Learning affine predictors for MPC of nonlinear systems via artificial neural networks ⋆

Daniele Masti * **Francesco Smarra** ** **Alessandro D'Innocenzo** **
Alberto Bemporad *

* *IMT School for Advanced Studies, Lucca, Italy*
(e-mail: {daniele.masti, alberto.bemporad}@imtlucca.it).
** *Università degli Studi dell'Aquila, L'Aquila, Italy*
(e-mail: {francesco.smarra, alessandro.dinnocenzo}@univaq.it).

Abstract: Nonlinear model predictive control (MPC) problems can be well approximated by linear time-varying (LTV) MPC formulations in which, at each sampling step, a quadratic programming (QP) problem based on linear predictions is constructed and solved at runtime. To reduce the associated computation burden, in this paper we explore and compare two methodologies for learning the entire output prediction over the MPC horizon as a nonlinear function of the current state but affine with respect to the sequence of future control moves to be optimized. Such a learning process is based on input/output data collected from the process to be controlled. The approach is assessed in a simulation example and compared to other similar techniques proposed in the literature, showing that it provides accurate predictions of the future evolution of the process and good closed-loop performance of the resulting MPC controller. Guidelines for tuning the proposed method to achieve a desired memory occupancy / quality of fit tradeoff are also given.

Keywords: Model Predictive Control, Artificial Neural Networks, System Identification, Deep Learning, Random Forests

1. INTRODUCTION

Designing a control law directly from experimental data has always been a topic of central interest in the control systems community. The classical approach is to first identify a model of the open-loop process via system identification (Ljung, 1987) from experimental data, validate it, and then proceed with the design of a model-based controller. The main advantage of this approach is that the identification and control design phases are decoupled. However, unnecessary burden due to excessively accurate system identification procedures might occur. In fact, many control design techniques do not need a very accurate model of the target process to synthesize the control law: for instance when designing a Model Predictive Controller (MPC) (Borrelli et al., 2017) there is no need of having a model that predicts accurately the evolution of the output beyond the prediction horizon. In other words, getting the best possible accuracy of the model in fitting the available input/output data is often not required for good closed-loop performance. In addition, an accurate fit often comes at the price of an excessive model complexity, that in turn makes model-based control laws more complicate (or even impossible) to synthesize.

The separation between system identification and model-based control design is also clear from the fact that, while in recent years many state-of-the-art system identification techniques focus on learning input-output models (see (Kocijan et al., 2005; Wang, 2017; Pillonetto et al., 2014)), most of the existing modern control and filtering techniques are based on state-space models. Many authors have tried to bridge this gap, we mention here the recent contribution (Masti and Bemporad, 2018) that proposed a method for learning a nonlinear state-space model of desired order from input/output data, with the goal of synthesizing state-feedback controllers such as nonlinear MPC controllers.

Recently, *model-free* data-driven techniques were proposed to completely bypass the phase of first identifying an open-loop model of the process, such as using direct controller synthesis methods (Piga et al., 2017), machine learning (Jain et al., 2017, 2018; Afram et al., 2017), policy-search (Ferrarotti and Bemporad, 2019), and reinforcement learning (Gros and Zanon, 2020). While very attractive in practice, the main drawbacks of model-free methods are the need for large amounts of data to synthesize optimal control laws and the different way one tunes the controller compared to model-based control approaches.

Main contribution and related work.

In this paper we explore a different way of approaching data-driven control design, tailoring the system identification procedure to the particular use one wants to make of the resulting model for control design. In particular, to handle nonlinear control problems within the linear time-varying (LTV) MPC framework, such as the real-time iteration scheme (Diehl et al., 2005), we explore how to fit the entire output prediction for LTV-MPC over a horizon of T steps as a nonlinear function of the current state (or, equivalently, past input/output pairs) and as an affine function of future control moves. In this way, the MPC problem can be solved via quadratic programming (QP) in spite of the nonlinearity of the system. This approach has

⋆ This work was partially supported by the Italian Government under Cipe resolution n.135 (Dec. 21, 2012), project INnovating City Planning through Information and Communication Technologies (INCIPICT), and by the project VALU3S (call: H2020-ECSEL-2019-2-RIA).

been explored in various forms by many authors but, as a whole, the literature on the topic is both scarse and scattered. In (Behl et al., 2016; Smarra et al., 2018a), the authors proposed to learn fixed-horizon affine models via regression trees (RT) and random forests (RF) for nonlinear systems; in (Terzi et al., 2019) an approach based on set-membership (SM) identification was devised to learn robust predictor for linear systems. The authors in (Liu and Kadirkamanathan, 1998) addressed instead the problem of building an adaptive control scheme in which fixed-horizon predictors based on Radial-Basis-Function (RBF) Artificial Neural Networks (ANNs) were learned online. Another comparable approach is the one shown in Afram et al. (2017), where the authors used an ANN to identify the dynamics of the system and setup an associated MPC problem. However, the resulting ANN model is nonlinear, so the MPC problem requires a more complex nonlinear solution method, while our approach requires only a quadratic programming solver, or even admits a simple analytic solution in the absence of constraints.

In this work we investigate if ANN-based solutions can be competitive with traditional machine learning approaches to learn fixed short-term horizon predictors for receding horizon control applications. To this end, we perform an in-depth comparison between the RT/RF-based approaches presented in (Smarra et al., 2018b, 2020) and modern reinterpretation of the idea presented in (Liu and Kadirkamanathan, 1998). In particular, with respect to this latter work, we focus on replacing the original non-parametric single-layer RBF-ANN architecture with a specifically tailored state-of-the-art deep-learning technique and introduce a completely offline learning procedure in place of the original online learning scheme.

Paper organization. After formulating the problem in Section 2, in Section 3 we present a specifically deep-learning based parametrization to learn a fixed-horizon affine model of a non-linear process. In Section 4 we provide a background on the methodology in (Smarra et al., 2018a), that will be used in Section 5 to compare the proposed approach via numerical simulations. Section 6 will provide guidelines for tuning the proposed approach in practical problems, with attention to reducing its computational requirements.

2. PROBLEM FORMULATION

Assume that we have collected data from a process described by the following (usually unknown) nonlinear discrete-time dynamical model

$$\Sigma = \begin{cases} x_{k+1} = f_\Sigma(x_k, u_k) \\ y_k = h_\Sigma(x_k) \end{cases} \tag{1}$$

where $k \in \mathbb{N}$ is the sampling step, $x_k \in \mathbb{R}^{n_x}$ the state of the system, $y_k \in \mathbb{R}^{n_y}$ the output vector, $u_k \in \mathbb{R}^{n_u}$ the input vector, and $f_\Sigma : \mathbb{R}^{n_u \times n_x} \to \mathbb{R}^{n_x}$, $h_\Sigma : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$. In case there is no clear definition of what the state vector x_k should be, due for example to physical insights, we assume that

$$x_k = \begin{bmatrix} y_k' & \cdots & y_{k-\delta_y}' & u_{k-1}' & \cdots & u_{k-\delta_u}' \end{bmatrix}' \tag{2}$$

for some model-order integers $\delta_y, \delta_u \geq 0$.

With the goal of synthesizing an MPC controller with prediction horizon $T \in \mathbb{N}$, we want to learn a mapping from x_k and $u_k, u_{k+1}, \ldots, u_{k+j-1}$ to y_{k+j}, where $j = 0, \ldots, T$ is the prediction step, from a dataset $\mathcal{D} = \{(y_k, u_k, x_k)\}_{k=0}^{N-1}$ collected from Σ, or simply $\mathcal{D} = \{(y_k, u_k)\}_{k=0}^{N-1}$ in case x_k just collects past inputs and outputs as in (2).

In order to avoid the recursive modeling approach commonly used in the literature (see for instance Afram et al. (2017)), we focus ourselves on creating T predictors that depend only on the available state measurements/estimations and on the inputs to be optimized. More precisely, we want to identify maps $h_j : \mathbb{R}^{n_x \times jn_u} \to \mathbb{R}^{n_y}$, $j = 1, \ldots, T$, each one taking x_k and $u_k, \ldots, u_{k+j-1}$ as inputs, such that

$$y_{k+j} = h_j(x_k, u_k, \ldots, u_{k+j-1}),\ \forall\, j = 1, \ldots, T. \tag{3}$$

We can rewrite (3) in the more compact form

$$Y_T = \begin{bmatrix} y_{k+1} \\ \vdots \\ y_{k+T} \end{bmatrix} = H_T(x_k, U_{T-1}), \tag{4}$$

where $H_T : \mathbb{R}^{n_x \times Tn_u} \to \mathbb{R}^{Tn_y}$ has a block-triangular structure due to (3) and $U_{T-1} = [u_k' \ \ldots \ u_{k+T-1}']'$.

Consider now the following MPC problem at each step k

$$\begin{aligned} \min_{U_{T-1}} \quad & (Y_T - R_T)' W_Q (Y_T - R_T) + \\ & (U_{T-1} - U_{T-1}^R)' W_R (U_{T-1} - U_{T-1}^R) + \delta U'\, W_S\, \delta U \\ \text{s.t. :} \quad & \\ & \hat{Y}_T = H_T(x_k, U_{T-1}) \\ & Y_{\min} \leq Y_T \leq Y_{\max} \\ & U_{\min} \leq U_{T-1} \leq U_{\max} \\ & \delta U_{\min} \leq \delta U_{T-1} \leq \delta U_{\max} \end{aligned} \tag{5}$$

where W_Q, W_R, W_S are weight matrices of appropriate dimensions, $R_T \in \mathbb{R}^{Tn_y}$ is the reference signal to track, $U_T^R \in \mathbb{R}^{Tn_u}$ is a corresponding input reference (possibly $U_T^R = 0$), $\delta U = [u_k' - u_{k-1}',\ \ldots, u_{k+T-1}' - u_{k+T-2}']'$, and $Y_{\min}, Y_{\max}$, $U_{\min}, U_{\max}$, $\delta U_{\min}, \delta U_{\max}$ are vectors of lower and upper bounds.

Problem (5) can be recast as a QP problem if we fit T ANN-based *affine* predictors of the following form

$$Y_T = F_T(x_k, \bar{U}_{T-1}) + G_T(x_k, \bar{U}_{T-1}) U_{T-1} \tag{6}$$

where $\bar{U}_{T-1}$ is a nominal input sequence, $F_T : \mathbb{R}^{n_x \times Tn_u} \to \mathbb{R}^{Tn_y}$ and $G_T : \mathbb{R}^{n_x \times Tn_u} \to \mathbb{R}^{Tn_y \times Tn_u}$.

The affine predictor (6) can be interpreted also as the first-order Taylor approximation of the non-linear mapping in (4). In fact, assuming that H_T is at least of class $\mathcal{C}^1$, we get

$$\begin{aligned} H_T(x_k, U_{T-1}) \approx & H_T(x_k, \bar{U}_{T-1}) \\ & + \sum_{j=0}^{T-1} \frac{\partial H_T}{\partial u_{k+j}}(x_k, \bar{U}_{T-1}) u_{k+j}. \end{aligned}$$

A simplified form of (6), obtained by fixing $\bar{U}_{T-1}$ a priori (for example, equal to a steady-state nominal input, or zero), is the following predictor

$$Y_T = F_T(x_k) + G_T(x_k)(U_{T-1} - \bar{U}_{T-1}) \tag{7}$$

with $F_T : \mathbb{R}^{n_x} \to \mathbb{R}^{Tn_y}$ and $G_T : \mathbb{R}^{n_x} \to \mathbb{R}^{Tn_y \times Tn_u}$.

3. TRAINING AFFINE PREDICTORS VIA ANNS

Given that the underlying system (1) generating the output signals y_{t+k} is Markovian, the components f_j, g_j of F_T, G_T are not completely independent functions between different prediction steps. Therefore, to regularize the training procedure we impose the following recursive and causal structure

$$y_{k+j} = f_j(x_k) + g_j(x_k) \begin{bmatrix} U_{j-1} - \bar{U}_{j-1} \\ y_{k+j-1} \end{bmatrix} \tag{8}$$

for $j = 2, \dots, T$. This is especially useful if we use ANN techniques (Goodfellow et al., 2016) or deep kernel machines (Suykens, 2017), that naturally benefit from stacking nonlinear layers/components.

Note that the affine nature of the predictor is maintained in (8), as the composition of affine functions remains affine. To simplify the notation, in the rest of the paper we will consider $U_{T-1} \equiv 0$.

The problem of learning the maps f_j and g_j in (8) can be posed as the following optimization problem

$$\begin{aligned} &\underset{\{f_j, g_j\}_{i=1}^{T}}{\arg\min} \quad \sum_{k=\max\{\delta_y, \delta_u\}}^{N-T} L(\hat{O}_k, O_k) \\ &\text{s.t.:} \\ &\qquad \hat{y}_{k+j} = f_j(x_k) + g_j(x_k)\,[U'_{j-1},\ \hat{y}'_{k+j-1}]' \\ &\qquad \hat{y}_{k+1} = f_1(x_k) + g_1(x_k) U_0 \\ &\qquad O_k = [y_{k+1} \ \cdots \ y_{k+T}]' \\ &\qquad \hat{O}_k = [\hat{y}_{k+1} \ \cdots \ \hat{y}_{k+T}]' \\ &\qquad j = 2, \dots, T, \end{aligned} \tag{9}$$

where the constraints in (9) impose the causal structure arising from (8) and $U_{j-1} = [u'_k, \dots, u'_{k+j-1}]'$. The loss function $L : \mathbb{R}^{Tn_y \times Tn_y} \to \mathbb{R}$ can be any loss function and can be chosen based on the physical meaning of the predicted output, for example using a cosine distance for an angular quantity or a cross entropy for a categorical output signal.

The optimization problem (9) has two conflicting objectives, due to trading off between short-term prediction accuracy and the ability to carry useful information from prediction $k + j$ to prediction $k + i$, for $i > j$. Note that learning the maps f_i, g_i, for $i = 1, \dots, T$ can be either carried out sequentially (one time step j at time) or in one shot. In this work we focus on the latter method.

The unconventional structure (8) restricts the pool of function approximators that can be employed to parameterize f_j and g_j. We exploit the nature of ANNs of being direct acyclic computational graphs to build into the topology of the network itself the constraints structure of (9). In this way, we reduce the learning procedure to a regression problem while retaining the capability of easily accessing the outputs of f_i, g_i as intermediate results of the single sub-components of the network. A schematic of the considered network is reported in Figure 1.

Choosing ANNs is also convenient for their flexibility to be trained with a wide class of loss functions (which allows one to easily add, for instance, regularizers or additional objectives) and to their theoretical property of being universal approximators (Barron, 1993), as well as to the large availability of well-maintained and mature frameworks for their training (Abadi et al., 2015; Seide and Agarwal, 2016).

ANN structure. The topology of the ANN used in this work closely follows the structure highlighted in (8). In particular, all the components f_i are grouped in a single stand-alone network, while each component g_i is instead a separate entity. In this work, we restrict our analysis to a densely connected feed-forward topology for each subnetwork, but in principle this is not mandatory. Each sub-network is thus composed by a series of nonlinear hidden layers and a final linear output layer with appropriate output shape. As in this work we analyze output signals in $\mathbb{R}^{Tn_y}$, we rely on the well known mean absolute error (MAE) figure as the loss function L:

$$L(Y, \hat{Y}) = \frac{1}{Tn_y} \|Y - \hat{Y}\|_1 \tag{10}$$

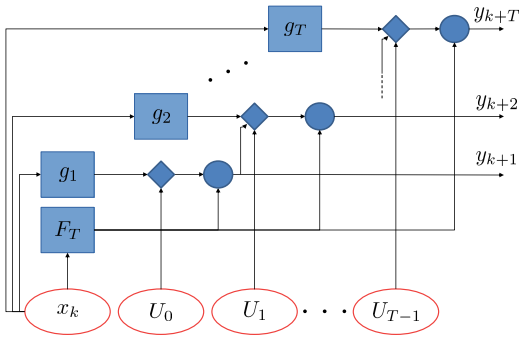

Fig. 1. ANN structure for predictions affine in the input.

4. SWITCHING AFFINE RT AND RF PREDICTORS

In this section we briefly recall the approach proposed in (Smarra et al., 2018b), recently experimentally validated in (Bünning et al., 2020), to create a switching affine modeling framework based on regression trees (RTs) and random forests (RFs), that can be used in the MPC formulation (5). For simplifying reading, we limit the discussion to scalar outputs ($n_y = 1$), although the approach can be easily extended for $n_y > 1$ as illustrated in (Smarra et al., 2018b).

We partition the original dataset $\mathcal{D}$ in 2 disjoint sets: $\mathcal{D}_c = \{u_k\}_{k=1}^{N}$, of data associated with input variables, and $\mathcal{D}_{nc} = \{x_k\}_{k=1}^{N}$ related to the remaining variables. As done in Section 3, the idea is to create T different predictors to predict y_{k+j}, and to derive a modeling framework in order to setup an MPC problem that leads to a QP.

We create T regression trees $\mathcal{T}_j$, $j = 1, \dots, T$, by applying the CART algorithm to the dataset $\mathcal{D}_{nc}$ (see (Breiman et al., 1984; Smarra et al., 2018b) for more details). In particular, for each tree $\mathcal{T}_j$, the CART algorithm partitions the set $\mathcal{D}_{nc}$ into subsets $\mathcal{D}_{nc,i,j}$, $i = 1, \dots, |\mathcal{T}_j|$, $j = 1, \dots, T$, where $|\mathcal{T}_j|$ is the number of regions of the partition associated with $\mathcal{T}_j$ (i.e., the number of leaves of $\mathcal{T}_j$). Then, using the control data in $\mathcal{D}_c$, we associate to each leaf i of each tree $\mathcal{T}_j$, corresponding to the partition $\mathcal{D}_{nc,i,j}$, the following affine model

$$x_{k+j} = A_{i_j} x_k + \sum_{\alpha=0}^{j-1} B_{i_j,\alpha} u_{k+\alpha} + f_{i_j}, \ \forall i_j, \ \forall j, \tag{11}$$

where matrices A'_{i_j}, $B'_{i_j,\alpha}$ and f'_{i_j} are in turn structured as

$$\begin{aligned} A_{i_j} &= \begin{bmatrix} a_1 & a_2 & \cdots & a_{\delta_y+1+\delta_u} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad f_{i_j} = \begin{bmatrix} f \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ B_{i_j,\alpha} &= \begin{bmatrix} b_{1,\alpha} & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ b_{n_u,\alpha} & 0 & \cdots & 0 \end{bmatrix}'. \end{aligned} \tag{12}$$

The coefficients of matrices A_{i_j}, $B_{i_j,\alpha}$ and f_{i_j} are obtained by fitting the set of samples

$$\{(x_{k+j}, u_k, \dots, u_{k+j-1}) : \; x_k \in \mathcal{D}_{nc,i,j}\} \qquad (13)$$

via least squares as defined in Problem 2 of (Smarra et al., 2018b). In particular, in order to consider the same information as in (8), we will constrain the parameters related to the input auto-regressive terms to be zero, i.e. $a_{\delta_y+2}, \dots, a_{\delta_y+1+\delta_u} = 0$.

The approach discussed above for RTs can be easily extended to the case of RFs (Breiman, 2001). The idea behind RFs is to extend the RT concept by growing multiple trees, considering different random subsets of the dataset to train each tree. The prediction is given by averaging the response of all the trees in the forest. At the price of an increased computational burden, the pros of this approach are the reduction of the overall variance of the prediction error and the mitigation of overfitting.

In our context, we create T RFs $\mathcal{F}_j$, $j = 1, \dots, T$, with $|\mathcal{F}_j|$ the number of trees of forest $\mathcal{F}_j$. We can derive a model as in (11) by solving the least squares problem introduced above for each leaf $\mathcal{D}_{i_j,\tau}$ of each tree $\mathcal{T}_\tau$, $\tau = 1, \dots, |\mathcal{F}_j|$, of each forest $\mathcal{F}_j$. In this way, we obtain a vector of parameters for the matrices in (12) associated to each leaf $\mathcal{D}_{i_j,\tau}$. The final model as in (11) can be obtained for each forest $\mathcal{F}_j$ by averaging the coefficients of all the matrices associated with $\mathcal{D}_{i_j,\tau}$.

It can be shown that for both RT and RF models (11) is a switching affine representation of (1), where the switching signal follows the partitioning imposed by the tree structures, and each leaf represents an operating mode (see Smarra et al. (2018b) for details).

5. SIMULATION RESULTS

5.1 Benchmark problem setup

Let the system Σ in (1) generating the data be the following discrete-time approximation of the well-known nonlinear two-tank system (Schoukens and Noël, 2017)

$$\begin{cases} x_{1,k+1} = x_{1,k} - k_1\sqrt{x_{1,k}} + k_2 u_k \\ x_{2,k+1} = x_{2,k} + k_3\sqrt{x_{1,k}} - k_4\sqrt{x_{2,k}} \\ y_k = x_{2,k} \end{cases} \qquad (14)$$

with $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, and $k_4 = 0.3$ and where $x_{i,k}$ represents the i-th component of $x_k \in \mathbb{R}^2_+$. On this system we analyze the performance of the affine model over an horizon of $T = 10$. To do so, we collect a training dataset $\mathcal{D}$ of $N = 10000$ samples by exciting (14) with a sequence of step signals of length 7 steps, each of random amplitude extracted from the univariate Gaussian distribution $G_u \backsim \mathcal{N}(\mu_u, \sigma_u^2)$, with $\mu_u = \sigma_u = 2$. The testing dataset is generated similarly and consist of 1000 samples. Both datasets have been normalized using the empirical mean and standard deviation computed on the training set. White zero-mean Gaussian noise with $\sigma_w = 0.02$ is superimposed on input/output signals.

Since we employ an early-stopping strategy as termination criterion for the training process of the ANN, 5% of the training dataset is used as a validation set to check the stopping criterion. Each computational node of the ANN is composed of two rectified linear units (ReLU) (Nair and Hinton, 2010) layers with 20 neurons each, followed by a final linear output layer.

As state x_k we choose past input/output values as in (2) with $\delta_y = \delta_u = 6$. The predictors are written in Python using Keras (Chollet et al., 2015) with Tensorflow as back-end, using AMSgrad (Sashank J. Reddi, 2018) for optimizing the weights. The total training procedure was carried out in around a minute on a Intel Core I5-6200U CPU machine with 16GB of RAM and required a negligible amount of memory.

Prediction step	e_{FIT}	e_{NRMSE}
1	0.957	0.99
2	0.957	0.99
3	0.95	0.989
4	0.948	0.988
5	0.94	0.986
6	0.928	0.983
7	0.914	0.98
8	0.9	0.977
9	0.88	0.972
10	0.858	0.967

Table 1. Accuracy of the affine ANN predictors for the benchmark (14)

5.2 Fitting performance

We first investigate the accuracy of the affine predictors as in (8) on the test set for the prediction horizon $T = 10$. Fitting performance over the horizon is computed in terms of FIT and NRMSE figures, defined as follows:

$$e_{\text{FIT}} = \max\left\{0, 1 - \frac{\|\hat{y} - y\|_2}{\|y - \bar{y}\|_2}\right\} \qquad (15)$$

$$e_{\text{NRMSE}} = \max\left\{0, 1 - \frac{\|\hat{y} - y\|_2}{\sqrt{\mathcal{S}_\mathcal{T}}(\max(y) - \min(y))}\right\} \qquad (16)$$

where $\hat{y}$ is the vector stacking the estimates of the true values vector y, $\bar{y}$ is the mean of y, and $\mathcal{S}_\mathcal{T}$ is the number of elements in y.

The results, reported in Table 1, show a very good prediction capability, that clearly decreases over the horizon as expected. This behavior can be also linked to the specific affine form of the predictors that is not able to correctly reproduce the nonlinear effect of past inputs on future outputs. This is not a severe limitation, due to the receding-horizon mechanism of MPC.

5.3 Performance comparison between ANN, RT, and RF

In this section we provide a comparison with the methodology introduced in (Smarra et al., 2018b) that we recalled in Section 4. Using data generated through the benchmark example (14), we built a model as in (8) as shown in Sections 3 and 5.1, and model (11), considering both RT and RF approaches. Validation results on the test set of 900 samples are shown in Figure 2 and Figure 3, where we compare the predicted trajectories over the horizon at $k + 1$ and $k + 10$.

Figure 4 reports e_{NRMSE} over the horizon for the three approaches. It is apparent that on the considered benchmark problem ANN and RF performance are overall quite close, and both outperform RT.

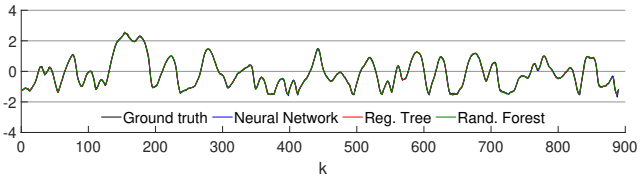

Fig. 2. Output prediction at $k + 1$

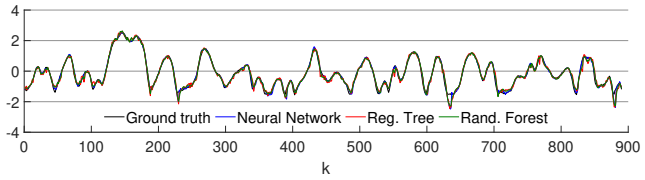

Fig. 3. Output prediction at $k + 10$

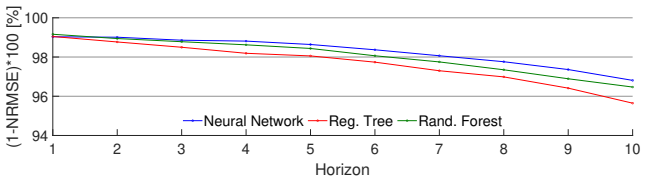

Fig. 4. Normalized root mean square error (NRMSE) over the prediction horizon

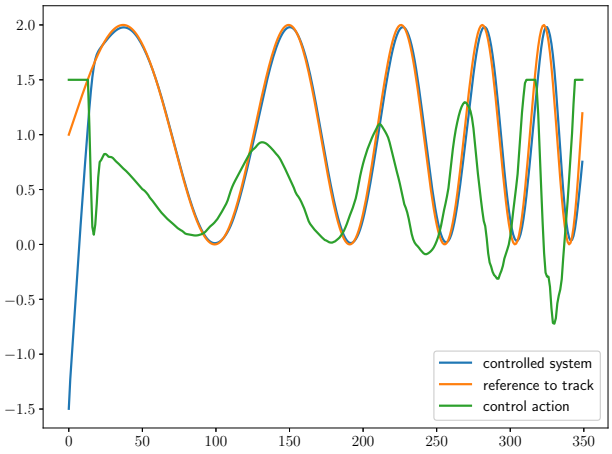

Fig. 5. Illustrative example of the performance of the LTV MPC for system (14) using ANN-based affine models via dynamic parametrization. x axis is for time steps, y is for the magnitude of the signals. All signals are in normalized units

5.4 Evaluating MPC closed-loop performance

We explore now the effect of using the learned affine predictors in a LTV-MPC control law in closed loop with (14). The LTV-MPC controller solves the QP problem (5) at each step, with H_T affine in the inputs as defined in (7). We set $W_Q = 1$, $W_R = W_S = 0.01$, $U^R_{T-1} = 0$ and impose the constraint $-1.5 \leq u_{k+j} \leq 1.5, \forall j = 0, \ldots, T-1$ with $T = 5$. Quantities are in normalized units.

The resulting performance of the controller in tracking a unit step superimposed to a sine sweep signal is reported in Figure 5. Despite predictions are only approximate, the controller provides satisfactory closed-loop performance. Regarding computation time, the evaluation of the affine predictors G_T and F_T and solving the QP problem (carried out by the general purpose solver based on (Kraft, 1988)) required $\leq$ 0.05 seconds on the same reference machine. Since the constructed LTV-MPC problem maps into a box-constrained least-squares problem, the efficient solution method proposed in (Saraf and Bemporad, 2019) could be used here for example to speed up computations even further.

6. COMPLEXITY REDUCTION

In the example presented in the Section 5.2, the ANN has ≈ 9000 parameters. Evaluating the network over the whole test set on the same machine used for training takes less than a second in total for the whole prediction horizon. For comparison, we fitted a nonlinear autoregressive model with exogenous inputs (NLARX) with comparable prediction performance, composed by two hidden layers NNs (each one composed of 20 neurons), by using MATLAB System Identification Toolbox (Ljung, 2001; Beale et al., 2018). Such NLARX model requires less than $\frac{1}{10}$ of the coefficients of the affine ANN predictor. This is not surprising, as we fit an entire horizon of predictions rather than a recursive model, and because neither in the design of the ANNs nor in the training process we took any action aimed at reducing the number of network parameters. Although storage requirement is already quite small in our approach, we discus next how to reduce memory occupancy of the ANN predictors, and therefore of the resulting MPC setup.

6.1 Memory occupancy vs. quality of fit tradeoff

It is well known in the literature that the addition of $\mathcal{L}_1$-penalties (a.k.a. the *shrinkage operator*) in an optimization problem induces sparse solutions (Tibshirani, 1996). When the optimization problem arises from fitting a model, $\mathcal{L}_1$-penalties also reduces possible overfitting issues.

Accordingly, to reduce the memory occupancy of the resulting ANNs, we modify the cost function in problem (9) as follows:

$$\underset{\{f_j, g_j\}_{k=1}^{T}}{\arg\min} \sum_{k=\max\{\delta_y, \delta_u\}}^{N-T} L(\hat{O}_k, O_k) + \lambda L_1(\Theta) \qquad (17)$$

where Θ is the overall set of non-bias weights $\theta^f_1, \theta^g_1, \ldots, \theta^f_T, \theta^g_T$ characterizing the ANNs associated with the predictors f_j and g_j, $j = 1, \ldots, T$, respectively, and

$$L_1(\Theta) = \sum_{j=1}^{T} \|\theta^f_j\|_1 + \|\theta^g_j\|_1 \qquad (18)$$

The scalar hyper-parameter $\lambda \geq 0$, decides the tradeoff between quality of fit and number of nonzero weights. Table 2 reports a realization of quality of fit, computed over all $T = 10$ steps, and of number of coefficients with absolute value $\geq 10^{-3}$ for different choices of λ. In this test all the remaining coefficients not satisfying such condition were manually set to zero after the training process for all but the $\lambda = 0$ case.

λ	e_{FIT}	NZ weights
0.01	0.853	325
0.005	0.864	350
0.001	0.901	560
0.0005	0.911	902
0	0.917	9001

Table 2. Illustrative example of the number of nonzero (NZ) weights and prediction fit obtained for different choices of λ.

7. CONCLUSIONS

In this work we developed an approach for learning an affine parameterization of output predictions from data via artificial

neural networks, conceived for efficiently formulating and solving LTV-MPC problems for nonlinear systems. We showed in numerical simulations that good performance, both in terms of capturing the dynamics of a nonlinear process and of closed-loop behavior, is achieved, with light computational load. We also showed that memory occupancy of the solution can be traded off with prediction accuracy by simply introducing $\mathcal{L}_1$-penalties during the training phase.

Experimental tests on embedded platforms on real application use cases will be performed in future work.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

Afram, A., Janabi-Sharifi, F., Fung, A. S., Raahemifar, K., 2017. Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system. Energy and Buildings 141, 96–113.

Barron, A. R., May 1993. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory 39 (3), 930–945.

Beale, M. H., Hagan, M. T., Demuth, H. B., 2018. Deep Learning Toolbox – User's Guide. The Mathworks, Inc.

Behl, M., Smarra, F., Mangharam, R., 2016. Dr-advisor: A data-driven demand response recommender system. Applied Energy 170, 30–46.

Borrelli, F., Bemporad, A., Morari, M., 2017. Predictive Control for Linear and Hybrid Systems. Cambridge University Press.

Breiman, L., 2001. Random forests. Machine learning 45 (1), 5–32.

Breiman, L., Friedman, J., Stone, C. J., Olshen, R. A., 1984. Classification and regression trees. CRC press.

Bünning, F., Huber, B., Heer, P., Aboudonia, A., Lygeros, J., 2020. Experimental demonstration of data predictive control for energy optimization and thermal comfort in buildings. Energy and Buildings 211, 109792.

Chollet, F., et al., 2015. Keras. `https://keras.io`.

Diehl, M., Bock, H., Schlöder, J., 2005. A real-time iteration scheme for nonlinear optimization in optimal feedback control. SIAM Journal on Control and Optimization 43 (5), 1714–1736.

Ferrarotti, L., Bemporad, A., 2019. Synthesis of optimal feedback controllers from data via stochastic gradient descent. In: Proc. of European Control Conference. Naples, Italy.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.

Gros, S., Zanon, M., 2020. Data-driven economic NMPC using reinforcement learning. IEEE Transactions on Automatic Control 65 (2), 636–648.

Jain, A., Smarra, F., Behl, M., Mangharam, R., 2018. Data-driven model predictive control with regression trees-An application to building energy management. ACM Transactions on Cyber-Physical Systems 2 (1), 4.

Jain, A., Smarra, F., Mangharam, R., 2017. Data predictive control using regression trees and ensemble learning. In: Decision and Control (CDC), 2017 IEEE 56th Annual Conference on. IEEE, pp. 4446–4451.

Kocijan, J., Girard, A., Banko, B., Murray-Smith, R., 2005. Dynamic systems identification with gaussian processes. Mathematical and Computer Modelling of Dynamical Systems 11 (4), 411–424.

Kraft, D., 1988. A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt.

Liu, G., Kadirkamanathan, V., 1998. Predictive control for nonlinear systems using neural networks. International Journal of Control 71 (6), 1119–1132.

Ljung, L., 1987. System identification: theory for the user. Prentice-Hall.

Ljung, L., 2001. System Identification Toolbox for MATLAB – User's Guide. The Mathworks, Inc.

Masti, D., Bemporad, A., 2018. Learning nonlinear state-space models using deep autoencoders. In: Proc. 57th IEEE Conf. on Decision and Control. Miami Beach, FL, USA, pp. 3862–3867.

Nair, V., Hinton, G. E., 2010. Rectified linear units improve restricted boltzmann machines. In: Int. Conf. on Machine Learning (ICML). Omnipress, pp. 807–814.

Piga, D., Formentin, S., Bemporad, A., 2017. Direct data-driven control of constrained systems. IEEE Transactions on Control Systems Technology 26 (4), 1422–1429.

Pillonetto, G., Dinuzzo, F., Chen, T., Nicolao, G. D., Ljung, L., 2014. Kernel methods in system identification, machine learning and function estimation: A survey. Automatica 50 (3), 657 – 682.

Saraf, N., Bemporad, A., 2019. A bounded-variable least-squares solver based on stable QR updates. IEEE Transactions on Automatic Control.

Sashank J. Reddi, Satyen Kale, S. K., 2018. On the convergence of Adam and beyond. Int. Conf. on Learning Representations.

Schoukens, M., Noël, J. P., 2017. Three benchmarks addressing open challenges in nonlinear system identification. IFAC-PapersOnLine 50 (1), 446–451.

Seide, F., Agarwal, A., 2016. CNTK: Microsoft's open-source deep-learning toolkit. In: 22nd ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining. KDD '16. ACM, New York, NY, USA, pp. 2135–2135.

Smarra, F., Di Girolamo, G. D., De Iuliis, V., Jain, A., Mangharam, R., D'Innocenzo, A., 2020. Data-driven switching modeling for mpc using regression trees and random forests. Nonlinear Analysis: Hybrid Systems 36.

Smarra, F., Jain, A., de Rubeis, T., Ambrosini, D., D'Innocenzo, A., Mangharam, R., 2018a. Data-driven model predictive control using random forests for building energy optimization and climate control. Applied Energy 226.

Smarra, F., Jain, A., Mangharam, R., D'Innocenzo, A., 2018b. Data-driven switched affine modeling for model predictive control. In: IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'18). IFAC, pp. 199–204.

Suykens, J. A., 2017. Deep restricted kernel machines using conjugate feature duality. Neural computation 29 (8), 2123–2163.

Terzi, E., Fagiano, L., Farina, M., Scattolini, R., 2019. Learning-based predictive control for linear systems: A unitary approach. Automatica 108, 108473.

Tibshirani, R., 1996. Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological) 58 (1), 267–288.

Wang, Y., 2017. A new concept using LSTM neural networks for dynamic system identification. In: 2017 American Control Conference (ACC). IEEE, pp. 5324–5329.