Tractable Data-Driven Model Predictive Control Using One-Step Neural Networks Predictors

Danilo Menegatti, *Student Member, IEEE*, Alessandro Giuseppi, *Senior Member, IEEE*, and Antonio Pietrabissa, *Senior Member, IEEE*

***Abstract*— Model Predictive Control (MPC) is a popular control strategy that relies on the availability of a prediction model to estimate future system trajectories over a finite time horizon. Recently, researchers have introduced Neural Networks (NNs) into the MPC framework for the development of data-driven prediction models. In MPC, the control actions are computed by solving iteratively, at each time-step, an optimization problem subject to state and input constraints. Finding the optimal solution to such a problem is a crucial challenge in the data-driven setting, due to the complexity and black-box nature of data-driven models such as NNs. This paper addresses this challenge by proposing a hierarchical deep NN formed by a set of cascading one-step NN predictors whose combination constitutes an interpretable prediction model over the entire prediction horizon. Thanks to the proposed NN architecture, it is shown that the resulting optimal control problem is tractable, as it can be solved by employing efficient iterative algorithms, and interpretable, so that input and state constraints can be enforced seamlessly. The effectiveness of the proposed method is validated through numerical simulations.**

***Note to Practitioners*—Model Predictive Control (MPC) is a widely used methodology in the industry which typically relies on the availability of a model in the form of step response, transfer function or state-space models. In some cases, the explicit model might not be available or its accuracy may be not sufficient for the required closed-loop performance. This paper aims to develop a simple and practical framework for deploying a model-free data-driven MPC solution based on deep learning. This objective is pursued by suggesting a novel approach using simple neural networks in a cascading interpretable structure. Such networks are used to predict the one-step evolution of the system, and their cascade represents the MPC prediction model over an arbitrary long prediction horizon. We characterize such a neural model focusing on its interpretability and tractability, deriving the resulting optimal control problem to be solved in a receding horizon strategy. We then show that the MPC optimization can be solved efficiently using highly efficient iterative algorithms that can be implemented in practice. Numerical simulations involving the use of the Alternating Direction Method of Multipliers (ADMM) algorithm show its effectiveness for both linear and nonlinear systems.**

Manuscript received 24 May 2024; accepted 28 August 2024. Date of publication 11 September 2024; date of current version 14 March 2025. This article was recommended for publication by Associate Editor A. M. Mangini and Editor M. P. Fanti upon evaluation of the reviewers' comments. *(Corresponding author: Danilo Menegatti.)*

The authors are with the Department of Computer, Control, and Management Engineering "Antonio Ruberti" (DIAG), Sapienza University of Rome, 00185 Rome, Italy (e-mail: menegatti@diag.uniroma1.it; giuseppi@diag.uniroma1.it; pietrabissa@diag.uniroma1.it).

Digital Object Identifier 10.1109/TASE.2024.3453668

***Index Terms*— Model predictive control, deep neural networks, mixed-integer convex programming, data-driven control.**

I. Introduction

MODEL Predictive Control (MPC) is a paradigm capable of combining closed-loop stability with performance optimization and input/state constraint satisfaction [1]. MPC relies on a so-called *receding horizon* procedure to iteratively solve an optimization problem over a finite prediction horizon to determine a state-dependent control action that uses a model to evaluate the future behaviour of the controlled system. Being a model-based control scheme, in MPC the quality and effectiveness of the determined control directly depends on the available prediction model, as constraint satisfaction and performances are evaluated on its predicted system trajectories.

For this reason, in order to implement an MPC controller, it is first necessary to conduct an adequate identification study to develop a suitable model that correctly captures the dynamics of the controlled system. In the typical setting, the understanding of the physical/logical phenomena governing system dynamics is combined with a parameter identification procedure based on the observation of the real system. Over the years, a significant effort has been spent in developing solutions that rely solely on data-driven analysis [2], [3] or pair it with a simplified modelling process [4], as extracting knowledge – or *learning* – from data allows for compensation for model uncertainties, restrictive model classes and unaccounted disturbances. The advantages of learning typically come at the cost of reduced guarantees of closed-loop properties such as stability, safety and robustness and/or lower performance, as it is common to rely on locally optimal solutions [5], [6].

In general, the quality of a model is proportional to its complexity, as a complex model, involving for example nonlinear functions and a large number of parameters, is more capable of accurately capturing complex system dynamics. In MPC, as the controller is required to solve optimization problems in real-time, the trade-off between the model complexity and its accuracy becomes critical. In fact, several applications have to settle for a sub-optimal control law due to the unavailability of a tractable model, requiring to resort to linearized and/or local models.

The call for complex, yet computationally tractable, models has led several studies to explore the combination of deep

learning and MPC. In fact, thanks to their nonlinear structure and large number of parameters, deep Neural Networks (NNs) proved to be one of the most powerful approximation tools currently available, finding application in many control systems domains.

The present work introduces a data-driven approach to implement a neural MPC controller in which the prediction model is composed by a series of cascading deep NN-based one-step predictors. This approach has several advantages over using a single deep NN as the prediction model, in terms of

- *Interpretability*. The output of each one-step predictor is the (predicted) system state, as a function of the NN input (i.e., the previous state-control pair). Therefore, the state predictions explicitly appear in the MPC optimization problem over the whole prediction horizon. Furthermore, including the state and control action constraints of the MPC problem is simply achieved by defining constraints on the input and output of the single one-step predictors.
- *Optimization*. The resulting MPC optimization problem structure is favourable for the employment of efficient iterative optimization algorithms (such as the Alternating Direction Method of Multipliers (ADMM) [7], already used in the context of model-based MPC [8]) with a parallel implementation.
- *Tractability*. In the case in which the one-step predictor is modelled by a deep NN with ReLU and linear activation functions, the MPC optimization one can be cast as a Mixed Integer Convex Programming (MICP) problem without approximations. Furthermore, the algorithm complexity does not depend on the linearity or nonlinearity of the underlying system.
- *Training*. The training of the whole deep NN prediction model reduces to that of the single and much smaller deep NN modeling the one-step predictor.

In the remainder of the paper, Section II presents a review of the state-of-the-art on the use of deep NN within the MPC framework; Section III states the required definitions and existing results; Section IV describes the proposed learning-based MPC scheme; Section V shows the ADMM implementation; Section VI illustrates the results of numerical simulations to validate the proposed approach on linear and nonlinear systems; finally, Section VII draws the conclusions and outlines future work.

Notation: Standard notation is used in the paper, with $\hat{x}$ denoting the value of x predicted by the NN, $x(t+k|t)$ denoting the future value of x, k time-steps ahead, computed on information available at time t, and $f(\cdot;\theta)$ denoting that the function f is parametrized by θ. The matrices I_n and $\mathbb{0}_{n\times m}$ denote the $n \times n$ identity matrix and a $n \times m$ matrix of zero elements, respectively. $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. The indicator function with respect to a set A is defined as $\mathbb{I}_A(x) = \begin{cases} 0 & \text{if } x \in A \\ \infty & \text{otherwise.} \end{cases}$

II. Related Works and Proposed Innovations

Originally introduced in the 1970s for the industrial context [1], over the years MPC has been extended to handle not only linear, but also nonlinear and multi-variable systems in the presence of constraints, disturbances and model uncertainties [1]. The popularity of MPC is mostly due to its high performance in the industrial setting, as its receding horizon procedure allows to combine the benefits of both open-loop and feedback control schemes by pairing a finite horizon optimization with an iterative state/feedback measurement.

Most of the recent studies on the use of deep NNs in MPC schemes can be mapped onto one of three categories: (i) works that use a NN to approximate or augment the system dynamics, thus using the NN as the MPC prediction model, as in this work; (ii) works that use a NN to approximate an MPC controller to directly implement a control law, similarly to Reinforcement Learning schemes; (iii) works that employ a NN to obtain the estimation of the future values of exogenous disturbances/references (see e.g., [9], [10], [11], [12]), typically in black-box fashion.

Interestingly, several of the first works [13], [14], [15], [16] use neural networks to develop prediction models, whereas a significant portion of the later works present in the literature either use deep NN to mimic/implement an MPC controller or enhance standard MPC solutions with data-driven predictions. In particular, the authors of [13] used a fairly simple neural network to approximate the dynamics of a nonlinear chemical reactor and derive a control action using an extended version of the Dynamic Matrix Control (DMC) algorithm, surpassing the performance of a PID scheme without assuming any particular structure for the system dynamics; in [14] the authors test various deep NN architectures, including recurrent neural networks, to predict the behaviour of an underwater vehicle, whereas the authors of [15] proposed a variant of the Generalized Predictive Control (GPC) algorithm [17] for settings in which the plant prediction model is constituted by a multi-layer feedforward neural network. A first survey of the various architectures used in the nineties is provided in [16].

A complementary approach has been followed in other studies (e.g., [18], [19], [20], [21], [22], [23], [24], [25]), where the deep NNs has been used to synthesize the controller directly. In works such as [20], [21], [22], [23], [24], and [25], this is done by training the deep NNs off-line to replicate an MPC controller so that the iterative optimization is replaced by a much faster inference of the deep NN that also makes use of the generalization capabilities of deep NNs. We mention the approach followed in [23], in which the authors approximate an explicit MPC control law starting from a given state by means of a feedforward neural network with Rectified Linear Units (ReLUs), which is then forced to generate feasible control inputs by projecting its outputs onto convex constraint regions derived from the maximal control invariant set of the system via the Dykstra's projection algorithm. The work in [23] differentiates our study as this work avoids the need for projecting the control actions on any kind of safe set, as the MPC constraints are included in an optimization problem.

Regarding works that use deep NNs to capture the system dynamics – as in the present paper – the recent literature has focused on developing more refined architectures and exploring the conditions under which some formal analysis

of the properties of the resulting neural controller can be carried-out. Several studies, including works such as [26], [27], [28], [29], [30], [31], and [32], explored how various kinds of recurrent NNs (e.g., Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU) networks) can be used as models in the context of MPC. In fact, recurrent NNs are dynamical systems themselves, as they are characterized by evolving internal states and have been tailored to analyze time series and sequence data (e.g., control or state trajectories). Recurrent NN have hence been studied extensively in the scope of system and control theory [33], [34], [35] for their ability to learn complex nonlinear dynamics. On the contrary, feedforward deep NNs, which have a significantly simpler and non-dynamic architecture, have been used as prediction models in works such as [9], [36], [37], and [38], offering the advantage of a much simpler training and optimization processes. Among the various recurrent NNs, LSTM have likely found the most applications in the context of MPC (e.g., see [26] and the references therein), as their architecture, specifically designed to capture both long and short-term dynamics, allows them to produce accurate predictions even when employing fairly small networks. The present paper focuses mostly on using feedforward deep NNs as prediction models, but, as we will discuss, our general architecture and approach can accommodate for different NN predictors, including LSTMs, at the cost of reduced tractability (i.e., increased complexity) and interpretability.

One of the main limitations that affects the majority of the works available in the literature is the fact that NN-based prediction models are often treated as black boxes, with only some recent works exploring how analytical properties of the deep NN (e.g., the Input-to-State Stability (ISS) of the LSTM model [31]) can translate into properties of the closed-loop system (e.g., offset-free tracking of constant references [32]). As a consequence, many studies have relied on numerical methods and gradient descent-like procedures to solve the MPC optimization, even in cases in which the NN architecture has been designed to be easily interpretable as in [30]. Indeed, several recent results rely on worst-case approaches such as tube-based MPC [39], [40] to integrate neural predictors in classic control schemes while maintaining some assurance in terms of performance and stability, restricting the role and capabilities of neural predictors and leading to more conservative control laws (e.g., by limiting the neural network to estimate only an additive term of the nominal control law to compensate for model mismatches). On the contrary, the present work aims to develop an *ad-hoc* deep NN architecture that allows for the seamless interpretation of its structure and requires minimal training and optimization efforts.

For our work, we take inspiration from the procedure introduced in the recent study [41], where a novel operator splitting method is proposed, relying on ADMM [7], for the problem of neural network verification in terms of output performance satisfaction in the presence of input perturbations. We mention that the authors of [41] make use of convex relaxations of the NN activation functions, which would not be suitable to assure the feasibility of the solutions of the constrained optimization problems found in MPC, while in this work we develop a procedure that relies on the exact reformulation of ReLU activation functions.

III. Preliminaries

A. Preliminaries on Model Predictive Control

We consider a discrete-time model $x(t+1) = f(x(t), u(t))$ of a controlled dynamical system, where $x \in \mathbb{X} \subseteq \mathbb{R}^{n_x}$ is the state, $u \subseteq \mathbb{U} \in \mathbb{R}^{n_u}$ is the control action and $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is a Lipschitz function. At a given instant of time t, the MPC controller task is to find the control sequence over a prediction horizon of N_p time-steps.

Let $\mathbf{x}(t) = [x(t+1|t), \dots, x(t+N_p|t)]^T$ and $\mathbf{u}(t) = [u(t|t), \dots, u(t+N_p-1|t)]^T$ be the vectors of the future states and control actions over the prediction horizon, respectively, computed at time t. By defining a cost function and by introducing state and input constraints, an optimal control problem at time t is then defined:

Problem 1: Let $x(t) \in \mathbb{X}$ be the system state at time t, let $J : \mathbb{R}^{N_p n_x} \times \mathbb{R}^{N_p n_u} \to \mathbb{R} \cup \{+\infty\}$ be a function convex in both arguments and let $\mathbb{X}^i \subseteq \mathbb{X}$ and $\mathbb{U}^i \subseteq \mathbb{U}$ be closed convex sets. The convex optimization problem to be solved at time t in an MPC setting is

$$\min_{\mathbf{x}(t),\mathbf{u}(t)} J\big(\mathbf{x}(t), \mathbf{u}(t)|x(t)\big) \tag{1a}$$

$$\text{s.t.} \quad x(t+i|t) = f\big(x(t+i-1|t), u(t+i-1|t)\big), \tag{1b}$$

$$x(t+i|t) \in \mathbb{X}^i, \tag{1c}$$

$$u(t+i-1|t) \in \mathbb{U}^i, \quad i = 1, \dots, N_p; \tag{1d}$$

$$x(t|t) = x(t), \tag{1e}$$

where (1c) and (1d) are the state and input constraints.

The optimal solution is denoted by $\mathbf{u}^*(t) = [u^*(t|t), u^*(t+1|t), \dots, u^*(t+N_p-1|t)]^T$; only the first optimal control action is fed to the controlled system, i.e., $u(t) = u^*(t|t)$. At time $t+1$, the optimization problem is solved again, starting from the measured state $x(t+1)$ (*receding horizon* approach).[1]

Typically, as assumed in the remainder of the paper, the cost function (1a) is separable into N_p independent terms $J^i(x(t+i|t), u(t+i-1|t))$, one per time-step.

B. Preliminaries on Neural Networks

A feedforward deep NN can be thought of as an approximator of a nonlinear function, and is constituted by the weighted connection of a set of *neurons*, typically organized in arrays called *layers* so that the neurons of a given layer feed their output to the neurons of the following layer. Each neuron is characterized by an *activation function* producing an output that depends on the weighted sum of the neuron inputs.

Let us consider a NN composed by l layers; the first and final layers are the *input layer* and *output layer*, whereas the others are the *hidden layers*. Let $\varphi_{jk} : \mathbb{R} \to \mathbb{R}$ be the activation function of the k-th neuron of the j-th layer with n_j neurons, and let $s_j = [\sigma_{j1}, \dots, \sigma_{jn_j}]^T \in \mathbb{R}^{n_j}$ be the vector collecting

[1] In Problem (1), the control horizon is equal to the prediction horizon N_p. The case $N_u < N_p$ can be tackled by adding $N_p - N_u$ constraints $u(t+i|t) = u(t+i-1|t)$, for all $i = N_u, \dots, N_p - 1$.

the output of the n_j neurons of layer j. Neuron k output is then

$$\sigma_{jk} = \varphi_{jk}(w_{jk}s_{j-1} + b_{jk}), \tag{2}$$

where $w_{jk} \in \mathbb{R}^{n_{j-1}}$ is the row-vector of weights and $b_{jk} \in \mathbb{R}$ is an additional parameter known as bias. Let θ_j collect the parameters (weights and biases) of the neurons of layer j. The layer function $\phi_j : \mathbb{R}^{n_{j-1}} \to \mathbb{R}^{n_j}$ is then defined as the vector-valued function, parameterized by θ_j, such that

$$s_j = \phi_j(s_{j-1}; \theta_j) = \begin{bmatrix} \varphi_{j1}(w_{j1}s_{j-1} + b_{j1}) \\ \dots \\ \varphi_{jn_j}(w_{jn_j}s_{j-1} + b_{jn_j}) \end{bmatrix}. \tag{3}$$

The NN parameters $\theta = \{\theta_j\}_{j=1,\dots,l}$ define the function approximator and their values are identified through a data-driven optimization process known as *training*. In the typical setting, all the neurons belonging to a layer share the same activation function, (e.g., ReLU, sigmoid, *tanh* or linear functions).

Given a function $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ and a dataset $D = \{\{x_n, y_n\}_{n=1,\dots|D|} | y_n = f(x_n)\}$, the training task is to learn an approximating function from data. The NN is represented by the function $f_{NN} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, parameterized by θ, defined as

$$f_{NN}(x; \theta) = \phi_l(\phi_{l-1}(\dots(\phi_1(x; \theta_1))\dots; \theta_{l-1}); \theta_l). \tag{4}$$

with $x \in \mathbb{R}^{n_x}$, $n_1 = n_x$ and $n_l = n_y$. The training task is to learn the best parameters θ for f_{NN} to approximate f.

The number of layers, the number of neurons per layer and the choice of activation functions are directly related to the performance of a NN. Generally, the approximation capability of a NN increases with the number of hidden layers and of neurons (universal theorem of approximation [42]).

In this paper, NNs with ReLU activation functions for the input and hidden layers and with linear output activation functions are considered. This kind of NNs is widely used in regression problems as the ReLU functions allow to train deep NN with a large number of layers minimizing the effects of training-related practical issues such as vanishing gradients.

C. Preliminaries on ADMM

The Alternating Direction Method of Multipliers (ADMM) [7] is an operator splitting method suitable to solve convex optimization problems.

Leveraging on the decomposability of dual ascent algorithms and converge properties of the method of multipliers, it solves problems in the form:

$$\min_{y,z} \; J_y(y) + J_z(z) \tag{5a}$$

$$\text{s.t.} \quad Ay + Bz = d \tag{5b}$$

with $y \in \mathbb{R}^{n_y}$, $z \in \mathbb{R}^{n_z}$, $d \in \mathbb{R}^{n_c}$, where n_c is the number of constraints, and with convex cost functions $J_y : \mathbb{R}^{n_y} \to \mathbb{R} \cup \{\infty\}$ and $J_z : \mathbb{R}^{n_z} \to \mathbb{R} \cup \{\infty\}$. Note that the optimization variables have been split into y and z, with the objective function assumed to be separable across this splitting.

Similarly to the method of multipliers, the augmented Lagrangian is considered, defined as

$$L_\rho = J_y(y) + J_z(z) + m^T(Ay + Bz - d) + \frac{\rho}{2}||Ay + Bz - d||_2^2 \tag{6}$$

where m is the vector of the dual variables and $\rho > 0$ is the penalty parameter.

As in dual ascent algorithms, the problem is solved in an iterative way, for $k = 0, 1, \dots$:

$$y^{k+1} = \text{argmin}_y L_\rho(y, z^k, m^k), \tag{7a}$$

$$z^{k+1} = \text{argmin}_z L_\rho(y^{k+1}, z, m^k), , \tag{7b}$$

$$m^{k+1} = m^k + \rho(Ay^{k+1} + Bz^{k+1} - d). \tag{7c}$$

The stopping criterion relies on the computation of the primal and dual residuals, defined as $r^k = Ay^k + Bz^k - d$ and $s^k = \rho A^T B\left(z^k - z^{k-1}\right)$, and of two thresholds, i.e., the absolute and relative tolerances, denoted by $\varepsilon_{abs} > 0$ and $\varepsilon_{rel} > 0$ (see [43], [44] for details). Other algorithm parameters are $\tau > 1$ and $\mu > 1$, used to update the penalty parameter ρ as

$$\rho^{k+1} = \begin{cases} \tau\rho^k & \text{if } ||r^k||_2 > \mu||s^k||_2, \\ \rho^k/\tau & \text{if } ||s^k||_2 > \mu||r^k||_2, \\ \rho^k & \text{otherwise.} \end{cases} \tag{8}$$

The convergence of the algorithm is guaranteed under two assumptions: 1) the functions J_y and J_z are closed, proper, and convex; 2) the augmented Lagrangian has a saddle point. Practically, the algorithm works well even in problems which do not meet the convergence assumptions, with on-going research on ADMM for mixed-integer programming problems [45], [46].

IV. Proposed Model Predictive Control With One-Step Neural Network Predictors

In this section, we consider the problem of controlling a system with unkown or unmodeled dynamics. To synthesize a suitable control law, we resort to identifying through a data-driven analysis a *neural representation* of its dynamics.

The section is organized as follows: Section IV-A presents the prediction model built from the one-step predictors; Section IV-B formalizes the MPC optimization problem with the developed prediction model; Section IV-C details the case of NNs with ReLU and linear activation functions.

A. Prediction Model With One-Step Neural Networks

As introduced in Section III-A, MPC requires a model of the process dynamics $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ to predict the trajectory followed by the state of the system along a prediction horizon. This section describes the feedforward NN that has the task of approximating the function f. With reference to an MPC problem with a prediction horizon of N_p time-steps, the NN, denoted by f_{NN}, is referred to as *one-step neural predictor*.

Definition 1: The one-step neural predictor is defined as the function $f_{NN} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, such that, at time t,

$$\hat{x}(t+1|t) = f_{NN}\big(x(t), u(t); \theta\big), \tag{9}$$

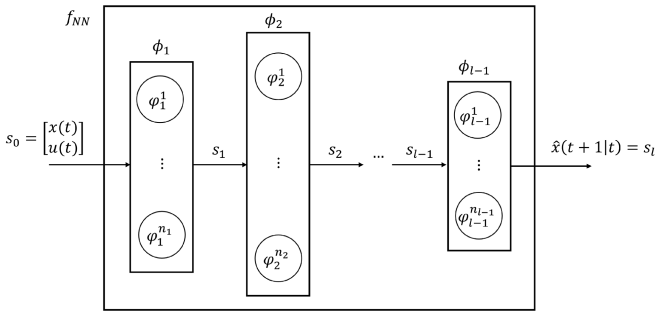

Fig. 1. Graphical representation of the feedforward neural network f_{NN} playing the role of a one-step predictor. The sizes n_j of the inner layers, i.e., the number of neurons, are project choices, whereas the size of the input layer is $n_x + n_u$ and the size of the output layer is n_x.

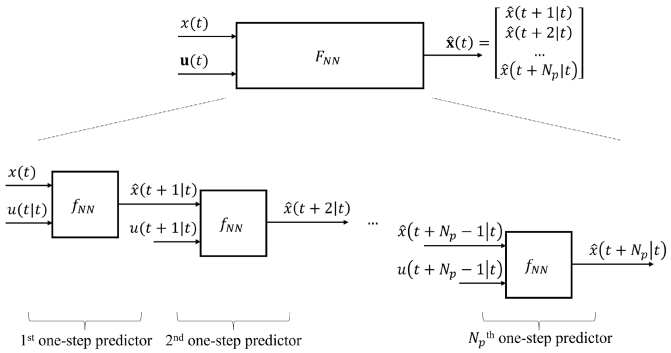

Fig. 2. Graphical representation of the deep NN F_{NN}, built from the one-step predictor f_{NN} and constituting the prediction model.

where $\hat{x}(t+1|t)$ is the estimated value of the system state x predicted at time t and θ collects the NN parameters.

As described in Section III-B, the NN relies on the activation functions. Let l the number of layers of the one-step predictor. The output of the k-th neuron of layer j is denoted by $\sigma_{jk} = \varphi_{jk}(w_{jk}\sigma_{jk} + b_{jk})$, for $k = 1, \dots, n_j$ and $l = 1, \dots, l$, where n_j is the number of neurons in layer j, and the weights w_{jk} and the bias b_{jk} are parameters whose values are the result of the training phase. The input of the NN is $s_0 = \left[x^T(t), u^T(t)\right] \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$ and the output is $\hat{x}(t+1|t) = s_l \in \mathbb{R}^{n_x}$. The total number of neurons is denoted by $n_w = \sum_{j=1,\dots,l} n_j$, and the one-step neural predictor is depicted in Fig. 1.

A common choice for activation functions in regression problems is the use of a linear function for the output layer, as it does not restrict the output space to any specific interval, and ReLU functions for all other layers.

The ReLU activation function of neuron k of layer j is

$$\varphi_{jk}\left(w_{jk}s_{j-1} + b_{jk}\right) = \max\left\{0, w_{jk}s_{j-1} + b_{jk}\right\}. \tag{10}$$

The linear activation function of the neuron k in the output layer l is

$$\varphi_{lk}\left(w_{lk}s_{l-1} + b_{lk}\right) = w_{lk}s_{l-1} + b_{lk}. \tag{11}$$

Remark 1: As the trained one-step predictor is set up, i.e., as the neural network weights w_{jk} and biases b_{jk} are found, it is possible to identify upper- and lower-bounds for the input of each neuron by feeding the one-step predictor with the available data and checking its inner signals. Considering all the input pairs of the neural network in the training set Σ, let $\bar{M}_{jk}$ and $\underline{M}_{jk}$ be positive constants such that

$$-\underline{M}_{jk} < \varphi_{jk}(w_{jk}\sigma_{jk} + b_{jk}) < \bar{M}_{jk}, \quad \forall [x, u]^T \in \Sigma. \tag{12}$$

These bounds will be useful in the following Sections.

The NN in equation (9) is used in the prediction model as a one-step predictor. To obtain the overall prediction model, i.e., the trajectory predictions over the whole prediction horizon, N_p one-step predictors are concatenated.

Definition 2: The neural prediction model at time t is

$$\hat{\mathbf{x}}(t) = F_{NN}\left(x(t), \mathbf{u}(t); \theta\right), \tag{13}$$

where $\hat{\mathbf{x}}(t) = \left[\hat{x}(t+1|t), \dots, \hat{x}(t+N_p|t)\right]^T$ is the vector of the states predicted by the NN and $F_{NN} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{N_p n_x}$ is a vector-valued function built from the concatenation of the one-step predictors (9):

$$F_{NN}\left(x(t), \mathbf{u}(t); \theta\right) = \begin{bmatrix} f_{NN}\left(x(t), u(t|t); \theta\right) \\ f_{NN}\left(f_{NN}\left(x(t), u(t|t); \theta\right), u(t-1|t)\right) \\ \dots \\ f_{NN}\left(f_{NN}\left(\dots\left(f_{NN}\left(x(t), u(t|t); \theta\right), u(t-1|t); \theta\right), \right.\right. \\ \left.\left.\dots\right) u(t+N_p-1|t); \theta\right) \end{bmatrix}. \tag{14}$$

The prediction model is represented in Fig. 2, which highlights the interpretability of the deep NN since its structure allows the identification of the state variables at each step of the prediction horizon.

B. Optimization Problem With Neural Network Prediction Model

If the neural pedictor f_{NN} is available instead of the function f, the first constraints (1b) of the optimization Problem (1) is substituted by

$$\hat{x}(t+i|t) = f_{NN}\left(\hat{x}(t+i-1|t), u(t+i-1|t)\right), \tag{15}$$

for $i = 1, \dots, N_p$. The resulting problem will be referred to as NN optimization problem.

The inner structure of the one-step predictor f_{NN}, described by equation (14), can be made explicit in the constraints of the NN optimization problem.

Problem 2: The NN optimization problem to be solved at each time-step t with initial state $x(t) = x$ reads as

$$\min_{s} \quad \sum_{i=1,\dots,N_p} J^i(s_l^i, \Lambda_u s_0^i) \tag{16a}$$

$$\text{s.t.} \quad s_j^i = \phi_j\left(s_{j-1}^i\right), \quad j = 1, \dots, l, \tag{16b}$$

$$s_l^i \in \mathbb{X}^i, \tag{16c}$$

$$s_0^i \in \mathbb{X}^i \times \mathbb{U}^i, \quad i = 1, \dots, N_p; \tag{16d}$$

$$\Lambda_x s_0^1 = x, \tag{16e}$$

$$\Lambda_x s_0^i = s_l^{i-1}, i = 2, \dots, N_p, \tag{16f}$$

with $\Lambda_x = \left[I_{n_x} | \mathbb{0}_{n_x \times n_u}\right]$ and $\Lambda_u = \left[\mathbb{0}_{n_u \times n_x} | I_{n_u}\right]$.

In Problem 2, considering that s_0^i and s_l^i are the input and output of the i-th one-step predictor, it holds that

$\Lambda_x s_0^i = \hat{x}(t+i-1|t)$, $\Lambda_u s_0^i = u(t+i-1|t)$ and $s_l^i = \hat{x}(t+i|t)$. The constraints of Problem 2 recast the ones of Problem 1:

- constraints (15) are cast as constraints (16b), representing the dynamics of each one-step predictor i, and (16f), linking the input of the i-th one-step predictor to the output of the $(i-1)$-th one;
- (16e) states the initial conditions (1e);
- constraints (16c)-(16d) express the input and output constraints of each one-step predictor, i.e., the state and control input constraints (1c)-(1d).

Remark 2: Problem 2 highlights the structure of the neural prediction model, i.e., the fact that it is the concatenation of one-step NNs, with constraints (16b) describing the dynamics of each one-step predictor and constraints (16f) which are the coupling constraints. The NN optimization problem at time t can be thought as a set of N_p smaller, "local" MICP sub-problems, aimed at minimizing $J^i(\hat{x}(t+i|t), u(t+i-1|t))$ subject to constraints (16b)-(16d), for $i = 1, \ldots, N_p$, with initial conditions set by the coupling constraints (16f) and, for the first sub-problem, by the initial state (16e). This structure is favorable for the possibility to apply efficient solution algorithms employing decomposition-coordination procedures, in which the solutions to the local sub-problems are coordinated to find a solution to the larger coupled problem. Decomposition-coordination algorithms are iterative algorithms alternating the decomposition step, in which the local sub-problems are solved in parallel, with the coordination step, in which the coupling constraints are dealt with.

Remark 3: The convergence speed of the decomposition-coordination algorithms is sensible to the initial guess. Generally, the receding horizon approach of MPC is such that a reliable initial guess at time-step $t+1$ is retrieved from the solution of the optimization problem at time-step t. A reliable initial guess can be computed by starting from the partial feasible solution obtained by the optimal solution at time t (for instance, for the predicted state, we would set the initial guess as $\hat{x}(t+i+1|t+1) = \hat{x}(t+i|t)$, for $i = 2, \ldots, N_p - 1$).

C. The Case of ReLU and Linear Activation Functions

The benefits of the proposed one-step predictor approach are valid in case of general activation functions, since the related constraints (16b) appear within the local sub-problems. However, depending on the activation functions, the constraints might be non-convex and their handling require specific optimization techniques – for instance, sigmoid or tanh functions the constraints might require convex or piecewise linear approximations. The use of ReLU activation functions (10) for the input and inner layers and linear activation functions (11) for the output layer is common in regression problems, as the former exhibits sparsity for negative inputs and mitigates the vanishing gradient problem, while the latter allows for a direct interpretation of the NN's output. With this choice, Problem 2 becomes a Mixed Integer Convex Programming (MICP) problem, since constraints (16b) are written as convex ones without the need of approximations:

$$\sigma_{jk}^i = \max\left\{0, w_{jk} s_{j-1}^i + b_{jk}\right\}, \quad k = 1, \ldots, n_j,\ j = 0, \ldots, l-1, \tag{17a}$$

$$s_l^i = w_l s_{l-1} + b_l, \tag{17b}$$

with $w_l = \left[w_{l1}^T, \ldots, w_{ln_l}^T\right]^T$ and $b_l = \left[b_{l1}, \ldots, b_{ln_l}\right]^T$.

By introducing a binary decision variable $\delta \in \{0, 1\}$, a constraint of the type $y = \max\{0, x\}$, where x and y are two scalar unknowns, can be rewritten as the set of 4 linear constraints:

$$y \geq 0, \quad y \geq x, \quad y \leq M\delta, \quad y \leq x + M(1-\delta), \tag{18}$$

where M is a "large" constant such that $x < M$ for any value of x. This kind of formulation is known in the literature as "big-M" reformulation [47]. In general, the upper-bound M is chosen as "large enough" based on the problem knowledge. However, the performance of the solvers depends on the choice of the upper-bound: if it is too large, it can lead to numerical instability or slow convergence, whereas if it is too small the accuracy of the solution might be affected. As discussed in Remark 1, the NN modelling proposed in Section III-A allows to derive a reasonable choice of the "big-M" constant for each neuron of the one-step predictor.

For all neurons k, layers j and predictors i, a binary unknown $\delta_{jk}^i \in \{0, 1\}$ is then introduced to formulate the optimization Problem 2 as a Mixed-Integer Convex Programming (MICP) one - or Mixed-Integer Quadratic Programming (MIQP) in case of quadratic cost functions.

Problem 3: The NN optimization problem with ReLU input and inner layers and with linear output layer is a MICP, written as Problem 2 with constraints (16b) substituted by

$$\sigma_{jk}^i \geq 0, \tag{19a}$$

$$\sigma_{jk}^i \geq (w_{jk} s_{j-1}^i + b_{jk}), \tag{19b}$$

$$\sigma_{jk}^i \leq \bar{M}_{jk} \delta_{jk}, \tag{19c}$$

$$\sigma_{jk}^i \leq (w_{jk} s_{j-1}^i + b_{jk}) + \bar{M}_{jk}(1-\delta_{jk}), \tag{19d}$$

$$\delta_{jk}^i \in \{0, 1\}, \quad k = 1, \ldots, n_j,\ j = 1, \ldots, l-1; \tag{19e}$$

$$s_l^i = w_l s_{l-1}^i + b_l, \tag{19f}$$

In Problem 3, for each one-step predictor $i = 1, \ldots, N_p$ the constraints are such that constraints (19a)-(19e) express the dynamics (16b) of the ReLU neurons of the input and inner layers under the "big-M" reformulation (see equations (18)), with $\bar{M}_{jk}$ evaluated as in Remark 1.

V. ADMM Solution to the NN Optimization Problem

This section describes how iterative decomposition-coordination algorithms can be used to efficiently solve Problems 2 and 3. Specifically, Section V-A shows that, at each iteration, the proposed architecture enables parallel optimization in the decomposition step, whereas Section V-B shows that the constraints of Problem 3 can be recast as linear equality constraints to apply the ADMM algorithm (5).

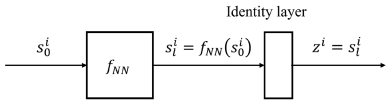

Fig. 3. Use of identity layers to decouple the one-step predictors.

A. Step 1: Definition of Identity Output Layers

We now introduce an unknown vector z, collecting the proxy variables $z^i \in \mathbb{R}^{n_x}$ acting as copies of s_l^i, for $i = 1, \dots, N_p - 1$. Introducing the variables z^i is equivalent to introducing an identity layer after the output of the i-th one-step predictor (see Figure 3), with the aim of decoupling the inner dynamics of the one-step predictors. With the introduction of the proxy variables, Problem 2 is written as follows:

Problem 4: The NN optimization Problem 2 with decoupled dynamics of the one-step predictors is written as

$$\min_{s} \quad \sum_{i=1,\dots,N_p} J^i(s_l^i, \Lambda_u s_0^i) \tag{20a}$$
$$\text{s.t.} \quad s_j^i = \phi_j(s_{j-1}^i), \quad j = 1, \dots, l, \tag{20b}$$
$$s_0^i \in \mathbb{X}^i \times \mathbb{U}^i, \tag{20c}$$
$$s_l^i \in \mathbb{X}^i, \quad i = 1, \dots, N_p; \tag{20d}$$
$$s_l^i - z^i = 0, \quad i = 1, \dots, N_p - 1; \tag{20e}$$
$$\Lambda_x s_0^1 = x, \tag{20f}$$
$$\Lambda_x s_0^i = z^{i-1}, i = 2, \dots, N_p. \tag{20g}$$

In Problem 4, constraints (20b)-(20d) describe the independent (*decoupled*) dynamics of the N_p one-step predictors, constraints (20e) describe the identity layers, and constraints (20f) and (20g) collects the initial conditions (16e) and the coupling constraints (16f), respectively, written in terms of the proxy variables. This approach is inspired by [41], where identity layers are added between subsequent NN layers to split the original problem into smaller sub-problems, one per NN layer. Similarly, our proxy variables, added between subsequent one-step predictors, will allow to decompose Problem 4 into N_p sub-problems (Section V-B).

Remark 4: In principle, the proposed approach can be adapted for arbitrary NNs (e.g., LSTM), as the structure of Problem 4 can be preserved by substituting constraints (20b) with

$$s_l^i = f_{NN}(s_0^i), \tag{21}$$

where f_{NN} represents the dynamics of each one-step predictor independently from its architecture. However, constraints (21) are generally not convex for most NN architectures and activation functions.

B. Step 2: ADMM Formulation

To use the ADMM algorithm, the inequality constraints must be re-formulated as equality constraints. As in Problem 3, by using the layer models of Section III-B with ReLU and linear activation functions, constraints (20b) of problem 4 are expressed by the linear constraints (19a)-(19e), with the introduction of the vector of integer variables δ. We further introduce the vector of non-negative slack variables ξ to write the 3 sets of inequalities (19b)-(19d) as equality ones,[2] obtaining the following problem:

Problem 5: The NN optimization Problem 2, with decoupled dynamics of the one-step predictors, ReLU input and inner layers and linear output layer, is written as Problem 4, with constraints (20b) expressed by the following constraints:

$$\sigma_{jk}^i \geq 0, \tag{22a}$$
$$-\sigma_{jk}^i + w_{jk} s_{j-1}^i + b_{jk} + \xi_{jk1}^i = 0, \tag{22b}$$
$$\sigma_{jk}^i - \bar{M}_{jk}\delta_{jk} + \xi_{jk2}^i = 0, \tag{22c}$$
$$\sigma_{jk}^i - (w_{jk} s_{j-1}^i + b_{jk}) - \bar{M}_{jk}(1 - \delta_{jk}) + \xi_{jk3}^i = 0, \tag{22d}$$
$$\xi_{jkp}^i \geq 0, \quad p = 1, 2, 3, \tag{22e}$$
$$\delta_{jk}^i \in \{0, 1\}, \quad k = 1, \dots, n_j, \, j = 1, \dots, l - 1; \tag{22f}$$
$$s_l^i = w_l s_{l-1}^i + b_l. \tag{22g}$$

By collecting the variables s^i, δ^i and ξ^i into the vector $y^i = \left[{s^i}^T, {\delta^i}^T, {\xi^i}^T\right]^T$, the optimization problem is written in matrix form[3] as

$$\min_{y} \quad \sum_{i=1,\dots,N_p} \tilde{J}^i(y^i) \tag{23a}$$
$$\text{s.t.} \quad A^i y^i + B^i z = d^i, \quad i = 1, \dots, N_p, \tag{23b}$$

where:

- the cost function $\tilde{J}^i(y^i)$ is defined as

$$\tilde{J}^i(y^i) = J^i(s_l^i, \Lambda_u s_0^i) + \mathbb{I}_{\mathbb{R}^{n_w}_{\geq 0}}(s^i) + \mathbb{I}_{\mathbb{R}^{n_\xi}_{\geq 0}}(\xi^i) + \\ + \mathbb{I}_{\{0,1\}^{n_\delta}}(\delta^i) + \mathbb{I}_{\mathbb{X}^i \times \mathbb{U}^i}(s_0^i) + \mathbb{I}_{\mathbb{X}^i}(s_l^i), \tag{24}$$

  where $n_\xi = 3(n_w - n_l)$, $n_\delta = n_w - n_l$ and the indicator functions are used to include the convex constraints (22a), (22e), (22f), (20c) and (20d), respectively;
- constraints (23b) represent, in matrix form, the linear equality constraints (22b)-(22d), (22g), and the initial conditions (20f), if $i = 1$, or (20g), otherwise.

Problem (23) resembles the ADMM one (5). The augmented Lagrangian (6) of the ADMM is now written as

$$L_\rho(y, z) = \sum_{i=1,\dots,N_p} \left(\tilde{J}^i(y^i) + J_1^i(y^i, z, m^i)\right), \tag{25}$$

where

$$J_1^i(y^i, z, m^i) = m^i(A^i y^i + B^i z - d^i) + \\ + \frac{\rho}{2}\|A^i y^i + B^i z - d^i\|_2^2 \tag{26}$$

and the m^i's are the dual variables, collected in the vector m.

Algorithm 1: To solve Problem 5 by using the ADMM algorithm (4), the following optimization problems are solved iteratively, starting from an initial guess (y^0, z^0, m^0):

$${y^i}^{k+1} = \text{argmin}_{y^i}\left(\tilde{J}^i(y^i) + J_1^i(y^i, z^k, {m^i}^k)\right), \\ i = 1, \dots, N_p, \tag{27a}$$

[2]The constraint $ay + bz \leq d$ can be written as $ay + bz + \xi = d$ with slack variable $\xi \geq 0$.

[3]See the Appendix for the definitions of the matrixes.

$$z^{k+1} = \text{argmin}_z \sum_{i=1,\dots,N_p} J_1^i(y^{i\,k+1}, z, m^{i\,k}), \tag{27b}$$

$$m^{i\,k+1} = m^{i\,k} + \rho(A^i y^{i\,k+1} + B^i z^{k+1} - d^i), \quad i = 1, \dots, N_p, \tag{27c}$$

for $k = 0, 1, 2, \dots$, until the primal and dual residuals are sufficiently small (see Section III-C).

Remark 5: In Step 2 of Algorithm 1, (27b) is a simple unconstrained convex (or quadratic) program and, in Step 3, (27c) consists in straightforward computations. In Step 1, (27a) describes N_p MICP (or MIQP) problems which, thanks to the developed prediction model structured with one-step predictors, can be solved in parallel, improving the overall algorithm complexity, run-time and scalability for large values of the prediction horizon.

VI. Numerical Simulations

This section presents some numerical simulation results with two examples: a linear system and a nonlinear system, in Sections VI-A and VI-B, respectively. The ADMM parameters were set as follows: $\varepsilon_{abs} = 10^{-3}$, $\varepsilon_{rel} = 10^{-4}$ and initial value of the parameter $\rho^0 = 10$, which is updated by means of residual balancing (8) with $\mu = 10$ and $\tau = 1.2$. The simulations were carried out on a machine learning machine equipped with an Intel Core i9 9900k processor with 128 GB of RAM and a Nvidia GeForce RTX 3090 as GPU.

A. Example With Linear System

This section considers an unstable discrete-time linear system, described by the function $f(x, u) = Mx + Nu$, with matrices $M = \begin{bmatrix} 1.1 & 0.5 \\ -0.5 & 1.1 \end{bmatrix}$ and $N = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. A standard MPC approach was implemented, with quadratic cost function

$$J(x, u) = \sum_{i=1}^{N_p} \left(\|x(t+i|t)\|_Q + \|u(t+i-1|t)\|_R \right) + \\ + \|x(t+N_p|t)\|_P, \tag{28}$$

with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $R = 10$, $P = \begin{bmatrix} 15.2814 & -0.9740 \\ -0.9740 & 5.1226 \end{bmatrix}$ and $N_p = 15$. The input and state variables are bounded as $u \in [-\bar{u}, \bar{u}]$, with $\bar{u} = 0.25$ and $x_1, x_2 \in [-\bar{x}, \bar{x}]$, with $\bar{x} = 0.65$. The matrix P was calculated in such a way that the final cost, i.e. the last term of the cost function, is equal the optimal cost of a LQR, according to the dual-mode paradigm [48]: provided that N_p is large enough, the solution found by the MPC algorithm is then optimal for the infinite-horizon constrained optimization problem.

The same linear system was then used to generate a database of about 10^5 input-output pairs, with input $[x \quad u]^T$ and output $x' = f(x, u)$, used, in turn, to train two NNs for implementing the Neural Network One-Step Predictor $f_{NN}(x)$. The structure of the first NN has one ReLU input layer, 2 ReLU inner layers and one linear output layer; the other NN is a LSTM composed of 128 units followed by a linear layer. Data were generated by computing trajectories with length equal to 100 time-steps with random initial state $x \in [-3\bar{x}, 3\bar{x}]$ and random initial input $u \in [-3\bar{u}, 3\bar{u}]$, this last kept constant over the whole trajectory (the NN dynamics was trained in broader conditions with respect to the constrained one). The training set and the test set were selected as 90% and 10% of the data, respectively.

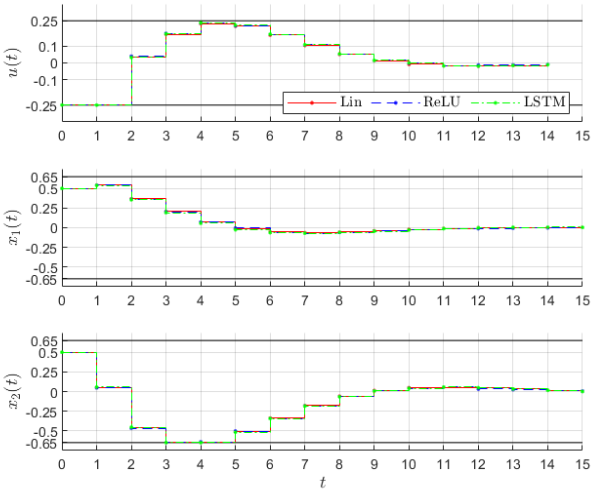

Fig. 4. Linear system: control actions and state trajectories with upper- and lower-bounds.

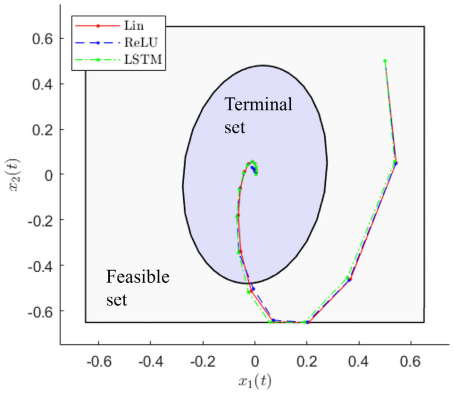

Fig. 5. Linear system: state trajectories with feasible and terminal sets.

The training of $f_{NN}(x)$ is done for $E = 5$ epochs considering a batch size $b = 100$, using Stochastic Gradient Descent (SGD) as the optimizer. As is common in regression problems, the Mean Squared Error (MSE) is chosen as both loss function and performance metric. After E epochs, the Neural Network One-Step Predictor was able to predict the state evolution with a prediction error lower than 0.1%.

Figures 4 and 5 collect the simulation results with initial conditions $x(0) = [0.5 \quad 0.5]^T$. The "Lin" plots show the solution $\mathbf{u}^*$ of the MPC problem (1) with the linear model $f(x, u) = Mx + Nu$ and the resultant state trajectories.

The "ReLU" plots show the solutions of the neural MPC problem 3, i.e., the sequences of control actions $\mathbf{u}^*$, and the state trajectories predicted by the neural network model $f_{NN}(x)$, with ReLU activation functions for the inner layers.

The "LSTM" plots are obtained by using the LSTM neural model as $f_{NN}(x)$ in Problem 4, with constraints (21) considered instead of (20b). The ADMM algorithm was then used

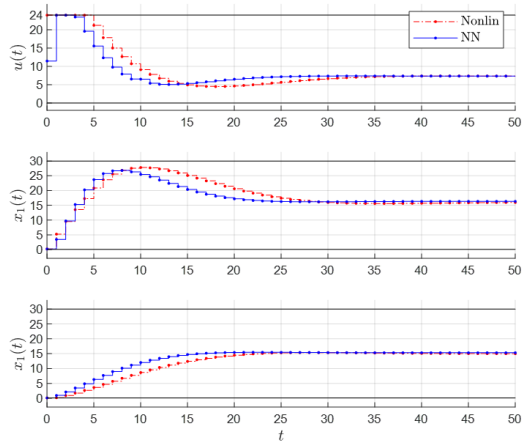

Fig. 6. Nonlinear system: control actions and state trajectories with upper- and lower-bounds; $R = 0.1$.

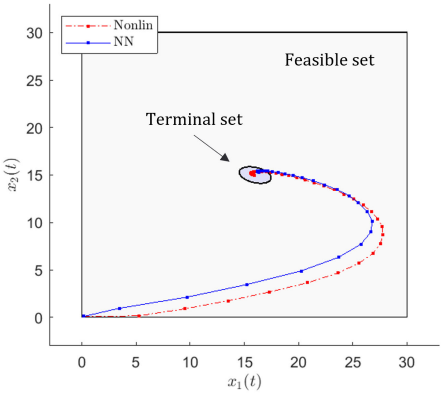

Fig. 7. Nonlinear system: state trajectories; $R = 0.1$.

to solve the following problem:

$$\min_{s^i} \sum_{i=1,\dots,N_p} J^i(s_l^i, \Lambda_u s_0^i) + \mathbb{I}_{\mathbb{X}^i \times \mathbb{U}^i}(s_0^i) + \mathbb{I}_{\mathbb{X}^i}(s_l^i) \tag{29a}$$

$$\text{s.t.} \quad F^i s^i + G^i z = v^i, \quad i = 1, \dots, N_p, \tag{29b}$$

where constraints (29b) represent, in matrix form, the initial conditions (20f), if $i = 1$, or (20g), otherwise. In the iterative procedure (analogous to the one described in Algorithm 1), the parallel problems of Step 1 had to be solved numerically.

The trajectories of the NN implementations are evaluated using the various NNs at time 0 starting from the state initial conditions $x(0)$. Despite the use of a data-driven solution without any prior knowledge of the system, the figures show that the plots obtained with the neural MPC are close to the plots obtained with the optimal control actions $\mathbf{u}^*$, meet both input and state constraints and guarantee the system stability by driving the state trajectories towards the final set.

Table I collects some key performance indicators of the simulation. By calculating the optimization problem (OP) dimension as the number of unknowns times the number of constraints, the table also shows that the size of the MIQP Problem 3 obtained with the ReLU NN is two orders of magnitude larger than the original QP Problem 1. However, by using the ADMM algorithm, the MIQP problem is recast to the one of iteratively solving in parallel $N_p = 15$ instances of

TABLE I

LINEAR SYSTEM: SIZE OF THE OP PROBLEMS, NUMBER OF PROBLEMS TO SOLVE, SOLVING TIME, NUMBER OF ITERATIONS AND COST FUNCTION VALUE OF MPC (QP), CENTRALIZED NEURAL MPC (MIQP), ITERATIVE NEURAL MPC (ADMM)

Algorithm	OP size	N. of OP	Time	n_{iter}	J
QP (Lin)	$1.2 \cdot 10^3$	1	0.02s	–	5.462
MIQP (ReLU)	$1.8 \cdot 10^5$	1	$\sim$ 1s	–	5.462
ADMM (ReLU)	$0.9 \cdot 10^3$	15	0.01s	37	5.504
ADMM (LSTM)	$0.8 \cdot 10^3$	15	0.05s	41	5.532

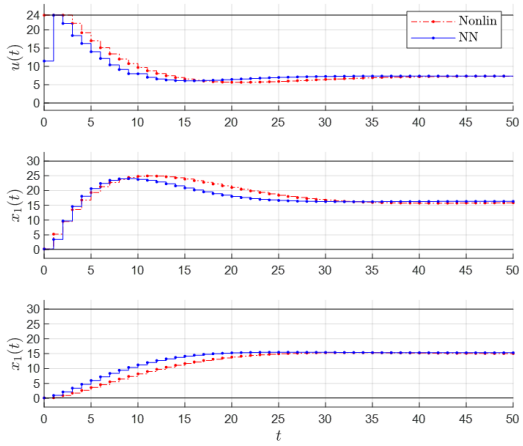

Fig. 8. Nonlinear system: control actions and state trajectories with upper- and lower-bounds; $R = 0.2$.

the MIQP Problem (27a), with smaller dimension and lower solving time with respect to the original QP Problem 1. The reduction in problem size is significant and allows the practical application of the proposed procedure to find the solution of Problem 3. With LSTM, the problem size is similar but the optimization problem has nonlinear constraints. Finally, the table reports the solving time and number of iterations needed to find the solution with the ADMM algorithm when the LSTM is used as predictor; in this case the parallel problems are solved numerically, slowing down the solving time (see Remark 4).

As the MIQP problem formulation obtained for the ReLU NN is equivalent to the original QP problem, they both obtain the optimal cost. A sub-optimal solution is obtained with the ADMM algorithm: the cost function value worsens by 0.78% for the ReLU solution and 1.28% for the LSTM one (probably penalized by the need to use numerical solvers). We mention that the ADMM performance might be improved by refining the NN models (e.g., by increasing the size/quality of training database, improving the tuning of the NN hyper-parameters or increasing the number and/or size of the NN layers).

B. Example With Nonlinear System

This Section considers the coupled tank model reported in [49], described by the nonlinear system

$$\begin{cases} x_1(t+1) = x_1(t) - \delta \dfrac{A_1}{A}\sqrt{2g x_1(t)} + \delta \dfrac{k_p}{A} u(t) \\ x_2(t+1) = x_2(t) - \delta \dfrac{A_2}{A}\sqrt{2g x_2(t)} + \delta\sqrt{2g x_1(t)}. \end{cases} \tag{30}$$

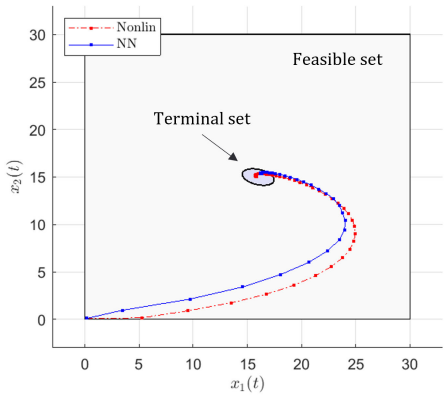

Fig. 9. Nonlinear system: state trajectories; $R = 0.2$.

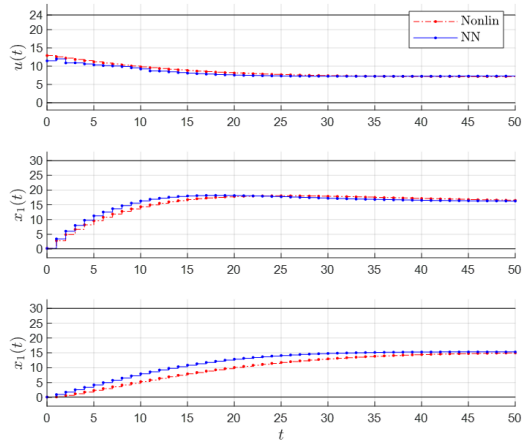

Fig. 10. Nonlinear system: control actions and state trajectories with upper- and lower-bounds; $R = 2$.

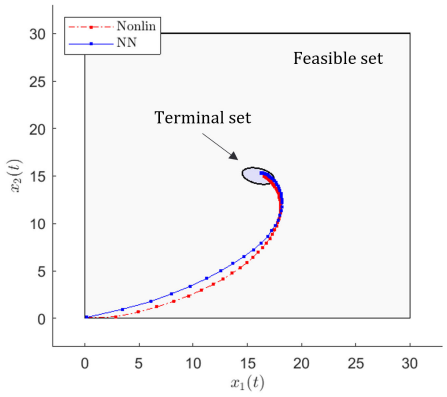

Fig. 11. Nonlinear system: state trajectories; $R = 2$.

where x_1 and x_2 are the depths of liquid in a pair of connected tanks, $A = 15.2$ cm^2 is the tank cross-section area, $A_1 = 0.13$ cm^2 and $A_2 = 0.14$ cm^2 are the outflow orifices areas, $g = 981$ cm s^{-2} is acceleration due to gravity, $k_p = 3.3$ cm^3 s V is the pump gain, $\delta = 1.4$ s is the time-step. The control task is to stabilize the system around a constant level of liquid in the tanks, with $x_1^r = 15.9$, $x_2^r = 15$ cm, in the presence of inflows and outflows. The height of the tanks is $\bar{x} = 30$ cm and the pump voltage, playing the role of the control action u, is bounded by $\bar{u} = 24$ V.

TABLE II
NONLINEAR SYSTEM: NUMBER OF PROBLEMS TO SOLVE, SOLVING TIME, NUMBER OF ITERATIONS AND COST VALUES FOR DIFFERENT VALUES OF R WITH MPC (NONLIN) AND ITERATIVE NEURAL MPC (ADMM)

	Nonlinear	ADMM (ReLU)
N. of OP	1	50
Time	$\sim$ 1s	0.01s
n_{iter}	70	59
$J(R = 0.1)$	11501	12573
$J(R = 0.2)$	11769	12785
$J(R = 2)$	16125	17269

For comparison purposes, the model-based nonlinear MPC approach of [49] was implemented, with cost function (28), with state cost $Q = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ cm^{-2}, control cost $R = 0.2$ V^{-2}, terminal cost $P = \begin{bmatrix} 2.1 & 1.2 \\ 1.2 & 5.9 \end{bmatrix}$ cm^{-2} and $N_p = 50$.

The same nonlinear system was then used to generate a database of input-output pairs $\{[x \quad u]^T, f(x,u)\}$, with f given by (30), used, in turn, to train a Neural One-Step Predictor $f_{NN}(x)$ with the same procedure used for the linear system case in Section VI-A. The ReLU NN has the same structure of the one developed for the linear system case and was trained on 10^6 input-output pairs.

The nonlinear MPC algorithm of [49], needed to control the nonlinear system (30), is much more complex with respect to the linear one of the former section and requires an iterative procedure (see [49] for details). Conversely, the single problem (27a) is exactly the same as the one of the linear example in section IV-A, with same problem size, as the same structure for the one-step predictor f_{NN} was used. However, the number N_p of problems to be solved in parallel and the number of iterations required to reach the solution are larger. Table II collects the simulation characteristics.

Figures 6-11 shows the closed-loop application of the MPC strategies, considering the real process as identical to the nonlinear model, with initial conditions $x(0) = [0.2 \quad 0.1]^T$ cm and desired final state $[15 \quad 15]^T$ cm. The results are shown for different values of R (see the figure captions).

The figures show that the trajectories obtained with the neural MPC are close to the ones obtained with the model-based MPC, meet both input and state constraints and are steered towards the final state. Table II shows the performance of the two approaches: for the different values of R, the difference in the cost function values ranges between 6.6% and 8.5%. In evaluating these results, we underline again that the proposed neural MPC approach is purely data-driven, whereas the model-based MPC exploits the perfect knowledge of the system model since no uncertainties or disturbances were introduced in the simulations: the presence of a model mismatch could significantly worsen the performance of the model-based approach; conversely, if needed, the performance of the neural MPC might be improved by using a more refined NN for the one-step predictor.

VII. CONCLUSION AND FUTURE WORKS

In the context of data-driven MPC, the paper proposes an approach enabling the tractable solution of the MPC optimal

$$A^i = \begin{bmatrix} \boldsymbol{w}^i & -I_{n_h} & \mathbb{0}_{n_h\times n_x} & \mathbb{0}_{n_h} & I_{n_h} & \mathbb{0}_{n_h} & \mathbb{0}_{n_h} \\ \mathbb{0}_{n_h\times(n_x+n_u)} & I_{n_h} & \mathbb{0}_{n_h\times n_x} & -\bar{\boldsymbol{M}} & \mathbb{0}_{n_h} & I_{n_h} & \mathbb{0}_{n_h} \\ -\boldsymbol{w}^i & I_{n_h} & \mathbb{0}_{n_h\times n_x} & \bar{\boldsymbol{M}} & \mathbb{0}_{n_h} & \mathbb{0}_{n_h} & I_{n_h} \\ \mathbb{0}_{n_x\times(n_x+n_u)} & -\boldsymbol{w}_l & I_{n_x} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} \\ \Lambda_x & \mathbb{0}_{(n_x+n_u)\times n_h} & \mathbb{0}_{(n_x+n_u)\times n_h} & \mathbb{0}_{(n_x+n_u)\times n_h} & \mathbb{0}_{(n_x+n_u)\times n_h} & \mathbb{0}_{(n_x+n_u)\times n_h} & \mathbb{0}_{(n_x+n_u)\times n_h} \\ \mathbb{0}_{n_x\times(n_x+n_u)} & \mathbb{0}_{n_x\times n_h} & I_{n_x} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} & \mathbb{0}_{n_x\times n_h} \end{bmatrix},$$

$$B^i = \begin{bmatrix} & \mathbb{0}_{n_h\times n_x N_p} & \\ & \mathbb{0}_{n_h\times n_x N_p} & \\ & \mathbb{0}_{n_h\times n_x N_p} & \\ & \mathbb{0}_{n_x\times n_x N_p} & \\ \mathbb{0}_{n_x\times n_x(i-2)} & \beta^i & \mathbb{0}_{n_x\times n_x(N_p-i+1)} \\ \mathbb{0}_{n_x\times n_x(i-1)} & -I_{n_x} & \mathbb{0}_{n_x\times n_x(N_p-i)} \end{bmatrix}, \quad d^i = \begin{bmatrix} -\boldsymbol{b}_{-l} \\ \mathbb{0}_{n_h\times 1} \\ \boldsymbol{b}_{-l} + \bar{\boldsymbol{M}} \\ \boldsymbol{b}_l \\ \gamma^i \\ \mathbb{0}_{n_x} \end{bmatrix},$$

control problem using efficient iterative algorithms. This study relies on the development of a prediction model by concatenating a set of multi-layer Neural Networks (NNs) used as one-step predictors. This structure has a twofold advantage: (i) the optimal control problem can be formulated as a convex optimization with linear constraints, under the assumption that the NN employs ReLU and linear activation functions, regardless of the linear or nonlinear nature of the controlled system; (ii) the optimal solution can be found by means of efficient iterative and parallel optimization algorithms, such as the ADMM algorithm used in the numerical simulations that validate the approach.

Future work is aimed at analysing the convergence properties of iterative optimization algorithms with the provided problem formulation and at extending the formulation to include other activation functions and NN architectures to further improve the interpretability of the overall control loop.

APPENDIX
MATRICES OF THE EQUALITY CONSTRAINTS OF THE ADMM MICP FORMULATION

The matrices A^i, B^i and d^i of Problem (23) are, shown in the equation at the top of the page, where

$$\begin{aligned} \bar{\boldsymbol{M}} &= \mathrm{col}_{N_p}\left([\bar{M}_{jk}]_{k=1,\dots,n_j,\,j=1,\dots,l-1}\right), \\ \mathbf{b}_l &= \mathrm{col}_{N_p}\left(b_l\right), \\ \mathbf{b}_{-l} &= \mathrm{col}_{N_p}\left([b_{jk}]_{k=1,\dots,n_j,\,j=1,\dots,l-1}\right), \\ \gamma^i &= \begin{cases} x & \text{if } i=1 \\ \mathbb{0}_{n_x\times 1} & \text{if } i=2,\dots,N_p, \end{cases} \\ \beta^i &= \begin{cases} \mathbb{0}_{n_x} & \text{if } i=1 \\ -I_{n_x} & \text{if } i=2,\dots,N_p. \end{cases} \end{aligned}$$

REFERENCES

[1] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[2] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer, "Data-driven tracking MPC for changing setpoints," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6923–6930, 2020.

[3] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer, "Data-driven model predictive control with stability and robustness guarantees," *IEEE Trans. Autom. Control*, vol. 66, no. 4, pp. 1702–1717, Apr. 2021.

[4] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. A data-driven control framework," *IEEE Trans. Autom. Control*, vol. 63, no. 7, pp. 1883–1896, Jul. 2018.

[5] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Inf. Sci.*, vol. 235, pp. 3–35, Jun. 2013.

[6] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annu. Rev. Control, Robot., Auto. Syst.*, vol. 3, no. 1, pp. 269–296, May 2020.

[7] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[8] S. East and M. Cannon, "ADMM for MPC with state and input constraints, and input nonlinearity," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 4514–4519.

[9] B. Vatankhah and M. Farrokhi, "Nonlinear model-predictive control with disturbance rejection property using adaptive neural networks," *J. Franklin Inst.*, vol. 354, no. 13, pp. 5201–5220, Sep. 2017.

[10] M. Sadeghassadi, C. J. B. Macnab, B. Gopaluni, and D. Westwick, "Application of neural networks for optimal-setpoint design and MPC control in biological wastewater treatment," *Comput. Chem. Eng.*, vol. 115, pp. 150–160, Jul. 2018.

[11] G. Coccia, A. Mugnini, F. Polonara, and A. Arteconi, "Artificial-neural-network-based model predictive control to exploit energy flexibility in multi-energy systems comprising district cooling," *Energy*, vol. 222, May 2021, Art. no. 119958.

[12] P. Kumar, J. B. Rawlings, M. J. Wenzel, and M. J. Risbeck, "Grey-box model and neural network disturbance predictor identification for economic MPC in building energy systems," *Energy Buildings*, vol. 286, May 2023, Art. no. 112936.

[13] A. Draeger, S. Engell, and H. Ranke, "Model predictive control using neural networks," *IEEE Control Syst. Mag.*, vol. 15, no. 5, pp. 61–66, Oct. 1995.

[14] V. S. Kodogiannis, P. J. G. Lisboa, and J. Lucas, "Neural network modelling and control for underwater vehicles," *Artif. Intell. Eng.*, vol. 10, no. 3, pp. 203–212, Aug. 1996.

[15] D. Soloway and P. J. Haley, "Neural generalized predictive control," in *Proc. IEEE Int. Symp. Intell. Control*, Sep. 1996, pp. 277–282.

[16] S. Piche, B. Sayyar-Rodsari, D. Johnson, and M. Gerules, "Nonlinear model predictive control using neural networks," *IEEE Control Syst. Mag.*, vol. 20, no. 3, pp. 53–62, Jun. 2000.

[17] D. W. Clarke, C. Mohtadi, and P. S. Tuffs, "Generalized predictive control—Part I. The basic algorithm," *Automatica*, vol. 23, no. 2, pp. 137–148, Mar. 1987.

[18] J. G. Ortega and E. F. Camacho, "Mobile robot navigation in a partially structured static environment, using neural predictive control," *Control Eng. Pract.*, vol. 4, no. 12, pp. 1669–1679, Dec. 1996.

[19] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller," *J. Process Control*, vol. 16, no. 9, pp. 937–946, Oct. 2006.

[20] Y. Cao and R. B. Gopaluni, "Deep neural network approximation of nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11319–11324, 2020.

[21] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and fast tracking on a robot manipulator: Robust MPC and neural network control," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 3050–3057, Apr. 2020.
[22] E. T. Maddalena, C. G. D. S. Moraes, G. Waltrich, and C. N. Jones, "A neural network architecture to learn explicit MPC controllers from data," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11362–11367, 2020.
[23] S. Chen et al., "Approximating explicit model predictive control using constrained neural networks," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 1520–1527.
[24] D. Wang et al., "Model predictive control using artificial neural network for power converters," *IEEE Trans. Ind. Electron.*, vol. 69, no. 4, pp. 3689–3699, Apr. 2022.
[25] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3866–3878, Sep. 2020.
[26] M. Jung, P. R. da Costa Mendes, M. Önnheim, and E. Gustavsson, "Model predictive control when utilizing LSTM as dynamic models," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106226.
[27] N. Lanzetti, Y. Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangöz, and C. Jones, "Recurrent neural network based MPC for process industries," in *Proc. 18th Eur. Control Conf. (ECC)*, Jun. 2019, pp. 1005–1010.
[28] K. Patan, "Neural network-based model predictive control: Fault tolerance and stability," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 3, pp. 1147–1155, May 2015.
[29] Y. Pan and J. Wang, "Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks," *IEEE Trans. Ind. Electron.*, vol. 59, no. 8, pp. 3089–3101, Aug. 2012.
[30] I. Lenz, R. Knepper, and A. Saxena, "DeepMPC: Learning deep latent features for model predictive control," in *Proc. Robot., Sci. Syst. XI*, Jul. 2015.
[31] E. Terzi, F. Bonassi, M. Farina, and R. Scattolini, "Learning model predictive control with long short-term memory networks," *Int. J. Robust Nonlinear Control*, vol. 31, no. 18, pp. 8877–8896, Dec. 2021.
[32] F. Bonassi, C. F. O. da Silva, and R. Scattolini, "Nonlinear MPC for offset-free tracking of systems learned by GRU neural networks," *IFAC-PapersOnLine*, vol. 54, no. 14, pp. 54–59, 2021.
[33] C. Bailer-Jones, D. MacKay, and P. Withers, "A recurrent neural network for modelling dynamical systems," *Netw., Comput. Neural Syst.*, vol. 9, no. 4, pp. 531–547, Nov. 1998.
[34] K.-I. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Netw.*, vol. 6, no. 6, pp. 801–806, Jan. 1993.
[35] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2018.
[36] A. Fatehi, H. Sadjadian, A. Khaki-Sedigh, and A. Jazayeri, "Disturbance rejection in neural network model predictive control," *IFAC Proc. Volumes*, vol. 41, no. 2, pp. 3527–3532, 2008.
[37] P. Kittisupakorn, P. Thitiyasook, M. A. Hussain, and W. Daosud, "Neural network based model predictive control for a steel pickling process," *J. Process Control*, vol. 19, no. 4, pp. 579–590, Apr. 2009.
[38] M. A. Hosen, M. A. Hussain, and F. S. Mjalli, "Control of polystyrene batch reactors using neural network based model predictive control (NNMPC): An experimental investigation," *Control Eng. Pract.*, vol. 19, no. 5, pp. 454–467, May 2011.
[39] T. Zieger, A. Savchenko, T. Oehlschlägel, and R. Findeisen, "Towards safe neural network supported model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5246–5251, 2020.
[40] D. D. Fan, A.-A. Agha-Mohammadi, and E. A. Theodorou, "Deep learning tubes for tube MPC," 2020, *arXiv:2002.01587*.
[41] S. Chen, E. Wong, J. Z. Kolter, and M. Fazlyab, "DeepSplit: Scalable verification of deep neural networks via operator splitting," *IEEE Open J. Control Syst.*, vol. 1, pp. 126–140, 2022.
[42] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989.
[43] B. S. He, H. Yang, and S. L. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *J. Optim. Theory Appl.*, vol. 106, no. 2, pp. 337–356, Aug. 2000.
[44] S. L. Wang and L. Z. Liao, "Decomposition method with a variable parameter for a class of monotone variational inequality problems," *J. Optim. Theory Appl.*, vol. 109, no. 2, pp. 415–429, May 2001.
[45] A. Alavian and M. C. Rotkowitz, "Improving ADMM-based optimization of mixed integer objectives," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2017, pp. 1–6.
[46] Z. Liu and O. Stursberg, "Distributed solution of mixed-integer programs by ADMM with closed duality gap," in *Proc. IEEE 61st Conf. Decis. Control (CDC)*, Dec. 2022, pp. 279–286.
[47] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*. Philadelphia, PA, USA: SIAM, 2009.
[48] D. Q. Mayne, M. M. Seron, and S. V. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, Feb. 2005.
[49] M. Doff-Sotta and M. Cannon, "Difference of convex functions in robust tube nonlinear MPC," in *Proc. IEEE 61st Conf. Decis. Control (CDC)*, Dec. 2022, pp. 3044–3050.

Danilo Menegatti (Student Member, IEEE) received the master's degree in control engineering from the University of Rome "La Sapienza" in 2020, where he is currently pursuing the Ph.D. degree in automatic control, bioengineering and operations research with the Department of Computer, Control, and Management Engineering "Antonio Rubert" (DIAG). His research interests involve intelligent systems, distributed learning, and reinforcement learning applications.

Alessandro Giuseppi (Senior Member, IEEE) received the master's degree in control engineering and the Ph.D. degree in automatica from the University of Rome "La Sapienza" in 2016 and 2019, respectively. He is currently an Assistant Professor (RTDa) with the University of Rome "La Sapienza." Since 2016, he has been participating in eight EU and national research projects in the field of automatic control and artificial intelligence. His main research activities are in the fields of network control and intelligent systems, where he has published about 80 papers in international journals and conferences.

Antonio Pietrabissa (Senior Member, IEEE) received the degree in electronics engineering and the Ph.D. degree in systems engineering from the University of Rome "La Sapienza" in 2000 and 2004, respectively. He is currently an Associate Professor with the University of Rome "La Sapienza," where he teaches automatic control and process automation. Since 2000, he has been participating in more than 30 EU and national research projects. He serves as an Associate Editor for IEEE TRANSACTIONS AUTOMATION SCIENCE AND ENGINEERING and *Control Engineering Practice* (Elsevier). He is the author of more than 60 journal articles and 80 conference papers. His research focuses on the control of networks.

Open Access funding provided by 'Università degli Studi di Roma "La Sapienza" 2' within the CRUI CARE Agreement