Recurrent Neural Network Training with Convex Loss and Regularization Functions by Extended Kalman Filtering

Alberto Bemporad, *Fellow, IEEE*

***Abstract*— This article investigates the use of extended Kalman filtering to train recurrent neural networks with rather general convex loss functions and regularization terms on the network parameters, including ℓ_1-regularization. We show that the learning method is competitive with respect to stochastic gradient descent in a nonlinear system identification benchmark and in training a linear system with binary outputs. We also explore the use of the algorithm in data-driven nonlinear model predictive control and its relation with disturbance models for offset-free closed-loop tracking.**

***Index Terms*— Recurrent neural networks, nonlinear system identification, extended Kalman filtering, nonlinear model predictive control.**

I. Introduction

The use of artificial neural networks (NNs) for control-oriented modeling of dynamical systems, already popular in the nineties [1], is again flourishing thanks to the vast success of machine learning in many application domains and the availability of excellent software libraries for training NN models. Most frequently, *feedforward* NNs are used for modeling the output function in a neural-network autoregressive model with exogenous inputs (NNARX). On the other hand, *recurrent* neural networks (RNNs), as they are state-space models, are often more adequate for capturing the behavior of dynamical systems in a compact way. However, contrarily to training NNARX models based on minimizing the one-step-ahead output prediction error, training RNNs is more difficult due to the presence of the hidden states of the network. To circumvent this issue, procedures for learning neural state-space models were proposed in [2], [3], based on the idea of finding a (reduced-order) state-space realization of a NNARX model during training and considering the values of a (thin) inner layer of the model as the state vector.

To avoid unrolling the open-loop prediction of an RNN over the entire training dataset, suboptimal approaches were proposed, such as truncated backpropagation through time (truncated BPTT) [4]. Motivated by the fact that a similar issue of recurrence occurs in minimizing the *simulation* error when training NNARX models, the authors in [5] proposed to learn RNNs based on splitting the dataset into smaller batches and penalizing the inconsistency between state predictions across consecutive batches. Due to the difficulty of computing gradients of full unrolls and/or the presence of a large number of optimization variables, these approaches are used for offline learning and rely on (batch, mini-batch, or stochastic) gradient descent methods that, although often proved to converge only for convex problems, are widely adopted with success.

To circumvent their very slow convergence and, at the same time, be able to learn NN models incrementally when new data become available, training methods based on extended Kalman filtering (EKF) have been explored in [6] for *feedforward* networks, treating the weight/bias terms of the network as constant states to estimate. When dealing with *recurrent* networks, the authors in [7] distinguish between *parallel*-EKF, which estimates both the hidden states and the weights/bias terms, and *parameter-based* EKF, that only estimates the network parameters, and focus on the latter approach, whose convergence properties were investigated in [8].

Parallel-EKF was also studied in [9] for coinciding output and state vectors, which is, therefore, measurable as for NNARX models. In the context of nonlinear filtering, the authors in [10] investigated a parallel-EKF approach for the special case of RNNs that are linear in the input and the output. The method was extended in [11] to make the EKF implementation more efficient by considering how much each network parameter affects the predicted output. An ensemble Kalman filter method was proposed in [12]. An important aspect of EKF-based learning methods is that, as remarked in [13], the filter approximately provides the minimum variance estimate of the model parameters, and hence a quantification of model uncertainty via the covariance matrix associated with the state estimation error. All the aforementioned EKF-based methods consider quadratic penalties on the output prediction errors and the weight/bias terms of the RNN.

In this article, we also consider a parallel-EKF approach to recursively learn a general class of RNNs whose state-update and output functions are described by neural networks, including long-short term memory (LSTM) models [14], under arbitrary convex and twice-differentiable loss functions for penalizing output open-loop prediction errors and for regularizing the weight/bias terms of the RNN, and also extend to the case of ℓ_1-regularization for network topology selection. We also compare EKF to different formulations based on stochastic gradient descent (SGD) methods, in which the initial state, and possibly also the intermediate states, of the RNN are treated as optimization variables. We show the superiority of EKF with respect to SGD in a nonlinear identification benchmark, and apply the method in identifying a linear dynamical system with binary outputs. We also explore the use of EKF-based learning for data-driven nonlinear MPC design, showing a relation between the use of constant disturbance models and the adaptation of the bias terms in the neural networks for offset-free tracking, which we illustrate in a nonlinear control benchmark problem.

A. Notation

Given a vector $v \in \mathbb{R}^n$ and a matrix $A \in \mathbb{R}^{m\times n}$, v_i denotes the ith real-valued component of v, $A_{i,:}$ the ith row of A, $A_{:,j}$ its jth column, A_{ij} its (i,j)th entry. Given $v \in \mathbb{R}^n$ and a symmetric positive semidefinite matrix $Q = Q' \succeq 0$, $Q \in \mathbb{R}^{n\times n}$, we denote by $\|v\|_Q^2$ the quadratic form $v'Qv$, by $\|v\|_1 = \sum_{i=1}^n |v_i|$ the 1-norm of v, and by $\text{sign}(v)$ the vector whose ith component is the sign of v_i, $\text{sign} : \mathbb{R} \to \{-1, 0, 1\}$. Given $a, b \in \mathbb{N}$, $\delta_{a,b}$ denotes the Kronecker delta function ($\delta_{a,b} = 1$ if $a = b$ or 0 otherwise).

II. Recurrent neural network model

Consider a recurrent neural network (RNN) model with input $u \in \mathbb{R}^{n_u}$, predicted output $\hat{y} \in \mathbb{R}^{n_y}$, and state vector $x \in \mathbb{R}^{n_x}$ whose

The author is with the IMT School for Advanced Studies, Piazza San Francesco 19, Lucca, Italy. Email: alberto.bemporad@imtlucca.it

This work was partially supported by the Italian Ministry of University and Research under the PRIN'17 project "Data-driven learning of constrained control systems", contract no. 2017J89ARP.

state-update and output equations are described by the following parametric model

$$\begin{aligned} x(k+1) &= f_x(x(k),u(k),\theta_x) \\ \hat{y}(k) &= f_y(x(k),u(k),\theta_y) \end{aligned} \tag{1}$$

In (1), $\theta_x \in \mathbb{R}^{n_{\theta_x}}$ and $\theta_y \in \mathbb{R}^{n_{\theta_y}}$ collect the parameters of the model to learn from data. Special cases of (1) are recurrent (deep) neural networks (RNNs)

$$\left\{\begin{aligned} v_1^x(k) &= A_1^x \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} + b_1^x \\ v_2^x(k) &= A_2^x f_1^x(v_1^x(k)) + b_2^x \\ &\vdots \quad\quad \vdots \\ v_{L_x}^x(k) &= A_{L_x} f_{L_x-1}^x(v_{L_x-1}^x(k)) + b_{L_x}^x \\ x(k+1) &= v_{L_x}^x(k) \end{aligned}\right. \tag{2a}$$

$$\left\{\begin{aligned} v_1^y(k) &= A_1^y \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} + b_1^y \\ v_2^y(k) &= A_2^y f_1^y(v_1^y(k)) + b_2^y \\ &\vdots \quad\quad \vdots \\ v_{L_y}^y(k) &= A_{L_y}^y f_{L_y-1}^y(v_{L_y-1}^y(k)) + b_{L_y}^y \\ \hat{y}(k) &= f_{L_y}^y(v_{L_y}^y(k)) \end{aligned}\right. \tag{2b}$$

where $L_x - 1 \geq 0$ and $L_y - 1 \geq 0$ are the number of hidden layers of the state-update and output functions, respectively, $v_i^x \in \mathbb{R}^{n_i^x}$, $i = 1,\dots L_x - 1$, $v_{L_x}^x \in \mathbb{R}^{n_x}$, and $v_i^y \in \mathbb{R}^{n_i^y}$, $i = 1,\dots L_y$, the values of the corresponding inner layers, $f_i^x : \mathbb{R}^{n_i^x} \to \mathbb{R}^{n_{i+1}^x}$, $i = 1,\dots L_x - 1$, $f_i^y : \mathbb{R}^{n_i^y} \to \mathbb{R}^{n_{i+1}^y}$, $i = 1,\dots L_y - 1$ the corresponding activation functions, and $f_{L_y}^y : \mathbb{R}^{n_{L_y}^y} \to \mathbb{R}^{n_y}$ the output function, e.g., $f_{L_y}^y(v_{L_y}^y) = v_{L_y}^y$ to model numerical outputs or $[f_{L_y}^y(v_{L_y}^y)]_i = \left(1 + e^{[v_{L_y}^y]_i}\right)^{-1}$, $i = 1,\dots,n_y$, for binary outputs. The strict causality of the RNN (2) can be simply imposed by zeroing the last n_u columns of A_1^y.

The hyperparameters describing the RNN (2) are n_x, $\{n_i^x\}$, $\{n_y^y\}$, $\{f_i^x\}$, $\{f_i^y\}$ and dictate the chosen model structure. The parameters to learn are A_1^x, …, $A_{L_x-1}^x$, b_1^x, …, $b_{L_x-1}^x$, A_1^y, …, $A_{L_y-1}^y$, b_1^y, …, $b_{L_y-1}^y$, where $A_i^x \in \mathbb{R}^{n_i^x \times n_{i-1}^x}$ is the matrix of weights for layer $\#i$ of the state-update neural function, $b_i^x \in \mathbb{R}^{n_i^x}$ the corresponding vector of bias terms, and $n_0^x \triangleq n_x + n_u$, $n_{L_x}^x \triangleq n_x$, and $A_i^y \in \mathbb{R}^{n_i^y \times n_{i-1}^y}$ is the matrix of weights for layer $\#i$ of the output neural function, $b_i^x \in \mathbb{R}^{n_i^x}$ the corresponding vector of bias terms, and $n_0^y \triangleq n_x + n_u$. In this case, θ_x and θ_y are the vectors obtained by stacking all the entries of the weight/bias terms defining the state and output equations, respectively, with $n_{\theta_x} = \sum_{i=1}^{L_x} n_i^x(n_{i-1}^x + 1)$ and $n_{\theta_y} = \sum_{i=1}^{L_y} n_i^y(n_{i-1}^y + 1)$.

We remark that all the results reported in this article also apply to *feedforward* neural networks, a special case of (2) for $n_x = 0$ and $n_{\theta_x} = 0$. Moreover, they can be clearly applied also to identify linear models, that is another special case of (2) when $L_x = L_y = 1$, $b_1^x = 0$, $b_1^y = 0$, as previously shown in [15]. Another popular instance of model (1) is the single-layer LSTM model [14], which we will consider in the form proposed in [16] with output equation as in (2b).

III. RNN TRAINING PROBLEM

Consider the following RNN learning problem: Given a training dataset $D_N \triangleq \{u(0), y(0), \dots, u(N-1), y(N-1)\}$, determine an optimal solution (x_0^*, θ^*) to the following mathematical program

$$\begin{aligned} \min_{x_0,\theta} \quad & V(x_0,\theta) \triangleq r_\theta(\theta) + r_x(x_0) + \frac{1}{N}\sum_{k=0}^{N-1} \ell(y(k),\hat{y}(k)) \\ \text{s.t.} \quad & \text{model equations (1) with } x(0) = x_0 \end{aligned} \tag{3}$$

where $\theta \triangleq \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix}$, $\ell : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to \mathbb{R}$ is a loss function penalizing the dissimilarity between the training samples $y(k)$ and the predicted values $\hat{y}(k)$ generated by simulating (1) from x_0, and $r_\theta : \mathbb{R}^{n_\theta} \to \mathbb{R}$, $r_x : \mathbb{R}^{n_x} \to \mathbb{R}$ are regularization functions. In the examples reported in Section VI we will consider the (weighted) mean-square error (MSE) loss $\ell_{\text{MSE}}(y,\hat{y}) = \frac{1}{2}\|y - \hat{y}\|_{W_y}^2$ where $W_y = W_y' \succ 0$ is a weight matrix, the (modified) cross-entropy loss $\ell_{\text{CE}\epsilon}(y(k),\hat{y}) = \sum_{i=1}^{n_y} -y_i(k)\log(\epsilon + \hat{y}_i) - (1 - y_i(k))\log(1 + \epsilon - \hat{y}_i)$ for binary outputs, the Tikhonov (or ℓ_2) regularization terms $r_\theta(\theta) = \frac{\rho_\theta}{2}\|\theta\|_2^2$, $r_x(x_0) = \frac{\rho_x}{2}\|x_0\|_2^2$ with $\rho_\theta, \rho_x \geq 0$, and the ℓ_1-regularization $r_\theta(\theta) = \lambda\|\theta\|_1$.

In this article we consider learning problems based on a single training dataset D_N. The extension to multiple training datasets $D_{N_1}^1, \dots, D_{N_{n_d}}^{n_d}$, $D_{N_d}^{n_d} \triangleq \{u^d(0), y^d(0), \dots, u^d(N_d-1), y^d(N_d-1)\}$, $d = 1,\dots,n_d$, $n_d > 1$ can be simply formulated as

$$\begin{aligned} \min_{x_0^1,\dots,x_0^{n_d},\theta} \quad & r_\theta(\theta) + \sum_{d=1}^{n_d} r_x(x_0^d) + \frac{1}{N_d}\sum_{k=0}^{N_d-1} \ell(y^d(k),\hat{y}^d(k)) \\ \text{s.t.} \quad & \hat{y}^d(k), x^d(k), \text{ and } u^d(k) \text{ satisfy model (1)} \\ & x^d(0) = x_0^d, \ d = 1,\dots,n_d. \end{aligned} \tag{4}$$

A. Condensed learning

An optimizer (x_0^*, θ_0^*) of Problem (3) can be computed by different unconstrained nonlinear programming (NLP) solvers [17], including the approach recently proposed in [18]. For a given value of (x_0,θ), evaluating $V(x_0,\theta)$ requires processing an *epoch*, i.e., the entire training dataset. Since the cost function in (3) is not separable due to the dynamic constraints (1), a gradient descent method corresponds to the steepest-descent steps

$$\begin{bmatrix} x_0^{t+1} \\ \theta^{t+1} \end{bmatrix} = \begin{bmatrix} x_0^t \\ \theta^t \end{bmatrix} - \alpha_t \begin{bmatrix} \frac{\partial V}{\partial x_0}(x_0^t,\theta^t) \\ \frac{\partial V}{\partial \theta}(x_0^t,\theta^t) \end{bmatrix} \tag{5}$$

where α_t is the learning rate at epoch $t = 1,\dots,N_e$. By applying the chain rule for computing derivatives, the gradient of the objective function with respect to (x_0,θ) can be evaluated efficiently by BPTT [19]. However, gradient computation involves well-known issues; for example, in the case of vanilla RNNs *vanishing gradients* [20] and *exploding gradients* effects have been reported. While heuristic remedies to alleviate this effect, such as approximating the gradients by truncated BPTT or different architectures such as LSTMs [14] were proposed, the root-cause lies in having *condensed* the problem by eliminating the intermediate variables $x(k)$, $k = 1,\dots,N-1$.

Problem (3) can be interpreted as a finite-horizon optimal control problem with free initial state x_0 (penalized by $r_x(x_0)$), where $\theta(k) \equiv \theta$ is the vector of manipulated inputs to optimize, $r_\theta(\theta)$ the corresponding penalty, $\hat{y}(k)$ the controlled output, $y(k)$ the output reference, $u(k)$ a measured disturbance (alternatively, x_0 could be also seen as an input only acting at time -1 to change $x(0)$ from $x(-1) = 0$). As well-known in solving MPC problems, the effect of *condensing* the problem (a.k.a. "direct single shooting" [21]) by eliminating state variables potentially leads to numerical difficulties. For example in standard linear MPC formulations an unstable linear prediction model $x(k+1) = Ax(k) + Bu(k)$ leads to a quadratic

program with ill-conditioned Hessian matrix, due to the presence of the terms A^k that appear in constructing the program (cf., [22]). The reader is also referred to the recent work [23] for a re-interpretation of the training problem of feedforward neural networks as an optimal control problem and the benefits of avoiding condensing from a solution-algorithm perspective.

B. Relaxed non-condensed learning

By following the analogy with MPC, the problem in (3) can be solved in *non-condensed* form (a.k.a. "direct multiple shooting" [24]) by also treating $x(1),\dots,x(N-1)$ (and possibly also $v_i^x(k)$, $v_j^y(k)$) as additional optimization variables, subject to the equality constraints imposed by the RNN model (2) or, more generally, (1). By further relaxing the model equations we get the following unconstrained nonlinear optimization problem

$$\begin{aligned}\min_{\substack{x_0,\,\theta\\ x_1,\dots,x_{N-1}}}\; & r_\theta(\theta)+r_x(x_0)+\frac{1}{N}\sum_{k=0}^{N-1}\ell(y(k),f_y(x_k,u(k),\theta_y))\\ &+\frac{\gamma}{2N}\sum_{k=0}^{N-2}\|x_{k+1}-f_x(x_k,u(k),\theta_x)\|_2^2\end{aligned}\tag{6}$$

where $\gamma>0$ is a scalar penalty promoting the consistency of the state sequence with model (1).

Note that in the case of MSE loss and ℓ_2-regularization, (6) can be solved as a nonlinear least-squares problem, for which efficient solution methods exist [17]. The relaxation (6) is also amenable for stochastic gradient-descent iterations that only update (x_k,x_{k+1},θ) when sample k is processed:

$$\begin{aligned}\begin{bmatrix}x_{k+1}^{k+1}\\ x_{k+1}^{k+1}\\ x_k^{k+1}\\ \theta^{k+1}\end{bmatrix}&=\begin{bmatrix}x_{k+1}^k\\ x_k^k\\ \theta^k\end{bmatrix}-\alpha_k\begin{bmatrix}0\\ \frac{\partial\ell}{\partial x}(y(k),f_y(x_k^k,u(k),\theta_y^k))\\ \frac{\partial\ell}{\partial\theta}(y(k),f_y(x_k^k,u(k),\theta_y^k))\end{bmatrix}\\ &-\gamma\alpha_k\begin{bmatrix}I\\ \frac{\partial f_x}{\partial x}(x_k^k,u(k),\theta_x^k)'\\ \frac{\partial f_x}{\partial\theta}(x_k^k,u(k),\theta_x^k)'\end{bmatrix}(x_{k+1}-f_k)-\frac{\alpha_k}{N}\begin{bmatrix}0\\ \nabla r_x(x_0^k)\delta_{k,0}\\ \nabla r_\theta(\theta^k)\end{bmatrix}\end{aligned}\tag{7}$$

where $f_k\triangleq f_x(x_k^k,u(k),\theta_x^k))$, $k=0,\dots,N-1$. Clearly, (7) can be also run on multiple epochs by resetting $k=0$ at each epoch $t=1,\dots,N_e$.

Finally, we remark that in case of multiple training datasets ($n_d>1$), the number of optimization variables remains Nn_x+n_θ, where $N=\sum_{d=1}^{n_d}N_d$, and simply some regularization terms $\gamma\|x_{k+1}-f_x(x_k,u_k,\theta_x)\|_2^2$ are skipped in (6) to take into account that the initial state of a new training sequence is not related to the final predicted state of the previous sequence.

C. Relaxed partially-condensed learning

The non-condensed form (6) has $n_\theta+Nn_x$ optimization variables, where in the present context of control-oriented RNN models usually $Nn_x\gg n_\theta$. *Partial condensing* [25] can reduce the training problem as follows. Let us split the dataset D_N into M batches, $M\le N$, of lengths $L_1,\dots,L_M$, respectively (for example, $L_1=L_2=L_{M-1}=\lceil\frac{N}{M}\rceil$, $L_M=N-(M-1)\lceil\frac{N}{M}\rceil$). Then, we solve the following problem:

$$\begin{aligned}\min_{x_0,x_1,\dots,x_{M-1},\theta}\;&\frac{1}{N}\sum_{j=0}^{M-1}\sum_{h=0}^{L_j-1}\ell(y(k_{ij}),f_y(\hat{x}_{h|j},u(k_{ij}),\theta_y)\\ &+r_\theta(\theta)+r_x(x_0)+\gamma\frac{N-1}{2N(M-1)}\sum_{j=0}^{M-2}\|x_{j+1}-\hat{x}_{L_j|j}\|_2^2\end{aligned}\tag{8}$$

where $k_{ij}\triangleq h+\sum_{s=0}^{j-1}L_s$ and $\hat{x}_{h|j}$ is the hidden state predicted by iterating (1) over h steps from the initial condition x_j under the input excitation $u(L_j),\dots,u(L_j+h-1)$, $h=1,\dots,L_j$, $j=0,\dots,M-1$. Similarly to the approach of [5], problem (8) can be solved by an SGD method by processing L_j samples at the time, that results in updating (x_{j+1},x_j,θ) at each SGD iteration. Note that (8) includes (6) as a special case by setting $M=N$, $L_j\equiv 1$.

IV. Training by Extended Kalman filtering

To address the *recursive* estimation of θ from real-time streams of input and output measurements, and to counteract the slow convergence of SGD methods, we consider the use of extended Kalman filtering (EKF) techniques as in [10], [11] to recursively update θ and the current hidden state vector. To this end, let us rewrite model (1) as the following nonlinear system affected by noise

$$\begin{aligned}x(k+1)&=f_x(x(k),u(k),\theta_x(k))+\xi(k)\\ y(k)&=f_y(x(k),u(k),\theta_y(k))+\zeta(k)\\ \theta(k+1)&=\theta(k)+\eta(k),\quad \theta(k)\triangleq\begin{bmatrix}\theta_x(k)\\ \theta_y(k)\end{bmatrix}\end{aligned}\tag{9}$$

where $\xi(k)\in\mathbb{R}^{n_x}$, $\zeta(k)\in\mathbb{R}^n_y$, and $\eta(k)\in\mathbb{R}^{n_\theta}$ are white, zero-mean, noise vectors with covariance matrices $Q_x(k)$, $Q_y(k)$, and $Q_\theta(k)$, respectively, with $Q_x(k)=Q_x(k)'\succeq 0$, $Q_x(k)\in\mathbb{R}^{n_x\times n_x}$, $Q_y(k)=Q_y(k)'\succ 0$, $Q_y(k)\in\mathbb{R}^{n_y\times n_y}$, $Q_\theta(k)=Q_\theta(k)'\succeq 0$, $Q_\theta(k)\in\mathbb{R}^{n_\theta\times n_\theta}$ for all k.

The model coefficient vector $\hat{\theta}(k)$ and the hidden state $\hat{x}(k)$ are estimated with data up to time k according to the following classical recursive EKF updates:

$$\begin{aligned}C(k)&=\left[\frac{\partial f_y}{\partial x}\quad 0\quad \frac{\partial f_y}{\partial\theta_y}\right]\Big|_{\hat{\theta}(k|k-1),\hat{x}(k|k-1),u(k)}\\ M(k)&=P(k|k-1)C(k)'[C(k)P(k|k-1)C(k)'\\ &\quad+Q_y(k)]^{-1}\\ e(k)&=y(k)-f_y(\hat{x}(k|k-1),u(k),\hat{\theta}_y(k|k-1))\\ \begin{bmatrix}\hat{x}(k|k)\\ \hat{\theta}(k|k)\end{bmatrix}&=\begin{bmatrix}\hat{x}(k|k-1)\\ \hat{\theta}(k|k-1)\end{bmatrix}+M(k)e(k)\\ P(k|k)&=(I-M(k)C(k))P(k|k-1)\end{aligned}\tag{10a}$$

$$\begin{aligned}\begin{bmatrix}\hat{x}(k+1|k)\\ \hat{\theta}(k+1|k)\end{bmatrix}&=\begin{bmatrix}f_x(\hat{x}(k|k),u(k),\hat{\theta}_x(k|k))\\ \hat{\theta}(k|k)\end{bmatrix}\\ A(k)&=\begin{bmatrix}\frac{\partial f_x}{\partial x}&\frac{\partial f_x}{\partial\theta_x}&0\\ 0&I&0\\ 0&0&I\end{bmatrix}\Big|_{\hat{\theta}(k|k),\hat{x}(k|k),u(k)}\\ P(k+1|k)&=A(k)P(k|k)A(k)'+\begin{bmatrix}Q_x(k)&0\\ 0&Q_\theta(k)\end{bmatrix}.\end{aligned}\tag{10b}$$

Note that in the single output case ($n_y=1$), the matrix inversion in (10a) becomes a simple division.

Let us recall the equivalence between the EKF (10) and Newton's method [26, Sect. 5.2] for solving (6) in the case of loss $\ell_{\rm MSE}$ and

ℓ_2-regularization, and of an additional penalty on updating θ:

$$\begin{aligned}\min_{\substack{x_0,x_1,\dots,x_{N-1}\\ \theta_0,\theta_1,\dots,\theta_{N-1}}} \quad & \frac{1}{2}\left\|\begin{bmatrix}x_0\\ \theta\end{bmatrix}-\begin{bmatrix}x(0|-1)\\ \theta(0|-1)\end{bmatrix}\right\|^2_{P(0|-1)^{-1}}\\ &+\frac{1}{2}\sum_{k=0}^{N-1}\|y(k)-f_y(x_k,u(k),\theta_{yk})\|^2_{Q_y^{-1}(k)}\\ &+\frac{1}{2}\sum_{k=1}^{N-2}\|x_{k+1}-f_x(x_k,u(k),\theta_{xk})\|^2_{Q_x^{-1}(k)}\\ &+\|\theta_{k+1}-\theta_k\|^2_{Q_\theta^{-1}(k)}.\end{aligned} \tag{11}$$

By dividing the cost function in (11) by N, we have that $Q_x=\frac{1}{\gamma}I$, $Q_y=W_y^{-1}$, while $Q_\theta(k)=\frac{1}{\alpha_k}I$ induces a learning-rate due to minimizing the additional term $\frac{\alpha_k}{2}\|\theta_{k+1}-\theta_k\|_2^2$. The initial vector $\begin{bmatrix}x(0)\\ \theta(0)\end{bmatrix}$ is treated as a random vector with mean $\begin{bmatrix}x(0|-1)\\ \theta(0|-1)\end{bmatrix}$ and covariance $P(0|-1)=P(0|-1)'\succeq 0$, $P(0|-1)\in\mathbb{R}^{(n_x+n_\theta)\times(n_x+n_\theta)}$. By the aforementioned analogy between EKF and Newton's method, we have that

$$P(0|-1)=\begin{bmatrix}\frac{1}{N\rho_x}I & 0\\ 0 & \frac{1}{N\rho_\theta}I\end{bmatrix} \tag{12}$$

is directly related to the ℓ_2-regularization terms $r_\theta(\theta)=\frac{\rho_\theta}{2}\|\theta\|_2^2$, $r_x(x_0)=\frac{\rho_x}{2}\|x_0\|_2^2$ used in (6) when $x(0|-1)=0$, $\theta(0|-1)=0$. We remark that the EKF iterations (10) are not guaranteed to converge to a global minimum of the posed training problem. The reader is referred to the EKF convergence results reported in [15], [27] for further details.

In the next section, we show how to extend EKF-based training to handle generic strongly convex and twice differentiable loss functions ℓ and regularization terms r_θ, r_x.

A. EKF with general output prediction loss

Lemma 1: Let $\ell:\mathbb{R}^{n_y}\times\mathbb{R}^{n_y}\to\mathbb{R}$ be strongly convex and twice differentiable with respect to its second argument $\hat{y}$. Then by setting

$$Q_y(k) \triangleq \left(\frac{\partial^2\ell(y(k),\hat{y}(k|k-1))}{\partial\hat{y}^2}\right)^{-1} \tag{13a}$$

$$e(k) = -Q_y(k)\frac{\partial\ell(y(k),\hat{y}(k|k-1))}{\partial\hat{y}} \tag{13b}$$

the EKF updates (10) attempt at minimizing the loss ℓ.

Proof: At each step k, let us take a second-order Taylor expansion of $\ell(y(k),\hat{y})$ around $\hat{y}(k|k-1)$ to approximate

$$\begin{aligned}\arg\min \ell(y(k),\hat{y}) &\approx \arg\min\{\tfrac{1}{2}\Delta y_k' Q_y^{-1}(k)\Delta y_k\\ &\quad +\varphi_k'\Delta y_k\}\\ &= \arg\min\{\tfrac{1}{2}\|\hat{y}(k|k-1)-Q_y(k)\varphi_k-\hat{y}\|^2_{Q_y^{-1}(k)}\}\end{aligned} \tag{14}$$

where $\Delta y_k=\hat{y}-\hat{y}(k|k-1)$ and $\varphi_k\triangleq\frac{\partial\ell(y(k),\hat{y}(k|k-1))}{\partial\hat{y}}$.

Due to the parallel between EKF and Newton's method recalled in (11), feeding the measured output $y(k)=\hat{y}(k|k-1)-Q_y(k)\varphi_k$ gives $e(k)$ and $Q_y(k)$ as in (13). ■

Remark 1: Note that for the MSE loss $\ell_{\mathrm{MSE}}(y(k),\hat{y})=\frac{1}{2}\|y(k)-\hat{y}\|^2_{W_y}$, as $Q_y(k)=W_y^{-1}$, we get $\varphi_k=Q_y^{-1}(k)(\hat{y}(k|k-1)-y(k))$ and hence from (13b) the classical output prediction error term $e(k)=y(k)-\hat{y}(k|k-1)$. ■

Remark 2: When the modified cross-entropy loss $\ell_{\mathrm{CE}\epsilon}(y(k),\hat{y})$ is used to handle binary outputs $y(k)\in\{0,1\}$ and predictors $\hat{y}(k|k-1)\in[0,1]$ we get $\varphi_k=-\frac{y(k)}{\epsilon+\hat{y}(k|k-1)}+\frac{1-y(k)}{1+\epsilon-\hat{y}(k|k-1)}$ and

$$Q_y(k)=\left(\frac{y(k)}{(\epsilon+\hat{y}(k|k-1))^2}+\frac{1-y(k)}{(1+\epsilon-\hat{y}(k|k-1))^2}\right)^{-1}. \tag{15a}$$

Hence, from (13b), we get $e(k)=-1-\epsilon-\hat{y}(k|k-1)$ for $y(k)=0$ and $e(k)=\epsilon+\hat{y}(k|k-1)$ for $y(k)=1$, or equivalently

$$e(k)=(1+2\epsilon)y(k)+\hat{y}(k|k-1)-1-\epsilon. \tag{15b}$$

Note that $\epsilon>0$ is used to avoid possible numerical issues in computing $Q_y^{-1}(k)$ when $\hat{y}(k|k-1)$ tends to 0 or 1. ■

B. EKF with generic regularization terms

We have seen in (12) how a quadratic regularization r_θ can be embedded in the EKF formulation by properly defining $P(0|-1)$. We now extend the formulation to handle more general regularization terms.

Lemma 2: Consider the generic regularization term

$$r_\theta(\theta)=\Psi(\theta)+\frac{1}{2}\rho_\theta\|\theta\|_2^2$$

and let $\Psi:\mathbb{R}^{n_\theta}\to\mathbb{R}$ be strongly convex and twice differentiable. Then by extending model (9) with the additional system output

$$y_\Psi(k)=\theta(k)+\mu_\Psi(k) \tag{16}$$

where $\mu_\Psi(k)$ has zero mean and covariance

$$Q_\Psi(k)=\left(\nabla^2_\theta\Psi(\theta(k|k-1))\right)^{-1} \tag{17a}$$

and by feeding the error term

$$e_\Psi(k)=-Q_\Psi(k)\nabla_\theta\Psi(\theta(k|k-1)) \tag{17b}$$

the EKF updates (10) attempt at minimizing $r_\theta(\theta)$.

Proof: Since $\Psi(\theta)=\frac{1}{N}\sum_{k=0}^{N-1}\Psi(\theta)$, at a given prediction step k consider the further loss term $\Psi(\theta)$ and its approximate minimization around $\theta(k|k-1)$

$$\begin{aligned}\arg\min\Psi(\theta) &\approx \arg\min\left\{\frac{1}{2}\Delta\theta_k' Q_\Psi^{-1}\Delta\theta_k+\gamma_k'\Delta\theta_k\right\}\\ &= \arg\min\left\{\frac{1}{2}\|\Delta\theta_k+Q_\Psi(k)\gamma_k\|^2_{Q_\Psi^{-1}(k)}\right\}\end{aligned}$$

where $\Delta\theta_k\triangleq\theta-\theta(k|k-1)$ and $\gamma_k\triangleq\nabla_\theta\Psi(\theta(k|k-1))$. By feeding the measurement $y_\Psi(k)=\theta(k|k-1)-Q_\Psi^{-1}(k)\gamma_k$ to the EKF, as in (13) we get the additional error term $e_\Psi(k)=-Q_\Psi(k)\gamma_k$ and the corresponding output Jacobian matrix $C_\Psi(k)=[0\ I]$. ■

Note that for $\Psi(\theta)=\frac{1}{2}\bar{\rho}_\theta\sum_{i=1}^{n_\theta}\theta_i^2$, then $Q_\Psi(k)=\frac{1}{\bar{\rho}_\theta}I$ and $e_\Psi(k)=-\hat{\theta}(k|k-1)$. Next Lemma 3 specializes the result of Lemma 2 to the case of separable regularization functions.

Lemma 3: Let $\Psi(\theta)=\sum_{i=1}^{n_\theta}\psi_i(\theta_i)$ and assume that each function $\psi_i:\mathbb{R}\to\mathbb{R}$ is strongly convex and twice differentiable. Then, after each measurement update k and before performing the time update in (10), minimizing the additional loss Ψ corresponds to 1) setting

$$\begin{aligned}\begin{bmatrix}\hat{x}(k_0)\\ \hat{\theta}(k_0)\end{bmatrix} &= \begin{bmatrix}\hat{x}(k|k-1)\\ \hat{\theta}(k|k-1)\end{bmatrix}+M(k)e(k)\\ P(k_0) &= (I-M(k)C(k))P(k|k-1)\end{aligned} \tag{18a}$$

2) iterating

$$
\begin{aligned}
C_{\Psi i}(k_i) &= I_{n_x+i,:} \\
e_{\Psi i}(k_i) &= -\frac{\psi_i'(\hat{\theta}_i(k_{i-1}))}{\psi_i''(\hat{\theta}_i(k_{i-1}))} \\
M(k_i) &= \frac{P_{:,n_x+i}(k_{i-1})}{P_{n_x+i,n_x+i}(k_{i-1}) + 1/\psi_i''(\hat{\theta}_i(k_{i-1}))} \\
\begin{bmatrix}\hat{x}(k_i)\\ \hat{\theta}(k_i)\end{bmatrix} &= \begin{bmatrix}\hat{x}(k_{i-1})\\ \hat{\theta}(k_{i-1})\end{bmatrix} + M(k_i)e_{\Psi i}(k_i) \\
P(k_i) &= P(k_{i-1}) - M(k_i)P_{n_x+i,:}(k_{i-1}) \qquad (18b)
\end{aligned}
$$

for $i = 1, \ldots, n_\theta$, and 3) assigning

$$
\begin{bmatrix}\hat{x}(k|k)\\ \hat{\theta}(k|k)\end{bmatrix} = \begin{bmatrix}\hat{x}(k_{n_\theta})\\ \hat{\theta}(k_{n_\theta})\end{bmatrix}, \quad P(k|k) = P(k_{n_\theta}). \qquad (18c)
$$

Proof: The result simply follows due to the independence of each component $\mu_{\Psi i}(k)$ in (16), $i = 1, \ldots, n_\theta$, and $\zeta(k)$ in (9), see, e.g., [28, Sect. 10.2.1]. In fact, the multi-output measurement update of the EKF can be processed sequentially, i.e., first $y(k)$ to get $\hat{x}(k_0)$, $\hat{\theta}(k_0)$, $P(k_0)$, and then the outputs $y_{\Psi i}(k)$ one by one, for $i = 1, \ldots, n_\theta$, finally getting $\hat{x}(k|k)$, $\hat{\theta}(k|k)$, and $P(k|k)$. ■

C. EKF-based training with ℓ_1-regularization

When training RNN models several degrees of freedom exist in selecting the model structure, i.e., the number of layers and of neurons in each layer of the feedforward neural networks f_x, f_y. It is common to start with a large enough number of parameters and then promote the sparsity of θ by introducing the penalty $\lambda\|\theta\|_1$ on θ. We next show how to handle such a sparsifier in our EKF-based setting.

Theorem 1: Minimizing the additional loss $\lambda\|\theta\|_1$ corresponds to the measurement updates as in (18) with (18) replaced by

$$
\begin{aligned}
\begin{bmatrix}\hat{x}(k_i)\\ \hat{\theta}(k_i)\end{bmatrix} &= \begin{bmatrix}\hat{x}(k_{i-1})\\ \hat{\theta}(k_{i-1})\end{bmatrix} - \lambda \operatorname{sign}(\hat{\theta}_i(k_{i-1}))P_{:,n_x+i}(k_{i-1}) \\
P(k_i) &= P(k_{i-1}), \quad i = 1, \ldots, n_\theta \qquad (19a)
\end{aligned}
$$

Proof: Consider the following smooth version of the 1-norm $\Psi(\theta) = \lambda \sum_{i=1}^{n_\theta} \sqrt{\theta_i^2 + \tau\theta_i^4}$, $\tau > 0$, and the associated derivatives $\psi_i'(\theta_i) = \lambda \frac{2\tau\theta_i^3 + \theta_i}{(\tau\theta_i^4 + \theta_i^2)^{\frac{1}{2}}}$, $\psi_i''(\theta_i) = \lambda \frac{\tau\theta_i^4(2\tau\theta_i^2+3)}{(\tau\theta_i^4+\theta_i^2)^{\frac{3}{2}}}$. By applying Lemma 3 we get the following updates $e_{\Psi i}(k_i) = -\frac{2\tau^2\hat{\theta}_i^4 + 3\tau\hat{\theta}_i^2 + 1}{\tau\hat{\theta}_i(2\tau\hat{\theta}_i^2+3)}$, $M(k_i) = \frac{\lambda\tau\hat{\theta}_i^4(2\tau\hat{\theta}_i^2+3)P_{:,n_x+i}}{\lambda\tau\hat{\theta}_i^4(2\tau\hat{\theta}_i^2+3)\pi_{ii} + (\tau\hat{\theta}_i^4+\hat{\theta}_i^2)^{\frac{3}{2}}}$ where for simplicity we set $\pi_{ii} \triangleq P_{n_x+i,n_x+i}$ and omitted "(k_{i-1})", and we also assumed $\theta_i \neq 0$. For $\tau \to 0$ we get $M(k_i)e_{\Psi i}(k_i) \to -\frac{\lambda\hat{\theta}_i^3}{(\hat{\theta}_i^2)^{\frac{3}{2}}}P_{:,n_x+i} = -\lambda \operatorname{sign}(\hat{\theta}_i)P_{:,n_x+i}$ and $M(k_i) \to 0$. For $\hat{\theta}_i \to 0$ we get $M(k_i)e_{\Psi i}(k_i) \to 0$ and $M(k_i) \to 0$ for all $\tau > 0$. As $M(k_i) \to 0$, we also get $P(k_i) = P(k_{i-1})$ in (18). ■

An alternative to (19a) is to evaluate all the sign terms upfront at $\hat{\theta}(k_0) = \hat{\theta}(k|k-1)$, leading to the update

$$
\begin{aligned}
\begin{bmatrix}\hat{x}(k|k)\\ \theta(k|k)\end{bmatrix} &= \begin{bmatrix}\hat{x}(k|k-1)\\ \theta(k|k-1)\end{bmatrix} + M(k)e(k) \\
&\quad -\lambda P(k|k-1)\begin{bmatrix}0\\ \operatorname{sign}(\hat{\theta}(k|k-1))\end{bmatrix} \\
P(k|k) &= (I - M(k)C(k))P(k|k-1) \qquad (19b)
\end{aligned}
$$

D. Complexity of EKF-based training

We briefly discuss the numerical complexity of the proposed EKF-based training method. The iterations (10) require the following steps: forming matrix $C(k)$ requires evaluating $n_y(n_x + n_{\theta_y})$ partial derivatives; evaluating $M(k)$ requires computing $D_1(k) \triangleq P(k|k-1)C(k)'$, $D_2(k) \triangleq C(k)D_1(k)$, the $n_y \times n_y$ inverse symmetric matrix $D_3(k) \triangleq [Q_y(k) + D_2(k)]^{-1}$, and then evaluate $M(k) = D_1(k)D_3(k)$, which has an overall complexity $O(n_y(n_x+n_\theta)^2 + (n_x+n_\theta)n_y^2 + n_y^3)$; note that no matrix inversion is required in (10) in the single-output case ($n_y = 1$), as it becomes a simple division. Evaluating $P(k|k) = P(k|k-1) - M(k)D_1(k)'$ has complexity $O(n_y(n_x+n_\theta)^2)$; forming matrix $A(k)$ requires evaluating $n_x(n_x+n_{\theta_x})$ partial derivatives; finally, updating $P(k+1|k)$ requires $O(n_x^3 + n_x(n_{\theta_x}^2 + n_{\theta_y}^2) + n_x^2 n_\theta)$ operations. The reader is referred to [Ch. 10] [28] for different implementations of the EKF, such as in factored or square-root form.

In the multi-output case ($n_y > 1$), an alternative and slightly different formulation to avoid inverting $Q_y(k) + D_2(k)$ is to take $Q_y(k)$ diagonal and treat measurements updates one by one as in (18). The drawback of this approach is that $M(k)$ and $P(k|k)$ must be updated n_y times. Since typically $n_y \ll n_\theta$, in general this would be computationally heavier than performing the computations in (10).

When the additional ℓ_1-penalty $\|\theta\|_1$ is included, Eq. (19) requires extra $O(n_\theta(n_x+n_\theta))$ operations. When including a more general separable penalty $\Psi(\theta) = \sum_{i=1}^{n_\theta}\psi_i(\theta)$, $Q_\Psi(k)$ is an $n_\theta \times n_\theta$ diagonal matrix and Eq. (17) requires computing n_θ second derivatives $\frac{d^2\psi_i}{d\theta_i^2}$ and their reciprocals, and $e_\Psi(k)$ involves $O(n_\theta)$ operations. Then (18) has a complexity $O(n_\theta(n_\theta+n_x)^2)$.

We finally remark that, according to our numerical experience, the main computation complexity is due to computing the partial derivatives in $A(k)$ and $C(k)$ (e.g., by back-propagation), especially in the case of neural networks with a large number L_x, L_y of layers. Clearly, the computation of Jacobian matrices is required by all gradient-based training algorithms.

E. Initial-state reconstruction

Given a model θ and a dataset $\{\bar{u}(0), \bar{y}(0), \ldots, \bar{u}(\bar{N}-1), \bar{y}(\bar{N}-1)\}$, testing the prediction capabilities of the model in open-loop simulation requires determining an appropriate initial state $\bar{x}(0)$. A way to get $\bar{x}(0)$ is to solve the following state-reconstruction problem with n_x variables

$$
\min_{\bar{x}_0} r_x(x_0) + \frac{1}{\bar{N}}\sum_{k=0}^{\bar{N}-1} \ell(\bar{y}(k), \hat{y}(k)) \qquad (20)
$$

where $\hat{y}(k)$ are generated by iterating (1) from $x(0) = \bar{x}_0$, or its equivalent non-condensed or partially-condensed form.

Solving (20) is not only useful to test a trained model on new data, but also when running the EKF (10) offline on training data over multiple epochs. In this case, Problem (20) can provide a suitable value for $x(0|-1)$ for the new epoch based on the last vector $\hat{\theta}(N-1|N-1)$ learned, that is used in (20) and as the initial condition $\theta(0|-1)$ for the new epoch. We also set $P(0|-1)$ equal to the value $P(N|N-1)$ from the previous epoch, although alternative re-initialization methods might be envisioned.

In the case of multiple training experiments $n_d > 1$, it is enough to run the EKF on each dataset $D_{N_d}^{n_d}$ by always resetting the initial state $x(0)^d$ as in (20) based on the first $\bar{N}$ samples in $D_{N_d}^{n_d}$, while vector θ and the covariance matrix P get propagated across experiments and epochs.

V. Nonlinear model predictive control

In a fully adaptive case, the EKF as in (10) can be used online both to estimate the state and to update the parameters of the model from streaming output measurements. Then, a nonlinear MPC controller can be setup by solving a finite-time nonlinear optimal control problem over the horizon $[k, k+p]$ at each execution time k for a given prediction horizon p, taking $\hat{x}(k|k)$ as the initial state of the prediction and using $\hat{\theta}(k|k)$ to make nonlinear predictions. An example of such an adaptive nonlinear MPC scheme, that we will use in the numerical experiments reported in Section VI, is

$$\begin{aligned} \min_{u_0,\ldots,u_{p-1}} \quad & \sum_{t=0}^{p} \|W^{\Delta u}(u_t - u_{t-1})\|_2^2 + \|W^{y}(y_t - r_t)\|_2^2 \\ \text{s.t.} \quad & x_{t+1} = f_x(x_t, u_t, \hat{\theta}_x(k|k)),\ y_t = f_y(x_t, u_t, \hat{\theta}_y(k|k)) \\ & u_{\min} \le u_t \le u_{\max} \end{aligned} \tag{21}$$

where $x_0 = \hat{x}(k|k)$ and $u_{-1} = u(k-1)$. Note that the last penalty on $u_p - u_{p-1}$ and the first penalty on $y_0 - r_0$ in (21) can be omitted in case of strictly-causal RNN models.

When full adaptation is not recommended, for instance to prevent excessive changes of the model parameters and/or to avoid computing the full EKF iterations, a non-adaptive nonlinear MPC setting can be used, in which an optimal vector θ^* of parameters is obtained offline by running (10) on a training input/output dataset and used to make model-based predictions. A possible drawback of the latter approach is that offsets may arise in steady state when tracking constant set-points due to model/plant mismatches. A common practice is to augment the observer with a disturbance model, that in the current nonlinear MPC setting corresponds to augment the prediction model as follows (cf., [29]):

$$\begin{aligned} x(k+1) &= f_x(x(k), u(k), \theta_x(k)) + B_d d(k) + \xi(k) \\ y(k) &= f_y(x(k), u(k), \theta_y(k)) + C_d d(k) + \zeta(k) \\ d(k+1) &= d(k) + \eta(k) \end{aligned} \tag{22}$$

and estimate $\hat{x}(k|k)$, $\hat{d}(k|k)$ by EKF. Clearly, such a solution corresponds to only updating the bias terms b_1^x, $b_{L_y}^y$ of the RNN model (2) by setting $b_1^x(k|k) = (b_1^x)^* + B_d d(k|k)$, $b_{L_y}^y(k|k) = (b_{L_y}^y)^* + C_d d(k|k)$. The (frequently used) case of pure output disturbance models ($B_d = 0$, $C_d = I$) corresponds to only updating $b_{L_y}^y$.

VI. Numerical experiments

We test the proposed EKF-based RNN learning method on system identification and nonlinear MPC problems. All computations are done in MATLAB R2022b on an Apple M1 Max machine using CasADi [30] for automatic differentiation. We use the particle swarm optimizer PSwarm [31] to solve the low-dimensional nonlinear programming problem (20) with $\bar{N} = 100$, initial population of $2n_x$ samples, and each component of x_0 restricted in $[-3, 3]$.

Unless stated differently, we use covariance matrices[1], $Q_x(k) \equiv 10^{-10} I$, $Q_\theta(k) \equiv 10^{-10} I$, $Q_y(k) \equiv 1$, $P(0|-1)$ as in (12), initial state $x(0|-1) = 0$, $\gamma = 10^{-4}$ in (6), and Adam [32] over $N_e = 500$ epochs with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ for SGD. Standard scaling of the input and output samples is performed by computing their means and standard deviations on training data. Model quality is judged in terms of the best fit rate BFR

[1]We observed that larger values of $Q_x(k)$, $Q_\theta(k)$ only slow down convergence speed without providing any other benefit.

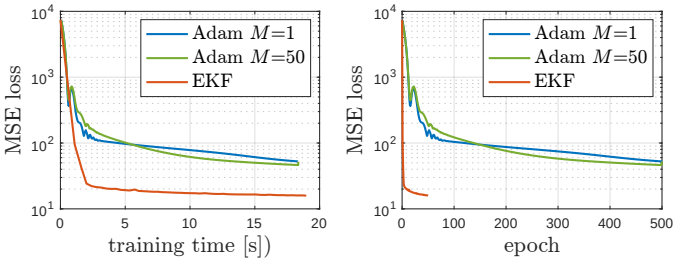

Fig. 1: Fluid damper benchmark: MSE loss $\frac{1}{2N}\sum_{k=0}^{N-1}(y(k)-\hat{y}(k))^2$ evaluated at each epoch

$= 100\left(1 - \frac{\|Y-\hat{Y}\|_2}{\|Y-\bar{y}\|_2}\right)$ for numeric outputs, where Y is the vector of measured output samples, $\hat{Y}$ the vector of output samples simulated by the identified model fed in open-loop with the input data, and $\bar{y}$ is the mean of Y, and by the accuracy $a = \frac{1}{N}\sum_{k=1}^{N}\delta_{\hat{y}_b(k),y(k)}$ for binary outputs, where $\hat{y}_b(k) = 0$ if $\hat{y}(k) < 0.5$ or 1 otherwise. In all tests, we initialize the weights of the neural networks of the model by using Xavier initialization [33], with zero bias terms.

A. Fluid damper benchmark

We consider the magneto-rheological fluid damper problem [34] used in the System Identification (SYS-ID) Toolbox for MATLAB R2022b [35] for nonlinear autoregressive (NARX) model identification, which consists of $N = 2000$ training data and 1499 test data. We want to train a RNN model (2) with $n_x = 4$ hidden states and shallow state-update and output network functions ($L_x = L_y = 2$) with $n_1^x = n_1^y = 6$ neurons, arctangent activation functions f_1^x, f_1^y, and linear output function f_2^y, and ℓ_2-regularization penalties $\rho_\theta = \rho_x = 10^{-3}$. We run the EKF-based learning method over $N_e = 50$ epochs and compare the results to those obtained by solving the fully condensed problem (3) and the partially-condensed problem (8) with $M = 50$ by using Adam with learning rate $l_r = 0.005$ (the value of l_r was chosen to get a good tradeoff between convergence speed and avoiding excessive oscillations).

Fig. 1 shows the resulting MSE loss values in a typical run. When using EKF, after each epoch the initial condition x_0 is reconstructed by solving (20) with $\bar{u}(k) = u(k)$, $\bar{y}(k) = y(k)$ to evaluate the MSE loss. It is apparent that EKF reaches a good-quality model already after one pass through the training dataset and outperforms the other methods. Another advantage of EKF is that little effort was put on tuning the covariance matrices Q_y, Q_x, Q_θ, $P(0|-1)$ for the EKF, while Adam required a careful tuning of the learning rate l_r and also of the penalty γ in case $M > 1$.

Table I compares the fit results obtained by running the training methods on both the same RNN model structure and an LSTM model [16] with 4 hidden and 4 cell states (i.e., $n_x = 8$ states) and the same output function f_y. The table shows the mean and standard deviation obtained over 20 runs (with $N_e = 25$ used for EKF), starting from different initial model parameters, always computing the fit on the model with the lowest MSE obtained among all epochs. For further comparison, the best model `Narx_6_2` reported in [35] provides a fit of 88.18% on training data and 85.15% on test data.

Fig. 2 shows the mean BRF obtained over 20 runs when training the RNN model (2) under ℓ_1-regularization $\lambda\|\theta\|_1$, introduced in the EKF as in (19b), for different values of λ. The figure also shows the mean percentage of zero entries in the resulting parameter vector θ, where each entry θ_i such that $|\theta_i| \le 10^{-3}$ is set to zero after EKF training. As expected, for increasing values of λ the parameter vector gets more sparse, at the price of decreased prediction quality. For large values of λ (roughly $\lambda > 10^{-3}$), results start deteriorating,

TABLE I: FLUID DAMPER BENCHMARK: MEAN BFR (STANDARD DEVIATION) OBTAINED OVER 20 RUNS

	BFR	Adam $M=1$	Adam $M=50$	EKF
RNN $n_\theta = 107$	training	89.12 (1.83)	88.56 (1.85)	92.82 (0.33)
	test	85.51 (2.89)	83.75 (4.71)	89.78 (0.58)
LSTM $n_\theta = 139$	training	89.60 (1.34)	87.47 (2.90)	92.63 (0.43)
	test	85.56 (2.68)	80.62 (6.89)	88.97 (1.31)

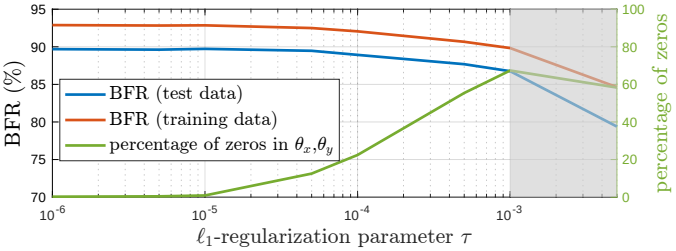

Fig. 2: Fluid damper benchmark: BRF and sparsity of θ optimized by EKF vs ℓ_1-regularization coefficient λ (mean values over 20 runs)

possibly due to the excessively large steps taken in (19b) that mine the convergence of the EKF.

B. Linear dynamical system with binary outputs

Consider 2000 input/output pairs generated by the following linear system with binary outputs

$$x(k+1) = \begin{bmatrix} .8 & .2 & -.1 \\ 0 & .9 & .1 \\ .1 & -.1 & .7 \end{bmatrix} x(k) + \begin{bmatrix} -1 \\ .5 \\ 1 \end{bmatrix} u(k) + \xi(k)$$

$$y(k) = \begin{cases} 1 & \text{if } \begin{bmatrix} -2 & 1.5 & 0.5 \end{bmatrix} x(k) - 2 + \zeta(k) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

from $x(0) = 0$, with the values of the input $u(k)$ changed with 90% probability from step k to $k+1$ with a new value drawn from the uniform distribution on $[0, 1]$. The disturbances $\xi_i(k), \zeta(k) \sim \mathcal{N}(0, \sigma^2)$, $i = 1, 2, 3$, are assumed independent. We consider the first $N = 1000$ samples for training, the rest for testing the model. We want to fit an affine model ($L_x = L_y = 1$) with sigmoidal output function $f_1^y(y) = 1/(1+e^{-A_1^y[x'(k)\ u(k)]' - b_1^y})$. Output data are not scaled.

We run EKF ($N_e = 25$) and Adam with $\rho_x = \rho_\theta = 10^{-2}$, the initial weights randomly generated as in [33] and further scaled by a factor $\frac{1}{20}$, and the remaining settings as in Section VI-A, modified cross-entropy loss $\ell_{\mathrm{CE}\epsilon}$ for $\epsilon = 0.005$, and EKF updates as in (15). Adam is run with learning rates selected by trial and error as $l_r = 0.01$ ($l_r = 0.001$) for $M = 1$ ($M = 50$) to trade off convergence rate and variance. Table II shows the accuracy obtained by EKF and Adam (for $M = 1$ and $M = 50$) for increasing values of σ. Note that we kept $Q_x = 10^{-10}$ in all tests, as we assumed not to know the intensity of the disturbances entering the system.

C. Nonlinear MPC benchmark: ethylene oxidation plant

We consider data generated from the ethylene oxidation plant model used as a nonlinear MPC benchmark in the Model Predictive Control Toolbox for MATLAB. A dataset of 2000 samples is generated by numerically integrating the system of nonlinear ordinary differential equations of the plant model with high accuracy and collecting samples every $T_s = 5$ s. The plant model has 4 states (gas density, C2H4 concentration, C2H4O concentration, and temperature

TABLE II: LINEAR SYSTEM WITH BINARY OUTPUTS: ACCURACY (%) ON TEST (TRAINING) DATA (MEAN VALUES OVER 20 RUNS)

σ	$M=1$	$M=50$	EKF
0.000	97.37 (96.86)	83.06 (86.07)	98.02 (97.91)
0.001	95.00 (98.41)	86.88 (88.41)	95.33 (98.66)
0.010	97.38 (97.47)	87.65 (85.59)	97.99 (98.52)
0.100	94.84 (94.49)	74.64 (83.94)	94.56 (95.44)
0.200	91.49 (90.80)	80.37 (82.88)	93.71 (92.22)

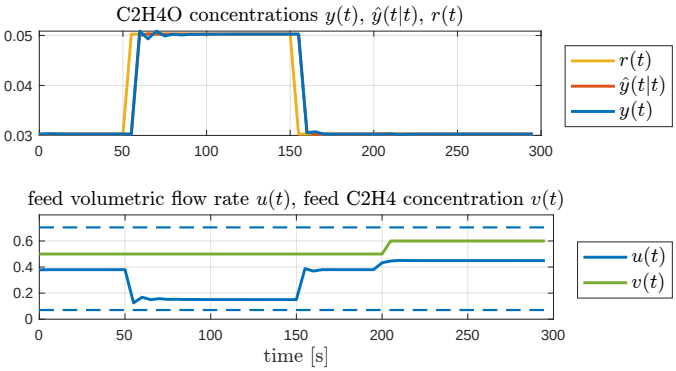

Fig. 3: Ethilene oxidation benchmark: NLMPC results

in the reactor), one output (y = C2H4O concentration), and two inputs (u = total volumetric feed flow rate, that can be manipulated, and v = C2H4 concentration of the feed, which is a measured disturbance). Half the dataset ($N = 1000$ samples) is used to train a RNN with $L_x = 3$ (i.e., a two-layer state-update neural network), $L_y = 1$, $n_x = 4$, $n_1^x = 6$, $n_2^x = 4$, affine output function, sigmoidal activation functions f_1^x, f_2^x, and unit output function f_1^y. We run the EKF-based training algorithm by processing the dataset in $N_e = 20$ epochs, which takes 4.54 s on the target machine. The resulting BRF is 94.33% on training data and 89.08% on test data.

The corresponding NLMPC controller, with MPC weights $W^{\Delta u} = 0.1$, $W^y = 10$, $u_{\min} = 0.0704$, $u_{\max} = 0.7042$, and prediction horizon $p = 10$, is implemented using MATLAB's `fmincon` solver with default parameters and no Jacobian information, warm-started from the shifted previous optimal solution. To close the feedback loop and get offset-free tracking of constant set-points, we apply EKF on line to estimate the state of the extended RNN model (22) with $B_d = 0$, $C_d = 1$ (output disturbance model); this corresponds to only adapting the bias coefficient b_1^y in (2b), with covariances $E[\xi(k)\xi(k)'] = 0.01I$, $E[\eta(k)^2] = 1$, $E[\zeta(k)^2] = 0.01$, and $P(0|-1) = I$. The obtained closed-loop results are depicted in Fig. 3, where it is apparent that a very good tracking is achieved despite the black-box RNN model used for prediction. The execution time ranges between 1.35 ms and 35.15 ms (8.86 ms on average) to solve the NLMPC problem and between 0.03 ms and 4.58 ms (0.15 ms on average) for state estimation.

VII. CONCLUSIONS

We have shown that EKF is an effective way of learning control-oriented RNN models from input and output data, even in the case of general strongly convex and twice-differentiable loss functions and ℓ_1-regularization. The approach is particularly suitable for online learning and model adaptation of recurrent neural networks and can be immediately extended to handle other classes of parametric nonlinear state-space models.

An interesting topic for future research is to study the conditions to impose on the RNN structure to make x and θ observable, in

particular to prevent over-parameterizing the model, and on how to choose model structure, loss function, and regularization terms to guarantee the asymptotic convergence of the filter.

REFERENCES

[1] J. Suykens, J. Vandewalle, and B. D. Moor, *Artificial neural networks for modelling and control of non-linear systems*. Springer Science & Business Media, 1995.

[2] V. Prasad and B. Bequette, "Nonlinear system identification and model reduction using artificial neural networks," *Computers & chemical engineering*, vol. 27, no. 12, pp. 1741–1754, 2003.

[3] D. Masti and A. Bemporad, "Learning nonlinear state-space models using autoencoders," vol. 129, p. 109666, 2021.

[4] R. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

[5] M. Forgione and D. Piga, "Model structures and fitting criteria for system identification with neural networks," in *14th IEEE International Conference on Application of Information and Communication Technologies (AICT)*, Tashkent, Uzbekistan, 2020.

[6] S. Singhal and L. Wu, "Training feed-forward networks with the extended Kalman algorithm," in *International Conference on Acoustics, Speech, and Signal Processing,*, 1989, pp. 1187–1190.

[7] G. Puskorius and L. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 279–297, 1994.

[8] X. Wang and Y. Huang, "Convergence study in extended Kalman filter-based training of recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 22, no. 4, pp. 588–600, 2011.

[9] R. Williams, "Training recurrent networks using the extended Kalman filter," in *IJCNN Int. Joint Conf. on Neural Networks*, vol. 4, 1992, pp. 241–246.

[10] M. Matthews and G. Moschytz, "Neural network nonlinear adaptive filtering using the extended Kalman filtering algorithm," in *Proc. of the Int. Neural Networks Conf.*, Paris, France, 1990, pp. 115–119.

[11] M. Livstone, J. Farrell, and W. Baker, "A computationally efficient algorithm for training recurrent connectionist networks," in *Proc. American Control Conference*, 1992, pp. 555–561.

[12] D. Mirikitani and N. Nikolaev, "Efficient online recurrent connectionist learning with the ensemble Kalman filter," *Neurocomputing*, vol. 73, no. 4-6, pp. 1024–1030, 2010.

[13] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter," *IEEE Transactions on Signal Processing*, vol. 40, no. 4, pp. 959–966, 1992.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] L. Ljung, "Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 1, pp. 36–50, 1979.

[16] F. Bonassi, E. Terzi, M. Farina, and R. Scattolini, "LSTM neural networks: Input to state stability and probabilistic safety verification," in *Learning for Dynamics and Control, Proceedings of Machine Learning Research*, 2020, pp. 85–94.

[17] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[18] A. Bemporad, "Training recurrent neural networks by sequential least squares and the alternating direction method of multipliers," 2022, http://arxiv.org/abs/2112.15348.

[19] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[20] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[21] G. Hicks and W. Ray, "Approximation methods for optimal control synthesis," *The Canadian Journal of Chemical Engineering*, vol. 49, no. 4, pp. 522–528, 1971.

[22] A. Bemporad and G. Cimini, "Reduction of the number of variables in parametric constrained least-squares problems," 2020, available on arXiv at http://arxiv.org/abs/2012.10423.

[23] B. Evens, P. Latafat, A. Themelis, J. Suykens, and P. Patrinos, "Neural network training as an optimal control problem: An augmented Lagrangian approach," in *Proc. 60^th Conf. Decision and Control*, Austin, TX, USA, 2021, pp. 5136–5143.

[24] H. Bock and K.-J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984.

[25] D. Axehill, "Controlling the level of sparsity in MPC," *Systems & Control Letters*, vol. 76, pp. 1–7, 2015.

[26] J. Humpherys, P. Redd, and J. West, "A fresh look at the Kalman filter," *SIAM Review*, vol. 54, no. 4, pp. 801–823, 2012.

[27] M. Boutayeb, H. Rafaralahy, and M. Darouach, "Convergence analysis of the extended Kalman filter used as an observer for nonlinear deterministic discrete-time systems," *IEEE Transactions on Automatic Control*, vol. 42, no. 4, pp. 581–586, 1997.

[28] B. Gibbs, *Advanced Kalman filtering, least-squares and modeling: a practical handbook*. John Wiley & Sons, 2011.

[29] M. Vaccari, D. Bonvin, F. Pelagagge, and G. Pannocchia, "Offset-free economic MPC based on modifier adaptation: Investigation of several gradient-estimation techniques," *Processes*, vol. 9, no. 5, p. 901, 2021.

[30] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[31] A. Vaz and L. Vicente, "PSwarm: A hybrid solver for linearly constrained global derivative-free optimization," *Optimization Methods and Software*, vol. 24, pp. 669–685, 2009, http://www.norg.uminho.pt/aivaz/pswarm/.

[32] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13^th Int. Conf. Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[34] J. Wang, A. Sano, T. Chen, and B. Huang, "Identification of Hammerstein systems without explicit parameterisation of non-linearity," *International Journal of Control*, vol. 82, no. 5, pp. 937–952, 2009.

[35] L. Ljung, *System Identification Toolbox for MATLAB*. The Mathworks, Inc., https://www.mathworks.com/help/ident.