improved. This can be done by tuning the bandwidth of $W_p(s)$, by properly tuning the weight, it will also be possible to achieve better γ values for the feedback system. For performances similar to the ones presented in Fig. 3.19 and Fig. 3.20, the γ value reached is about 3e04. Nonetheless, better values may be reached by both choosing different weights or by better tuning the current controller.

4
Conclusions

Finally, the aim of this thesis was to show of the extensive implementation of methods related to control theory to ANNs can result in a much faster convergence. What was proven is that it is possible to model the hyperparameters training dynamics and as shown in Chapter 3, the most effective way is to synthesize a robust controller. This could allow to enhance the stability margins of neural networks. Moreover, the application of robust control to networks training seems promising.
This work was limited to a single neuron training dynamics, but we believe that it can be easily extended to more complex networks. Furthermore, as shown in Chapter 3, each neuron can be easily modeled as an uncertain system with the help of the SMAC Toolbox [4].
In order to make this approach fully general two issues must be overcome. The first one regards the modeling of a full network. This issue is strictly related to MATLAB® as it is not possible to automatize the creation of function handles, which was a key part in obtaining the LFR of the network training dynamics. A way to overcome this problem could be use a different programming language to create the function handle and then import it inside the MATLAB® environment. Finally, as suggested in Section 3.5, the networks modeling should be done layer by layer, and then the LFR of each layer should be properly interconnected. The second one is more critical as it regards the inclusion of the activation function inside the training dynamics. As said in Chapter 2, a solution to this problem would be the use of Zames-Falb multipliers and then the use of integral quadratic constraints (IQC) to synthesize a controller.

Bibliography

[1] Lee, J. et al. (2019) Wide neural networks of any depth evolve as linear models under gradient descent." arXiv preprint arXiv:1902.06720.

[2] Jacot, A. et al. (2018) Neural Tangent Kernel: Convergence and Generalization in Neural Networks. *Advances in Neural Information Processing Systems*. arXiv:1806.07572

[3] Syrén, A. and Andersson V. (2020) Reining in Neural Networks with Control Theory.

[4] Magni, J-F. (2006) Linear Fractional Representation toolbox for use with Matlab. Available with the SMAC Toolbox at http://w3.onera.fr/smac/lfrt.

[5] Kulcsár, B. (2020) Lecture notes in Robust Control. Unpublished.

[6] Löfberg, J. (2004) YALMIP: A Toolbox for Modeling and Optimization in MATLAB. In: *In Proceedings of the CACSD Conference*.

[7] Skogestad, S. and Postlethwaite, I. (2005) Multivariable Feedback Control: Analysis and Design. Hoboken, NJ, USA: John Wiley & Sons, Inc. ISBN: 0470011688.

[8] Tóth, R. (2010) Modeling and identification of linear parameter-varying systems. Lecture Notes in Control and Information Sciences, Vol. 403, Springer, Heidelberg.

[9] Zames, G. and Falb, P. L. (1968) Stability Conditions for Systems with Monotone and Slope-Restricted Nonlinearities. SIAM Journal on Control - 6(1):pp. 89-108.

[10] Megretski, A. and Rantzer, A. (1997) System Analysis via Integral Quadratic Constraints. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 42, NO. 6, JUNE 1997.