Data-driven MPC applied to non-linear systems for real-time applications

Daniel Martin Xavier, Ludovic Chamoin, Laurent Fribourg

Data-driven MPC applied to non-linear systems for real-time applications*

Daniel Martin Xavier[1], Ludovic Chamoin[1], and Laurent Fribourg[2]

[1] Université Paris-Saclay, CentraleSupélec, ENS Paris-Saclay, CNRS, LMPS - Laboratoire de Mécanique Paris-Saclay, 91190, Gif-sur-Yvette, France
daniel.martin_xavier@ens-paris-saclay.fr
ludovic.chamoin@ens-paris-saclay.fr

[2] Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France
laurent.fribourg@ens-paris-saclay.fr

Abstract

Model Predictive Control (MPC) is a traditional technique widely employed on the control of constrained non-linear systems. In light of the increasing popularity of neural networks, data-driven MPC has emerged as an alternative to alleviate the computation burden of traditional control strategies. This paper aims to replace the constrained optimization problem by training a feed-forward neural network with data collected from an offline MPC simulation. The network's performance is then compared to that of traditional MPC using the Van der Pol oscillator as toy example. Finally, the convergence of the network training error is analytically proven by extending the analysis of neural tangent kernels (NTK) to underparameterized networks.

Contents

1 Introduction **1**

2 Model Predictive Control **3**
2.1 Data-driven MPC 3
2.2 Toy Example - Van der Pol Oscillator 4

3 Convergence of underparameterized neural networks **7**
3.1 Notation 7
3.2 Preliminaries 7
3.3 Gradient Descent for Neural Networks 8
3.4 Numerical Results 11

4 Conclusion **12**

1 Introduction

The most prevalent type of control strategy in industrial applications is the feedback controller, which involves the comparison of a reference signal $\boldsymbol{r}$ with a measured variable $\boldsymbol{y}$ to determine an appropriate value for the manipulated variable $\boldsymbol{u}$ using the error $\boldsymbol{e} = \boldsymbol{r} - \boldsymbol{y}$. This broad class of control strategies can be divided into three categories: classical controllers such as

*This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No. 101002857.

Proportional-Derivative-Integral (PID), predictive controllers such as Model Predictive Control (MPC) and Linear Quadratic Regulator (LQR), and repetitive controllers [24].

Within the domain of classical control strategies, the PID controller stands out as the most renowned, being extensively employed in industrial applications. Over the years, several setup rules have been proposed to adjust the controller's parameters, however finding an optimal parameterization can often be a challenging task. This is especially true for nonlinear and time-variant systems, where even higher control methods, such as back-stepping controllers, struggle to yield a suitable performance.

Predictive controllers, on the other hand, use a model of the system (digital twin) to predict its future behavior, being able to anticipate deviations from the reference. Within this class of controllers, MPC assumes a prominent role, relying on a repeated real-time optimization of a mathematical system model. Using the predictions of future states, MPC is able to determine the optimal trajectory of the manipulated variable $\boldsymbol{u}$, which is then adjusted at each time step by subsequent optimizations.

Model Predictive Control offers several advantages over classical control techniques, as it can anticipate the system's behavior and naturally consider hard constraints on the optimization problem. Furthermore, the parameterization of the controller is often more intuitive compared to other methods: the performance is usually improved at the cost of a higher computational effort, which is defined by a weighting coefficient.

In certain real-world applications, however, the mathematical model of the system may not be available due to the complexity of modeling or the lack of *a priori* knowledge regarding the phenomenon of interest. In such cases, data-driven models have proven to be effective to perform predictive control, using data obtained from sensors to identify the underlying dynamics of the system through auto-regressive, machine learning or physics-guided models [14].

Nonetheless, there has been limited progress in mitigating the computational burden of the MPC controller on real-time applications. That's why the present paper investigates the use of data-driven MPC to achieve fast computation of the control law while providing an analytical guarantee of the neural networks (NN). The goal is to replace the traditional constrained optimization problem in MPC by a NN to control a nonlinear system.

In recent years, the interest in neural networks has increased due to its remarkable performance across a wide range of applications. However, the comprehension of why such networks converge when trained via gradient descent, and generalize well over unseen data remained an open question for a while. Some studies have attempted to characterize the geometric landscape of objective functions in order to explain training convergence [21, 11].

Another approach consisted in investigating the trajectory of optimization and demonstrating convergence toward a global minimum [16, 9]. In this context, Zhang et al. [27] were among the first to study the generalization of neural networks, showing that they can attain zero training error on sufficiently large nets. This class of NNs are called overparameterized neural networks, characterized by a number of parameters m much larger than the number of data n ($m \gg n$).

Recently, the notion of Neural Tangent Kernel (NTK) matrix was introduced, providing an elegant explanation of the linear convergence towards 0 of the training error in overparameterized NNs with smooth activation functions [12]. The explanation essentially lies in the fact that, during the Gradient Descent (GD) procedure, the $n \times n$ NTK matrix stays close to its infinite limit, with all eigenvalues remaining positive.

Thereafter, Du et al. [10, 8] expanded this analysis to prove convergence of neural networks with non-smooth and non-convex activation functions. Arora et al. [3] also employed this framework to analyze convergence rates when using random and true labels for gradient descent.

However, most of the focus of these papers lies in the analysis of overparameterized neural networks, which is not the case of a considerable number of real-world applications.

In this paper, we address the problem of underparameterized neural networks ($m \ll n$), where the NTK matrix is positive semi-definite. Thus, its nullspace $\mathcal{N}$ is no longer reduced to 0, but it is a space of dimension larger than 1. Moreover, the NTK matrix evolves over time and its coefficients do not remain close to those of initialization. Nevertheless, under certain assumptions, this paper demonstrates that the training error converges linearly to a constant which can be accurately evaluated at the beginning of the GD procedure.

The remainder of the paper proceeds follows this structure: Section 2 presents the traditional MPC strategy and data-driven approaches with preliminary results using the Van der Pol oscillator as a toy example. Section 3 outlines the theoretical guarantees of the network convergence and the results are illustrated using the same toy example. Finally, Section 4 contains the conclusions and prospects for this work.

2 Model Predictive Control

Model Predictive Control (MPC) is a receding horizon strategy that uses a model of the system being controlled to predict its future states and compute suitable commands through a constrained optimization problem [24]. This strategy was first applied in chemical plants and oil refineries in order to deal with constraints in nonlinear systems [23]. Following its successful industrial implementation in the 1980s, the research community introduced theoretical tools to mathematically assure the stability and optimality of MPC controllers [19].

The advantage of MPC over other predictive strategies lies in its ability to continuously recompute a new trajectory based on data measured at each time step. Consequently, the controller is robust by design, capable of accounting for unexpected disturbances by dynamically recalculating paths to steer the system back to the desired state.

2.1 Data-driven MPC

Current research focuses on replacing the mathematical model on traditional MPC by data-driven models, which do not require *a priori* information about the system dynamics. Draeger et al. [7], for instance, employed a MPC based on the predictions generated by a neural network to control the pH levels in a neutralization reactor. Furthermore, Kaiser et al. [15] proposed the Sparse Identification of Nonlinear Dynamics (SINDY) to determine the prediction model of the system through a collection of predefined functions.

Alternative approaches concentrate on the derivation of the feasibility set or the cost function by data-driven methods in order to improve the controller performance. In this context, Berberich et al. [4] proposed a methodology to compute the terminal cost and terminal set for data-driven MPC using measured input-output data. Furthermore, Collet et al. [6] presented a fatigue-oriented MPC approach that employs a data-driven cost function to optimize the fatigue trade-off on wind turbines.

This paper aims to replace the constrained optimization problem in traditional MPC by a neural-network to alleviate its computational burden. The proposed strategy involves trading optimality for a fast computation of the control law. The neural network is trained using data generated by running MPC simulations offline. As a result, the data-driven MPC controller proposed in this paper is more suitable for real-time applications.

The majority of works in this topic focus on using neural networks to approximate the performance of an Explicit MPC controller. Chen et al. [5], for instance, proposed a modified

reinforcement learning policy for controlling a linear system, ensuring feasibility by projecting the control inputs onto polytopes. Furthermore, Winqvist et al. [26] developed a 2-layer neural network with constrained ReLU based activation function, incorporating a projection layer to guarantee recursive feasibility and asymptotic stability.

An Explicit MPC computes the control law offline in the form of a piecewise affine function for each control region of the state space. These regions are convex polytopes for linear MPC, making this strategy unsuited for highly nonlinear systems as the number of regions grows exponential with the model complexity. This work stands out from the existing literature by proposing the use of neural networks to replace the optimization problem on the control of non-linear systems.

To validate this approach, the proposed strategy is tested on a toy example of the Van der Pol oscillator, which is presented in the following subsection. In the remainder of this paper, the MPC problem is implemented using the MPC Toolbox in Python [18], which solves the differential algebraic equations of the nonlinear model through the Sundials CVODE, and the optimization problem using the Interior-Point Method [20].

2.2 Toy Example - Van der Pol Oscillator

The Van der Pol oscillator is a classical proof-of-concept dynamical system for optimal control applications. It was originally introduced by Balthazar Van der Pol for modeling the triode oscillations in electrical circuits [2]. The system possesses 2 states $\boldsymbol{x} = [x_1, x_2]$, a control action u, and a damping coefficient $\mu = 1$. Its dynamics are described by the following equations:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1 + u \end{cases} \tag{1}$$

The goal is to use MPC to make the system converge to a desired position x_1^{ref}, and then get this data to train offline a feed-forward neural network. The MPC problem is formulated using a time step of $T_s = 0.5$s, an initial condition of $\boldsymbol{x_0} = [1, 0]$ and a prediction horizon of $N = 5$. The command applied to the oscillator is constrained to the interval $u \in [-1, 1]$ and the cost function associated to the optimization problem is defined as:

$$J(\boldsymbol{x}, u) = \sum_{i=0}^{N-1} \|x_1^{ref}(k+i) - x_1(k+i)\|_2 + \alpha\|\Delta u(k+i)\|_2 \tag{2}$$

$$\text{s. t.} \quad u_{lb} \leq u(k+i) \leq u_{ub}, \quad \forall i \in [0, ..., N-1] \tag{3}$$

where u_{lb} and u_{up} represent the lower and upper bounds for the command, respectively, $\Delta u(k) = u(k) - u(k-1)$ the command variation, and $\alpha = 0.1$ a weighting coefficient that regulates the trade-off between reducing the error and increasing the command. Furthermore, the current time step is denoted by k, and the error is represented by $\epsilon(k) = x_1 - x_1^{ref}$.

The MPC simulations are conducted offline through the definition of different references that are randomly created between [-1,1], each reference being applied for 10s. The nonlinear model of the system is implemented using CasADi [1], being defined using the same time step as the MPC controller ($T_s = 0.5$s). The MPC simulation is 1800s long, resulting in a dataset of $S = 3600$ data points, which is divided into 3 sets: $S_{train} = 2160$ for training, $S_{val} = 720$ for validation, and $S_{test} = 720$ for testing.

Once the data is collected, the neural network is designed seeking a good balance between model complexity and performance. Its architecture is depicted in Figure 1, being a one-step-ahead predictive controller with one hidden layer. The network has 4 entries in the input layer, 10 neurons in the hidden layer, and 1 output in the last layer. It was implemented using PyTorch [22] with a ReLU activation function applied to the hidden layer, and a Hardtanh to the output layer to constrain the command within the interval $[-1, 1]$.

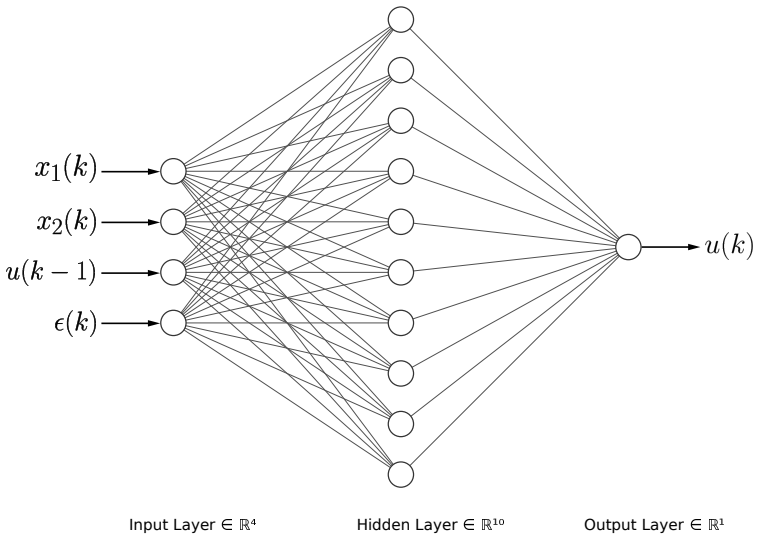

Figure 1: Architecture of the neural network used to mimic the behavior of an optimization problem.

The neural network was trained using the AdamW [17] optimization algorithm during 10^4 epochs with a learning rate of 10^{-3}, and a weight decay of 10^{-6}. The performance of the trained network is assessed using two metrics: the root mean squared error (RMSE) in (4), and the coefficient of determination (R2 score) in (5). In these equations, u_i^{MPC} represents the command computed by MPC, u_i^{NN} the command predicted by the neural network, and $\bar{u}^{\text{MPC}}$ an average of MPC commands.

$$\text{RMSE} = \sqrt{\frac{1}{S_{test}} \sum_{i=1}^{S_{test}} \left(u_i^{\text{MPC}} - u_i^{\text{NN}}\right)^2} \tag{4}$$

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{S_{test}} \left(u_i^{\text{MPC}} - u_i^{\text{NN}}\right)^2}{\sum_{i=1}^{S_{test}} \left(\bar{u}^{\text{MPC}} - u_i^{\text{NN}}\right)^2} \tag{5}$$

The trained network is capable of simulating the MPC behavior over the test dataset with an error of RMSE $= 0.0314$, indicating a satisfactory level of generalization over unseen data. Furthermore, the NN obtains a high coefficient of determination $\text{R}^2 = 0.9975 \sim 1$, showing its efficiency in the regression task. Figure 2 illustrates the command computed by MPC compared to the one predicted by the neural network using the test set, highlighting its capacity in learning the behavior of an optimization problem.

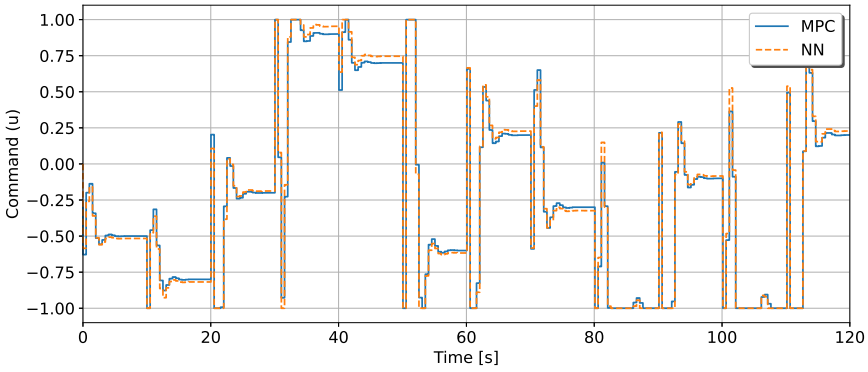

Figure 2: Comparison between the true command (u^{MPC}) and the command predicted by the neural network (u^{NN}) using the test dataset.

Finally, to evaluate the performance of the data-driven MPC, the NN is used to control the Van der Pol oscillator online. The simulation is initialized at $\boldsymbol{x_0} = [-0.3, 0]$, following the same reference trajectory as in the testing dataset. Figure 3 presents the simulation results, illustrating the capability of the neural network to effectively steer the system to the desired reference. In comparison to traditional MPC, the proposed strategy yields a trajectory with more oscillations, which is empirically verified by the RMSE as shown in Table 1.

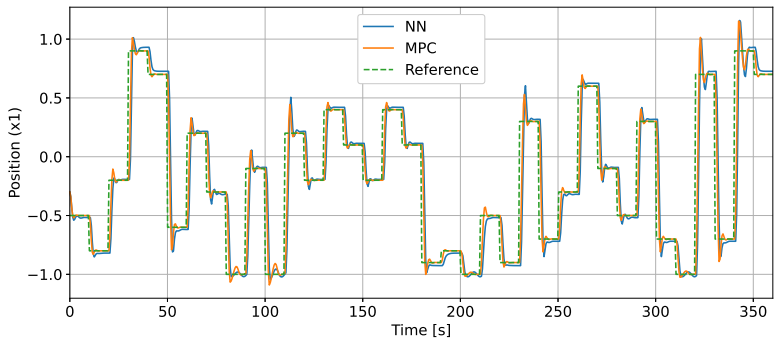

Figure 3: Comparison between traditional and data-driven MPC on following the reference x_1^{ref} on the testing dataset.

Remark 1. These closed-loop oscillations suggest that the traditional MPC was not finely tuned to achieve zero overshoot. As the NN is trained to reproduce the MPC performance, the network will learn to oscillate if this is the trajectory yield by MPC. In order to avoid such oscillations, the traditional MPC should be carefully tuned by adjusting the prediction horizon N and the weighting coefficient α. Another option is to enforce precision by penalizing the deviations from the reference in the function being minimized by the GD algorithm.

However, the primary goal of the proposed strategy is to prioritize fast computation of the

control law at the expense of optimality, which was satisfactorily achieved using neural networks. The average run-time of traditional MPC is 2.4201 ms, over 50 times slower than data-driven MPC, primarily due to the time-consuming nature of the constrained optimization problem. This difference would only become more pronounced with a larger prediction horizon $N > 5$, confirming the suitability of data-driven over traditional MPC for real-time applications.

Table 1: Metrics for the closed-loop simulation.

Strategy	**RMSE**	**Computation Time (ms)**
MPC	0.252	2.420
NN	0.312	0.045

3 Convergence of underparameterized neural networks

In recent years, the interest in neural networks has increased due to its outstanding performance across a wide range of applications. Nevertheless, they lacked a formal formulation of theoretical properties regarding their guarantees of convergence for years. Within the existing literature, Jacot et al. [12] proposed that the convergence of gradient descent could be characterized by a kernel in the case of overparameterized neural networks with smooth activation functions. Subsequently, Du et al. [10] expanded this analysis to prove the convergence of neural networks with non-smooth activation functions.

In this paper, the analysis of convergence using a kernel is further expanded to encompass underparameterized networks, in which the number of trainable parameters are smaller compared to available data ($m \ll n$). After some preliminaries (Subsection 3.2), the proof of the error convergence on underparameterized neural networks is presented (Subsection 3.3), followed by some numerical results on the Van der Pol oscillator (Subsection 3.4).

3.1 Notation

In this paper, we denote by $\mathbb{R}$ and $\mathbb{N}$ the set of real and natural numbers, respectively. These symbols are annotated with subscripts to restrict them in the usual way, e.g., $\mathbb{R}_{>0}$ denotes the positive real numbers. We also denote by $\mathbb{R}^n$ an n-dimensional Euclidean space, and by $\mathbb{R}^{n\times m}$ a space of real matrices with n rows and m columns.

We use bold letters for vectors and bold capital letters for matrices. Given a matrix $\boldsymbol{A}$, let $\boldsymbol{A}_{i,j}$ be its (i,j)-th entry, $\lambda_{\min}(\boldsymbol{A})$ its minimal eigenvalue, and $\boldsymbol{A}^\top$ its transpose. The Euclidean norm is denoted by $\|\cdot\|$, the Frobenius norm by $\|\cdot\|_F$, and the inner product by $\langle\cdot,\cdot\rangle$. Let $\boldsymbol{I}_n$ be the $n\times n$ identity matrix, $[n]$ the set $\{1,\dots,n\}$, and $\sigma(\cdot)$ the ReLU function $\sigma(z) = \max\{z, 0\}$.

We denote by $\mathbb{I}\{E\}$ the indicator function for an event E, by $B \uplus C$ the disjoint union of sets B and C, and by $\mathcal{V}_1 \oplus \mathcal{V}_2$ the direct sum of vector spaces $\mathcal{V}_1$ and $\mathcal{V}_2$. We also use the abbreviation i.i.d. to indicate that a collection of random variables is independent and identically distributed. Finally, the time-discrete version of a time-continuous object ξ is denoted as $\tilde{\xi}$.

3.2 Preliminaries

We recall from Du et al. [10] some definitions regarding the application of the gradient descent algorithm to neural networks. We consider a NN of the form:

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(\boldsymbol{w}_r^\top \boldsymbol{x}) \tag{6}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ represents the input, $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_m] \in \mathbb{R}^{d \times m}$ the weight matrix of the first layer composed by vectors $\boldsymbol{w}_r \in \mathbb{R}^d$ for $r \in [m]$, and $a_r \in \mathbb{R}$ the output weight. Additionally, $\boldsymbol{\theta} = [\boldsymbol{w}_1{}^\top, \dots, \boldsymbol{w}_m{}^\top, a_1{}^\top, \dots, a_m{}^\top]$ denotes a vector containing the parameters from both layers.

This work focuses on the empirical risk minimization problem with the quadratic loss defined in (7). Indeed, given a training data set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, the objective of the optimization algorithm is to minimize the following loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} (f(\boldsymbol{\theta}, \boldsymbol{x}_i) - y_i)^2 \tag{7}$$

In order to do so, Du et al. [10] fixed the parameters of the second layer of the network and applied the gradient descent algorithm to the weights of the first layer. In this paper, however, both layers are optimized simultaneously during training through the following update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - h \frac{\partial \mathcal{L}(\boldsymbol{\theta}_k)}{\partial \boldsymbol{\theta}_k} \tag{8}$$

The gradient formula for each layer is, then, given by:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{w}_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^{n} (f(\boldsymbol{\theta}, \boldsymbol{x}_i) - y_i) a_r \boldsymbol{x}_i \mathbb{I}\{\boldsymbol{w}_r^\top \boldsymbol{x}_i \geq 0\} \tag{9}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial a_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^{n} (f(\boldsymbol{\theta}, \boldsymbol{x}_i) - y_i) \sigma(\boldsymbol{w}_r^\top \boldsymbol{x}_i) \tag{10}$$

Given the discontinuous nature of the ReLU activation function, one may consider $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ as a convenient notation for the right hand side of (9) and (10). Lastly, the discrete equation (8) corresponds to the Euler discretization, with a step size h, of the set of ordinary differential equations defined by:

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\frac{\partial \mathcal{L}(\boldsymbol{\theta}(t))}{\partial \boldsymbol{\theta}(t)}. \tag{11}$$

3.3 Gradient Descent for Neural Networks

When training a neural network, the convergence of the GD algorithm to a globally optimal solution comes down to showing the convergence of the error (i.e. difference between the prediction of the NN and the ground truth) to zero [10]. For this reason, the rest of this paper will focus on the analysis of the error dynamics $\boldsymbol{v} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, defined by:

$$\boldsymbol{v} = \boldsymbol{p} - \boldsymbol{y}. \tag{12}$$

where $\boldsymbol{p} = (p_1, \dots, p_n)^\top \in \mathbb{R}^n$ is a vector with all n predictions $p_i = f(\boldsymbol{\theta}, \boldsymbol{x}_i)$ and $\boldsymbol{y} = (y_1, \dots, y_n)^\top \in \mathbb{R}^n$. As demonstrated by Du et al. [10], the continuous dynamics of the error $\boldsymbol{v}$ can be written in a compact way:

$$\frac{d}{dt}\boldsymbol{v}(t) = -\boldsymbol{H}[\boldsymbol{\theta}(t)]\boldsymbol{v}(t), \quad \boldsymbol{v}(0) = \boldsymbol{v}_0 \tag{13}$$

where $\boldsymbol{H}[\boldsymbol{\theta}] : \mathbb{R}_{\geq 0} \to \mathbb{R}^{n\times n}$ is the NTK matrix from a kernel associated with the ReLU function, being symmetric positive semi-definite as follows:

$$\boldsymbol{H}_{ij}[\boldsymbol{\theta}] = \left\langle \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{x}_j)}{\partial \boldsymbol{\theta}} \right\rangle$$

The discrete version resulting from the Euler discretization of (13), corresponding to the GD algorithm of $\tilde{\boldsymbol{v}}$, reads:

$$\tilde{\boldsymbol{v}}_{k+1} - \tilde{\boldsymbol{v}}_k = -h\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]\tilde{\boldsymbol{v}}_k \tag{14}$$

where h is the step size, and $\tilde{\boldsymbol{v}}(0) = \tilde{\boldsymbol{v}}_0$. We are now ready to formalize the problem:

Problem 1. *Given the discrete time system in (14), provide conditions over the matrix $\boldsymbol{H}[\boldsymbol{\theta}] : \mathbb{R}_{\geq 0} \to \mathbb{R}^{n\times n}$, the loss function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ and the step size h to ensure the convergence of $\tilde{\boldsymbol{v}}_k$ to zero, together with an explicit bound on its convergence rate.*

In the literature, Jerray et al. [13] provided a solution to Problem 1 under the following assumptions:

Assumption 1. *The gradient descent algorithm used for updating the weights of the neural network in (8) converges to a local minima $\boldsymbol{\theta}^*$, i.e. $\frac{\partial \mathcal{L}(\tilde{\boldsymbol{\theta}}_k)}{\partial \boldsymbol{\theta}}$ converges to 0 as k goes to the infinity.*

Assumption 2. *There exist $\lambda^* > 0$ and $t_0 \geq 0$ such that, for all $t \geq t_0$: $\lambda_{\min}(\boldsymbol{H}[\boldsymbol{\theta}(t)]) \geq \lambda^*$, where the time-varying matrix $\boldsymbol{H}[\boldsymbol{\theta}](t)$ is given in (13).*

Assumption 3. *$\mathcal{L}$ is locally strongly convex around every local minimizer $\boldsymbol{\theta}^*$ of $\mathcal{L}$, that is: for every local minimizer $\boldsymbol{\theta}^*$, there is a neighborhood around $\boldsymbol{\theta}^*$ on which $\mathcal{L}$ is strongly convex.*[1]

These assumptions are natural in the context of overparameterized NNs, as discussed in [13], leading to:

Theorem 1. *Under Assumptions 1 and 3, if the step size h satisfies $h < \frac{2}{L}$, where L is the Lipschitz constant of the loss function $\mathcal{L}$, then the sequence $\|\tilde{\boldsymbol{\theta}}_k - \boldsymbol{\theta}_k\|$, $k \in \mathbb{N}$ converges to 0, where $\tilde{\boldsymbol{\theta}}_k$ is defined by (8) and $\boldsymbol{\theta}_k = [\boldsymbol{w}_1^\top(kh), \ldots, \boldsymbol{w}_m^\top(kh), a_1(kh), \ldots, a_m(kh)]$ with $\boldsymbol{\theta}(t)$ defined by (11).*

Besides the above theorem, Jerray et al. [13] used Assumption 2 to derive the following theorem:

[1] Remark: The local strong convexity property is equivalent to the fact that $-\frac{\partial \mathcal{L}(\tilde{\boldsymbol{\theta}}_k)}{\partial \boldsymbol{\theta}}$ belongs to a contractive region, i.e, there exists $k_0 \in \mathbb{N}$, a convex region D and a positive real $\gamma > 0$ such that $\tilde{\boldsymbol{\theta}}_k \in D$ for all $k \geq k_0$, $\boldsymbol{\theta} \in D$ and $\boldsymbol{\theta}' \in D$ the following relation is true:

$$\left\langle \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \frac{\partial \mathcal{L}(\boldsymbol{\theta}')}{\partial \boldsymbol{\theta}}, \boldsymbol{\theta} - \boldsymbol{\theta}' \right\rangle \geq \gamma \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|^2$$

This is also equivalent to the positive definiteness of the Hessian of $\mathcal{L}$ in the neighborhood of each local minimizer $\boldsymbol{\theta}^*$. Moreover, in view of [25], the strong convexity corresponds to the special case of contracting gradient descent in the identity metric.

Theorem 2. *Under Assumptions 1, 2 and 3, if the step size h satisfies $h < \frac{2}{L}$, where L is the Lipschitz constant of the loss function $\mathcal{L}$, then there exist $\mu \in (0,1)$ and $k_0 \in \mathbb{N}$ such that, for all $k \geq k_0$:*

$$\|\tilde{\boldsymbol{v}}_k\| \leq \|\tilde{\boldsymbol{v}}_{k_0}\| \mu^{k-k_0}.$$

In this paper, the Assumption 2 does not hold, and without loss of generality, we suppose that the space $\mathcal{V}(t)$ of eigenvectors of $\boldsymbol{H}[\boldsymbol{\theta}(t)]$ decomposes as $\mathcal{V}_1(t) \oplus \mathcal{V}_2(t)$, where $\mathcal{V}_1$ (resp. $\mathcal{V}_2$) is the cluster of eigenvectors of eigenvalues $\lambda_i(t)$ greater than (resp. less than or equal to) some $\alpha > 0$. Formally Assumption 2 is replaced by:

Assumption 4. *There exist $\lambda^* \in \mathbb{R}_{>0}$, $t_0 \geq 0$, $I_1 = \{0, \dots, n_1 - 1\}$ and $I_2 = \{n_1, \dots, n_1 + n_2 - 1\}$ with $n_1 + n_2 = n$ such that the eigenvalues can be divided into:*

- *$\lambda_i(t) \geq \lambda^*$ for all $i \in I_1, t \geq t_0$,*
- *$\lambda_i(t) = 0$ for all $i \in I_2, t \geq t_0$.*

where $\{\lambda_i(t)\}_{i \in I_1 \uplus I_2}$ is the eigenvalues set of the NTK matrix $\boldsymbol{H}[\boldsymbol{\theta}(t)]$, and I_1 (resp. I_2) the index set of eigenvectors spanning $\mathcal{V}_1(t)$ (resp. $\mathcal{V}_2(t)$).

Under Assumption 4 instead of Assumption 2, the result of Theorem 2 becomes:

Theorem 3. *Under Assumptions 1, 3 and 4, if the step size h satisfies $h < \frac{2}{L}$ and $h < \frac{2}{\lambda^*}$, where L is the Lipschitz constant of the loss function $\mathcal{L}$, then there exists k_0 such that for all $k \geq k_0$:*

$$\|\tilde{\boldsymbol{v}}_k\| \leq \sqrt{(1 - \frac{1}{2}\lambda^* h)^{2(k-k_0)} \|\tilde{\boldsymbol{v}}_{k_0}^1\|^2 + \|\tilde{\boldsymbol{v}}_{k_0}^2\|^2} \tag{15}$$

where $\tilde{\boldsymbol{v}}_k^i$ $(i = 1, 2)$ is the projection of the vector $\tilde{\boldsymbol{v}}_k$ on the span of the n_i eigenvectors of $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}(k)]$. The right-hand side of (15) thus converges to the constant $b = \|\tilde{\boldsymbol{v}}_{k_0}^2\|$.

Proof. By Theorem 1, which relies on Assumptions 1 and 3, but not Assumption 2, we know that for $h < \frac{2}{L}$, $\|\tilde{\boldsymbol{\theta}}_k - \boldsymbol{\theta}_k\|$ converges to 0 as $k \to \infty$. Let $\tilde{\lambda}_i(kh)$ $(i \in [n])$ denote the eigenvalues of the matrix $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$ obtained from GD, which is discrete in time. Recall that Assumption 4 specifies that the eigenvalues $\lambda_i(t)$ of $\boldsymbol{H}[\boldsymbol{\theta}(t)]$ with $i \in [n_1]$ are in $[\lambda^*, \infty)$, and those with $i \in [n_2]$ are null. It follows by continuity, using the fact that $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$ is positive semi-definite, that there exists k_0 such that for all $k \geq k_0$:

$$\tilde{\lambda}_i(kh) \in [\frac{\lambda^*}{2}, \infty) \text{ for all } i \in I_1, \tag{16}$$

$$\tilde{\lambda}_i(kh) \in [0, \frac{\lambda^*}{2}) \text{ for all } i \in I_2. \tag{17}$$

We know that for $k \geq k_0$, $\mathcal{V}_k^1$ and $\mathcal{V}_k^2$ are orthogonal (since all the eigenvalues $\tilde{\lambda}_i(kh)$ with $i \in I_1$ are different from the eigenvalues $\tilde{\lambda}_j(kh)$ with $j \in I_2$). The discretized dynamics of the error in (14) then writes:

$$\tilde{\boldsymbol{v}}_{k+1} = (\boldsymbol{I}_n - h\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k])\tilde{\boldsymbol{v}}_k. \tag{18}$$

We also know that the NTK matrix $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$ can be decomposed as:

$$\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k] = \boldsymbol{P}_k \boldsymbol{D}_k \boldsymbol{P}_k^\top \tag{19}$$

where $\boldsymbol{P}_k$ is the transition matrix whose columns are the eigenvectors of $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$, and $\boldsymbol{D}_k$ is the diagonal eigenvalue matrix with first n_1 elements in $[\frac{\lambda^*}{2}, \infty)$, and last n_2 elements in $[0, \frac{\lambda^*}{2})$. From (18), it follows:

$$\boldsymbol{P}_k^\top \tilde{\boldsymbol{v}}_{k+1} = (\boldsymbol{I}_n - h\boldsymbol{D}_k)\boldsymbol{P}_k^\top \tilde{\boldsymbol{v}}_k \tag{20}$$

The $n \times n$ matrix $\boldsymbol{P}_k^\top$ is made of an upper $n_1 \times n$ matrix $\boldsymbol{P}_{k,1}^\top$ and a lower $n_2 \times n$ matrix $\boldsymbol{P}_{k,2}^\top$. The rows of the matrix $\boldsymbol{P}_{k,1}^\top$ are the n_1 eigenvectors of $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$ associated with eigenvalues in $[\frac{\lambda^*}{2}, \infty]$, and those of $\boldsymbol{P}_{k,2}^\top$ are the n_2 other eigenvectors. Hence, for a vector $\boldsymbol{v}$, $\boldsymbol{P}_{k,i}^\top \boldsymbol{v}$ $(i = 1, 2)$ corresponds to the projection $\tilde{\boldsymbol{v}}_k^i$ of $\boldsymbol{v}$ on the span $\mathcal{V}_k^i$ of the n_i eigenvectors of $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$. Equation (20) thus decomposes as:

$$\tilde{\boldsymbol{v}}_{k+1}^1 = (\boldsymbol{I}_{n_1} - h\boldsymbol{D}_k^1)\tilde{\boldsymbol{v}}_k^1 \tag{21}$$

$$\tilde{\boldsymbol{v}}_{k+1}^2 = (\boldsymbol{I}_{n_2} - h\boldsymbol{D}_k^2)\tilde{\boldsymbol{v}}_k^2 \tag{22}$$

where $\boldsymbol{D}_k^1$ is the $n_1 \times n_1$ top left submatrix of $\boldsymbol{D}_k$, and $\boldsymbol{D}_k^2$ the $n_2 \times n_2$ bottom right submatrix. For $k \geq k_0$ and $h < \frac{2}{\lambda^*}$, equations (21) and (22) lead to:

$$\|\tilde{\boldsymbol{v}}_{k+1}^1\| \leq \|\boldsymbol{I}_{n_1} - h\boldsymbol{D}_k^1\| \|\tilde{\boldsymbol{v}}_k^1\| \leq (1 - \frac{1}{2}\lambda^* h)\|\tilde{\boldsymbol{v}}_k^1\| \tag{23}$$

$$\|\tilde{\boldsymbol{v}}_{k+1}^2\| \leq \|\boldsymbol{I}_{n_2} - h\boldsymbol{D}_k^2\| \|\tilde{\boldsymbol{v}}_k^2\| \leq \|\tilde{\boldsymbol{v}}_k^2\| \tag{24}$$

It follows from (23) and (24) that there exists k_0 such that for all $k \geq k_0$ and $h < \frac{2}{\lambda^*}$:

$$\begin{aligned} \|\tilde{\boldsymbol{v}}_k^1\| &\leq (1 - \frac{1}{2}\lambda^* h)^{k-k_0} \|\tilde{\boldsymbol{v}}_{k_0}^1\| \\ \|\tilde{\boldsymbol{v}}_k^2\| &\leq \|\tilde{\boldsymbol{v}}_{k_0}^2\| \end{aligned}$$

Finally, using the fact that $\mathcal{V}_k^1$ and $\mathcal{V}_k^2$ are orthogonal:

$$\begin{aligned} \|\tilde{\boldsymbol{v}}_k\|^2 &= \|\tilde{\boldsymbol{v}}_k^1\|^2 + \|\tilde{\boldsymbol{v}}_k^2\|^2 \\ &\leq (1 - \frac{1}{2}\lambda^* h)^{2(k-k_0)} \|\tilde{\boldsymbol{v}}_{k_0}^1\|^2 + \|\tilde{\boldsymbol{v}}_{k_0}^2\|^2 \end{aligned} \tag{25}$$

□

Remark 2. In practice k_0 is small, which means that $\|\tilde{\boldsymbol{v}}_k^2\|$ is rapidly constant from $t_0 = k_0 h$. The training error $\|\tilde{\boldsymbol{v}}_k\|$ is thus asymptotically equal to $b = \|\tilde{\boldsymbol{v}}_{k_0}^2\|$. The constant b depends on the initial value $\tilde{\boldsymbol{\theta}}(0)$ of the NN parameters. Note that, apart from the choice of initial weight $\tilde{\boldsymbol{\theta}}(0)$, the GD procedure considered here is deterministic.

3.4 Numerical Results

In order to validate the above analysis, a toy example using the Van der Pol oscillator is used. The same NN architecture presented in Figure 1 is used, but this time only 1 batch of 20 data points are employed for training using the SGD (Stochastic Gradient Descent) optimization algorithm with a learning rate of 10^{-2}. The goal is to demonstrate that the error can be decomposed into two subsets ($\mathcal{V}_1$ and $\mathcal{V}_2$), the projection of the error onto $\mathcal{V}_1$ converges to 0 as $t \to \infty$, and the projection of the error onto $\mathcal{V}_2$ remains nearly constant during training.

Figure 4 illustrates the evolution of the eigenvalues during gradient descent, in which $\lambda^* = 10^{-3}$ is the threshold separating both space of eigenvalues: $\lambda_i > \lambda^*$. Furthermore, it can be

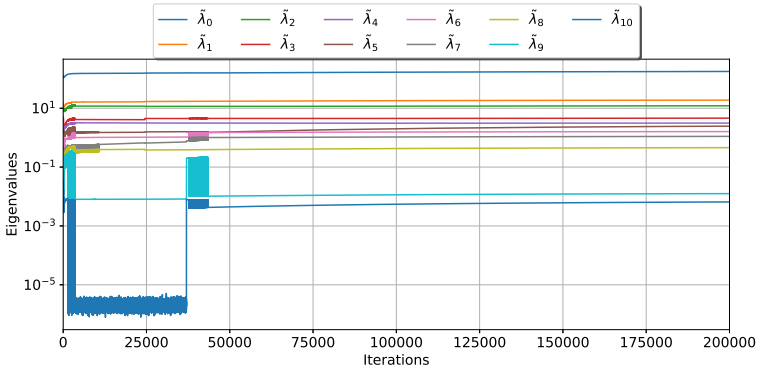

Figure 4: Evolution of the 11 positive eigenvalues $(\tilde{\lambda}_0, \dots, \tilde{\lambda}_{10})$ of the NTK matrix $\boldsymbol{H}[\tilde{\boldsymbol{\theta}}_k]$(the 9 other eigenvalues being approximately 0).

noticed that the eigenvalues evolve in time as the network is underparameterized, which makes the derivation of convergence bounds a difficult task. However, from the projection of the error on $\mathcal{V}_2$, a lower bound of convergence can be derived as depicted in Figure 5.

From the evolution of $||\boldsymbol{v}_2||$, it is possible to verify that the lower bound is not zero, remaining nearly constant ($b = 0.04$) during training. We can also verify some small oscillations of $||\boldsymbol{v}_2||$ due to the evolution of eigenvectors $\mathcal{V}_2$ during training as their associated eigenvalues. Despite the fact that the network error $\boldsymbol{v}$ is unable to converge to zero, the lower bound b is small, leading to good performances.

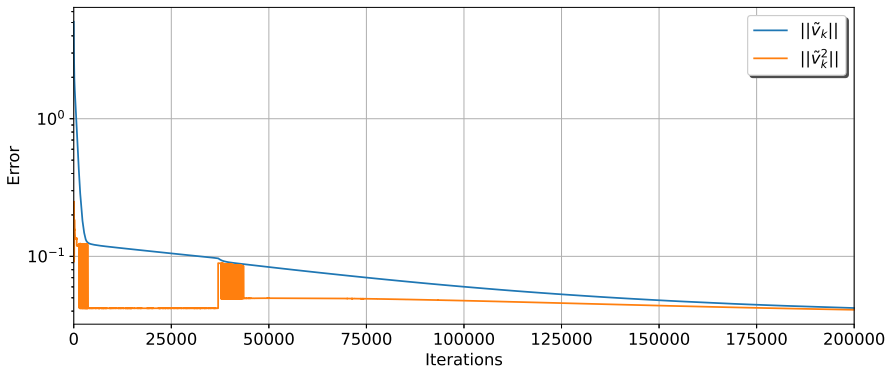

Figure 5: Log-scale evolution of the training error $\|\tilde{\boldsymbol{v}}_k\|$ (blue) and it's projection over $\mathcal{V}_2$ resulting in $\|\tilde{\boldsymbol{v}}_k^2\|$ (orange), which is a lower bound for the error.

4 Conclusion

In this paper we were interested in reducing the computational burden of MPC for the real-time synthesis of a control law. The method consists in training a neural network on the data

generated by a MPC controller, then using this neural network to control the system in real time. The theoretical guarantees of convergence are derived by expanding the analysis of neural tangent kernels to underparameterized networks.

From the results in Section 2, we conclude that data-driven MPC is a viable solution for simulating the behavior of a traditional MPc controller. It shifts the computational burden of MPC to offline training, speeding up the computation of a control law. Regarding performance, the traditional MPC is capable of following the reference more accurately than the data-driven approach, however this gap can be reduced by increasing the neural network complexity.

One of the main focuses of future research is the integration of hard constraints to the neural network, which is not evident as just adding a penalization factor to the cost function as it is done for soft constraints. Furthermore, other applications will be studied as the forging process, which is highly non-linear, and, finally, a proof-of-concept will be carried out in the context of Structural Health Monitoring (SHM) applications.

Regarding the theoretical aspects of network convergence, future work consists in deriving an upper bound of convergence. Moreover, this analysis can be extended to derive guarantees of convergence using Federative Learning for underparameterized neural networks. The interest in this particular type of learning is that training can be performed separately over different datasets, opening the opportunity for parallelization to improve computational time.

Acknowledgment

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No. 101002857.

References

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[2] Eric Aislan Antonelo, Eduardo Camponogara, Laio Oriel Seman, Eduardo Rehbein de Souza, Jean P. Jordanou, and Jomi F. Hubner. Physics-informed neural nets for control of dynamical systems, 2022.

[3] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks, 2019.

[4] Julian Berberich, Johannes Köhler, Matthias A. Müller, and Frank Allgöwer. On the design of terminal ingredients for data-driven mpc. *IFAC-PapersOnLine*, 54(6):257–263, 2021. 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.

[5] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D. Lee, Vijay Kumar, George J. Pappas, and Manfred Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, pages 1520–1527, 2018.

[6] David Collet, Mazen Alamir, Domenico Di Domenico, and Guillaume Sabiron. Data-driven fatigue-oriented mpc applied to wind turbines individual pitch control. *Renewable Energy*, 170:1008–1019, 2021.

[7] A. Draeger, S. Engell, and H. Ranke. Model predictive control using neural networks. *IEEE Control Systems Magazine*, 15(5):61–66, 1995.

[8] Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks, 2019.

[9] Simon S. Du, Jason D. Lee, Yuandong Tian, Barnabas Poczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima, 2018.

[10] Simon S. Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, abs/1810.02054, 2018.

[11] C. Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization, 2017.

[12] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

[13] Jawher Jerray, Adnane Saoud, and Laurent Fribourg. Using euler's method to prove the convergence of neural networks. *IEEE Control Systems Letters*, 6:3224–3228, 2022.

[14] Kadierdan Kaheman, Eurika Kaiser, Benjamin Strom, J. Nathan Kutz, and Steven L. Brunton. Learning discrepancy models from experimental data, 2019.

[15] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2219):20180335, 2018.

[16] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation, 2017.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[18] S. Lucia, A. Tatulea-Codrean, C. Schoppmeyer, and S. Engell. Rapid development of modular and sustainable nonlinear model predictive control solutions, 2017.

[19] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[20] Yu. Nesterov. Interior-point methods: An old and new approach to nonlinear programming. *Math. Program.*, 79(1–3):285–297, oct 1997.

[21] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks, 2017.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] S.Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

[24] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117:1–23, 11 2021.

[25] Patrick M. Wensing and Jean-Jacques Slotine. Beyond convexity—contraction and global convergence of gradient descent. *PLOS ONE*, 15(8):e0236661, aug 2020.

[26] Rebecka Winqvist, Arun Venkitaraman, and Bo Wahlberg. On training and evaluation of neural network approaches for model predictive control, 2020.

[27] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2017.