ON THE ADAPTATION OF RECURRENT NEURAL NETWORKS FOR SYSTEM IDENTIFICATION [*]

Marco Forgione
IDSIA[†]
Lugano
marco.forgione@supsi.ch

Aneri Muni
NNAISENSE[‡]
Lugano
aneri.muni@nnaisense.com

Dario Piga
IDSIA
Lugano
dario.piga@supsi.ch

Marco Gallieri
NNAISENSE
Lugano
marco.gallieri@nnaisense.com

ABSTRACT

This paper presents a transfer learning approach which enables fast and efficient adaptation of Recurrent Neural Network (RNN) models of dynamical systems. A *nominal* RNN model is first identified using available measurements. The system dynamics are then assumed to change, leading to an unacceptable degradation of the nominal model performance on the *perturbed* system. To cope with the mismatch, the model is augmented with an additive correction term trained on fresh data from the new dynamic regime. The correction term is learned through a Jacobian Feature Regression (JFR) method defined in terms of the features spanned by the model's Jacobian with respect to its nominal parameters. A non-parametric view of the approach is also proposed, which extends recent work on Gaussian Process (GP) with Neural Tangent Kernel (NTK-GP) to the RNN case (RNTK-GP). This can be more efficient for very large networks or when only few data points are available. Implementation aspects for fast and efficient computation of the correction term, as well as the initial state estimation for the RNN model are described. Numerical examples show the effectiveness of the proposed methodology in presence of significant system variations.

***K*eywords** Identification methods · Deep learning · Linear/nonlinear models · Recurrent neural networks · Model adaptation

1 Introduction

In the Deep Learning (DL) field, expressive model structures are defined in terms of complex compositions of simple linear/non-linear building blocks [1]. Furthermore, model learning is performed conveniently using general gradient-based optimization, and leveraging Automatic Differentiation (AD) for gradient computations [2]. Nowadays, specialized hardware and high-quality software tools for DL are available [3], easing the practitioner's work to a large extent.

In recent years, more and more research aimed at exploiting modern Deep Learning tools and concepts in System Identification (SI) has been pursued. Certain contributions are aimed at adapting existing DL architectures and training algorithms to the specific needs of SI. For instance, [4, 5] propose different approaches to deal with the initial conditions in the training of Recurrent Neural Networks (RNN), while in [6] 1D Convolutional Neural Networks are specialized and tested on SI tasks. Other contributions propose new DL-based model structures explicitly conceived for SI

[*] *Corresponding author*: M. Forgione.

[†] Dalle Molle Institute for Artificial Intelligence, IDSIA USI-SUPSI, Via la Santa 1, CH-6962 Lugano-Viganello, Switzerland.

[‡] NNAISENSE SA, Piazza Molino Nuovo 17, CH-6900 Lugano, Switzerland.

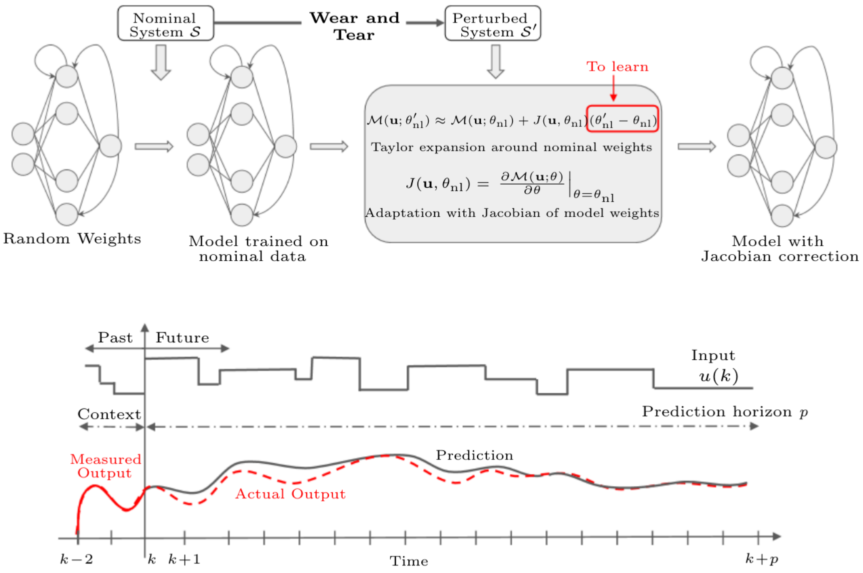
Nominal System $\mathcal{S}$
Wear and Tear
Perturbed System $\mathcal{S}'$
To learn
$\mathcal{M}(\mathbf{u};\theta'_{\mathrm{nl}}) \approx \mathcal{M}(\mathbf{u};\theta_{\mathrm{nl}}) + J(\mathbf{u},\theta_{\mathrm{nl}})(\theta'_{\mathrm{nl}}-\theta_{\mathrm{nl}})$
Taylor expansion around nominal weights
$J(\mathbf{u},\theta_{\mathrm{nl}}) = \frac{\partial \mathcal{M}(\mathbf{u};\theta)}{\partial \theta}\Big|_{\theta=\theta_{\mathrm{nl}}}$
Adaptation with Jacobian of model weights
Random Weights
Model trained on nominal data
Model with Jacobian correction
Past
Future
Input
$u(k)$
Context
Prediction horizon p
Measured Output
Prediction
Actual Output
$k-2$
k
$k+1$
Time
$k+p$

Figure 1: **(Top)** Nominal RNN model training and JFR adaptation with Jacobian features. **(Bottom)** Sequence handling for initialization of the RNN state.

purposes. For instance, [7] introduces a novel neural architecture which includes linear transfer functions as elementary building blocks, while [8] proposes model structures based on Koopman operator theory [9], and lift non-linear state-space dynamics to a higher-dimensional space where the dynamics are linear. The mapping from the original to the higher-dimensional space is learned using DL tools.

DL-based models for SI have shown to deliver state-of-the-art performance on standard benchmarks, see [8, 10]. In practice, however, the dynamics of real-world systems often vary over time due to, e.g., aging, changing configurations, and different external conditions. An open challenge is thus to *adapt* the identified DL-based dynamical models over time. In the *systems and control* field, different model adaptation techniques have been introduced and successfully tested in practice. Joint estimation of states and (slowly-varying) model parameters has been tackled by enriching the model structure with a stochastic description of parameter evolution over time, and applying standard filtering techniques. Non-linear extension of the Kalman Filter (KF) such as the Unscented Kalman Filter (UKF) [11] have been considered to obtain an (approximately) optimal estimate of state and parameters.

For models which are linear in the parameters, Bayesian Linear Regression (BLR) and *Ridge* regression [12] as well as Recursive Least Squares (RLS) [13] enable closed-form, even on-line estimation of the model parameters. For models that are not linear, on the other hand, the computational burden may become an issue. In [11], the UKF has been used to adapt (online) the last layer of a particular neural architecture. This entails the evaluation of the *forward pass* two times the number of states and parameters, plus the inversion of a large matrix. The procedure is repeated as each new measurement becomes available and carries over an approximate prior.

In this paper, we instead propose an approach to compute a local linear (with respect to its parameters) correction of the model dynamics. We use a Recurrent Neural Network nominal model and evaluate the linearization on entire sequences from the perturbed system in order to produce (state-dependent) Jacobian features that are used to perform JFR, and thus to update the model using efficient least-squares methods. Our approach is inspired by the recent transfer learning literature [14], where a first-order Taylor expansion of a nominal feed-forward neural network (modeling a static process) is introduced for model updating. With respect to [14], the contribution of the current paper is threefold:

- we extend the methodology of [14] to dynamical systems modeled as RNNs. In particular, we introduce specialized and computationally efficient derivations to obtain the RNN Jacobian features (needed for model updating) through *recursive* operations;
- we introduce an approach to initialize the RNN state based on past data, so that an estimate of the initial state is not needed on top of the parameter estimation;
- we extend the non-parametric view of the approach in [14] by using the RNN Jacobian features to define a Recurrent Neural Tangent Kernel (RNTK) to be employed in a Gaussian Process (GP) framework.

The rest of this paper is organized as follows. Section 2 is devoted to the problem formulation. The proposed approaches for nominal model training with initial state estimation and transfer learning are presented in Section 3. Details for efficient implementation are discussed in Section 4, along with analysis of the required computational and memory costs. The effectiveness of the proposed methodology is showcased in Section 5 on numerical examples from the chemical and electrical domains.

2 Problem formulation

Let us consider a discrete-time *dynamical system* $\mathcal{S}$ taking input values $u \in \mathbb{R}^{n_u}$ and generating output values $y \in \mathbb{R}^{n_y}$. At a given discrete-time step k, the system output possibly depends on all the previous input samples, *i.e.*,

$$y_k = \mathcal{S}(u_k, u_{k-1}, \dots, u_0). \tag{1}$$

Let $\mathcal{M}$ be a non-linear *dynamical model structure* sufficiently expressive to describe the system $\mathcal{S}$:

$$\hat{y}_k = \mathcal{M}(u_k, u_{k-1}, \dots, u_0;\ \theta), \tag{2}$$

where $\theta \in \mathbb{R}^{n_\theta}$ is a *parameter vector* to be determined.

We use the bold-face symbols $\mathbf{u}$ and $\mathbf{y}$ to denote the sequences of N input and output samples collected during an experiment. For the mathematical derivations, $\mathbf{u}$ and $\mathbf{y}$ may be interpreted as real-valued vectors of size N. The true system and the model dynamics will be written compactly in vector notation as $\mathcal{S}(\mathbf{u})$ and $\mathcal{M}(\mathbf{u};\ \theta)$, respectively.

In this paper, we assume that the model admits the following state-space representation:

$$x_{k+1} = \mathcal{F}(x_k, u_k; \theta_\mathcal{F}) \tag{3a}$$

$$\hat{y}_k = \mathcal{G}(x_k, u_k; \theta_\mathcal{G}), \tag{3b}$$

where $x_k \in \mathbb{R}^{n_x}$ is the *state vector* at time k, and $\mathcal{F}, \mathcal{G}$ are the state-update and output mappings, respectively, both parametrized by $\theta = \{\theta_\mathcal{F}, \theta_\mathcal{G}\}$. In particular, $\mathcal{F}$ and $\mathcal{G}$ in this paper are neural networks, and the overall model (3) is a Recurrent Neural Network (RNN).

We denote by $\mathrm{ops}(\mathcal{F})$ and $\mathrm{ops}(\mathcal{G})$ the computational cost required to simulate $\mathcal{F}$ and $\mathcal{G}$, respectively and we assume that $\mathrm{ops}(\mathcal{F})$ and $\mathrm{ops}(\mathcal{G})$ are both $\mathcal{O}(n_\theta)$. The computational cost of simulating the RNN over N time steps is thus $\mathrm{ops}(\mathcal{M}) = \mathrm{N}\left(\mathrm{ops}(\mathcal{F}) + \mathrm{ops}(\mathcal{G})\right)$, or simply $\mathcal{O}(Nn_\theta)$.

The system $\mathcal{S}$ operates initially in its *nominal* configuration. In this configuration, the training dataset $\mathcal{D}_\mathrm{tr} = \{\mathbf{u}_\mathrm{tr}, \mathbf{y}_\mathrm{tr}\}$ and the test dataset $\mathcal{D}_\mathrm{tst} = \{\mathbf{u}_\mathrm{tst}, \mathbf{y}_\mathrm{tst}\}$ are collected. These datasets are used to learn the parameter vector θ_nl of a *nominal* non-linear model and to asses its performance, respectively.

After some time, the system dynamics changes due to, e.g., aging or different external conditions. We denote by $\mathcal{S}'$ the system in this *perturbed* configuration and we assume that the nominal model (previously trained on $\mathcal{D}_\mathrm{tr}$) no longer describes of the behavior of $\mathcal{S}'$ with sufficient accuracy.

While we could re-run the full non-linear model training procedure on a new training dataset collected in the perturbed configuration, we would like to exploit the previously trained model and *adapt* it to obtain a description of the perturbed system in a more efficient manner. In this spirit, we collect a *transfer dataset* $\mathcal{D}_\mathrm{xf} = \{\mathbf{u}_\mathrm{xf}, \mathbf{y}_\mathrm{xf}\}$ and an *evaluation dataset* $\mathcal{D}_\mathrm{ev} = \{\mathbf{u}_\mathrm{ev}, \mathbf{y}_\mathrm{ev}\}$ from $\mathcal{S}'$. The purposes of these datasets are to adapt the nominal model to the perturbed configuration and to evaluate its performance, respectively.

In general, the model adaptation procedure should be less computational intensive and/or require less data than a (trivial) repetition of the training procedure from scratch.

In the following sections, derivations are shown for the single-input-single-output case for notational simplicity (i.e. $n_u = n_y = 1$). Extension to the multi-input-multi-output case is trivial, and considered in the Continuous-flow Stirred-Tank Reactor example presented in Section 5.

3 Model training and adaptation

3.1 Nominal model training

The parameter vector θ_nl of the nominal model may be selected by minimizing a regression loss such as the *mean square error* $\mathrm{MSE}(\mathcal{D}_\mathrm{tr}, \theta)$ with respect to θ on the training dataset $\mathcal{D}_\mathrm{tr}$:

$$\theta_\mathrm{nl} = \arg\min_{\theta \in \mathbb{R}^{n_\theta}} \overbrace{\frac{1}{N}\left\|\mathbf{y}_\mathrm{tr} - \mathcal{M}(\mathbf{u}_\mathrm{tr}; \theta)\right\|^2}^{=\mathrm{MSE}(\mathcal{D}_\mathrm{tr}, \theta)}. \tag{4}$$

The loss is minimized through iterative gradient-based optimization algorithms, such as plain gradient descent or modern variants like Adam [15]. For the required derivatives computation, standard reverse-mode Automatic Differentiation (AD) algorithms and software are used [3]. Accordingly, the number of operations required to compute the gradient of the loss w.r.t. θ is $k\text{ops}(\mathcal{M})$, where k is a constant guaranteed to be $k < 6$, see [2]. Thus, gradient computation (which is the most expensive operation for each iteration of gradient-based optimization) for a RNN model has cost $\mathcal{O}(Nn_\theta)$. Of course, a certain number n_{iter} of optimization iterations (hard to determine a priori) is needed to reduce the training loss to a reasonably low value. The overall computational cost of solving (4) is thus $\mathcal{O}(n_{\text{iter}}Nn_\theta)$. We will not discuss the nominal model training in more details, as standard practice is followed.

3.2 RNN initial state estimation

The initial state of a RNN, generally set to zero, can affect the transient predictions from the model. Capturing the first steps accurately is actually very important, for instance in applications such as Model Predictive Control (MPC) [16]. Estimating the initial state is addressed in [17] using a past window of both measurements and outputs, with guarantees provided for Echo State Networks (ESN). We extend this idea (without explicit guarantees) to a more general class of RNNs, where the predicted output $\hat{y}_k$ is part of the state:

$$x_{k+1} = \mathcal{F}(x_k, \hat{y}_k, u_k; \theta_{\mathcal{F}}) \tag{5a}$$

$$\hat{y}_{k+1} = \mathcal{G}(x_{k+1}, \hat{y}_k, u_k; \theta_{\mathcal{G}}). \tag{5b}$$

While predictions are performed by (5), the initial state is estimated using a window of past data (context), $\{y_{k-1}, \dots, y_{k-N_c}, u_{k-1}, \dots, u_{k-N_c}\}$, where the *measured* outpus y_{k-i} are sequentially fed to the model instead of the predicted ones for N_c steps, and x_{k-N_c} is set to zero. In other words, the state estimation is performed by opening the output prediction loop for the first for N_c steps:

$$x_{k+1} = \mathcal{F}(x_k, y_k, u_k; \theta_{\mathcal{F}}) \tag{6a}$$

$$\hat{y}_{k+1} = \mathcal{G}(x_{k+1}, y_k, u_k; \theta_{\mathcal{G}}), \tag{6b}$$

$$\text{for } k = 0, 1, \dots, N_c - 1. \tag{6c}$$

Since the state estimator is the model itself, we train by means of a joint backward pass through both estimation and forecasting. Sequence are split as in Figure 1 (bottom chart).

3.3 Model adaptation

Suppose the dynamical system $\mathcal{S}$ changes over time, and the performance of the nominal trained model $\mathcal{M}(\cdot; \theta_{\text{nl}})$ on the new dynamics $\mathcal{S}'$ drops to an unacceptable level. Following the reasoning in [14], we exploit the Jacobian $J(\mathbf{u}, \theta_{\text{nl}}) \in \mathbb{R}^{N \times n_\theta}$ of the nominal model $\mathcal{M}(\cdot; \theta_{\text{nl}})$ with respect to the parameter vector θ:

$$J(\mathbf{u}, \theta_{\text{nl}}) = \left.\frac{\partial \mathcal{M}(\mathbf{u}; \theta)}{\partial \theta}\right|_{\theta = \theta_{\text{nl}}} \tag{7}$$

to define an *adaptation* of the nominal model:

$$\mathcal{M}_\ell(\mathbf{u}; \theta_{\text{nl}}, \theta_\ell) = \mathcal{M}(\mathbf{u}; \theta_{\text{nl}}) + J(\mathbf{u}, \theta_{\text{nl}})\theta_\ell, \tag{8}$$

which depends *linearly* on a new parameter vector $\theta_\ell \in \mathbb{R}^{n_\theta}$, to be determined. The rationale behind this approach is the following—the perturbed system dynamics $\mathcal{S}'$ may be described by a model $\mathcal{M}(\mathbf{u}; \theta'_{\text{nl}})$ with a parameter θ'_{nl} slightly different from θ_{nl}. Then, $\mathcal{M}(\mathbf{u}; \theta'_{\text{nl}})$ may be approximated through a first-order Taylor expansion centered at the nominal parameter θ_{nl}:

$$\mathcal{M}(\mathbf{u}; \theta'_{\text{nl}}) \approx \mathcal{M}(\mathbf{u}; \theta_{\text{nl}}) + J(\mathbf{u}, \theta_{\text{nl}})(\theta'_{\text{nl}} - \theta_{\text{nl}}),$$

which corresponds indeed to the model structure (8) by setting $\theta_\ell = \theta'_{\text{nl}} - \theta_{\text{nl}}$.

In the following, for notation simplicity, we ignore the nominal non-linear contribution $\mathcal{M}(\mathbf{u}; \theta_{\text{nl}})$ at the right-hand side of model (8) and only retain the linear adaptation term $J(\mathbf{u}, \theta_{\text{nl}})\theta_\ell$. We estimate the parameters θ_ℓ of the linear adaptation term through a Jacobian Feature Regression (JFR), according to the probabilistic model:

$$\mathbf{y} = J(\mathbf{u}, \theta_{\text{nl}})\theta_\ell + \mathbf{e}, \qquad \mathbf{e} \sim \mathcal{N}(0, \sigma^2 I_N) \tag{9a}$$

$$\theta_\ell \sim \mathcal{N}(0, I_{n_\theta}). \tag{9b}$$

From (9), the *posterior* distribution of θ_ℓ conditioned on the transfer dataset $\mathcal{D}_{\text{xf}}$ is Gaussian:

$$\theta_\ell | \mathcal{D}_{\text{xf}} \sim \mathcal{N}(\bar{\theta}, \Sigma_{\theta_\ell}) \tag{10a}$$

with mean

$$\bar{\theta} = (J_{\text{xf}}^{\top} J_{\text{xf}} + \sigma^2 I_{n_\theta})^{-1} J_{\text{xf}}^{\top} \mathbf{y}_{\text{xf}} \tag{10b}$$

and covariance matrix:

$$\Sigma_\theta = \sigma^2 (J_{\text{xf}}^{\top} J_{\text{xf}} + \sigma^2 I_{n_\theta})^{-1}, \tag{10c}$$

where $J_{\text{xf}} = J(\mathbf{u}_{\text{xf}}, \theta_{\text{nl}})$.

The posterior distribution of θ_ℓ given by (10) may be used to infer the output distribution given a new input sequence $\mathbf{u}_*$ according to:

$$\mathbf{y}_* | \mathcal{D}_{\text{xf}}, \mathbf{u}_* \sim \mathcal{N}(\overbrace{J_* \bar{\theta}}^{=\bar{\mathbf{y}}_*},\ J_* \Sigma_\theta J_*^{\top}), \tag{11}$$

where $J_* = J(\mathbf{u}_*, \theta_{\text{nl}}) = \left.\frac{\partial \mathcal{M}(\mathbf{u}_*;\theta)}{\partial \theta}\right|_{\theta=\theta_{\text{nl}}}$.

3.4 Interpretation in function space

Similar to [14], the RNN-feature JFR introduced above can be equivalently formulated as a Gaussian Process (GP). In this work, the GP is parametrized with the finite-dimensional *Recurrent* Neural Tangent Kernel (RNTK) resulting from the RNN trained on nominal data. Using (10) and (11), we have the following predictive posterior output distribution:

$$\mathbf{y}_* | \mathbf{u}_*, \mathcal{D}_{\text{xf}} \sim \mathcal{N}\big(J_* (J_{\text{xf}}^{\top} J_{\text{xf}} + \sigma^2 I_{n_\theta})^{-1} J_{\text{xf}}^{\top} \mathbf{y}_{\text{xf}}, \sigma^2 J_* (J_{\text{xf}}^{\top} J_{\text{xf}} + \sigma^2 I_{n_\theta})^{-1} J_*^{\top}\big). \tag{12}$$

Using the matrix inversion lemma and following the derivations in [18, Chapter 1], the predictive posterior distribution may be equivalently written as in [14, Equation 3]:

$$\mathbf{y}_* | \mathbf{u}_*, \mathcal{D}_{\text{xf}} \sim \mathcal{N}(J_* J_{\text{xf}}^{\top} (J_{\text{xf}} J_{\text{xf}}^{\top} + \sigma^2 I_N)^{-1} \mathbf{y}_{\text{xf}}, \sigma^2 J_* (I_N - J_{\text{xf}}^{\top} (J_{\text{xf}} J_{\text{xf}}^{\top} + \sigma^2 I_{n_\theta})^{-1} J_{\text{xf}}) J_*^{\top}, \tag{13}$$

with the difference that, instead of computing J_{xf} for a deep network with point predictions, this is now computed for a RNN sequence-to-sequence task.

4 Implementation

The computational cost of inferring the posterior mean $\bar{\mathbf{y}}_*$ of the output $\mathbf{y}_*$ is analyzed in this section, both in parameter and in function space.

4.1 Parameter-space inference

The adaptation entails two steps: 1) Run the RNN on a fresh, small dataset and compute the posterior parameter mean $\bar{\theta}$; 2) Run the RNN on new data, compute the predictive posterior mean based on $\bar{\theta}$ and the current Jacobian. The scheme is depicted at the top of Figure 1.

For parameter-space inference, the main problem to be tackled is the computation of the posterior parameter mean $\bar{\theta}$. Once $\bar{\theta}$ is available, the posterior output mean is given by:

$$\bar{\mathbf{y}}_* = J_* \bar{\theta},$$

see eq. (11). The *Jacobian-vector product* $J_* \bar{\theta}$ can be obtained efficiently exploiting the same automatic differentiation tools already used in 3.1 to compute (for each iteration of gradient-based optimization) the derivatives of the training loss (4), thus with a number of operations $\mathcal{O}(N n_\theta)$, and without explicitly constructing the Jacobian matrix J_*. The reader is referred to [19, 2] for the details about Jacobian-vector product computations through AD.

In the following paragraphs, we discuss three different implementation approaches to compute the posterior parameter mean $\bar{\theta}$, along with a computationally efficient derivation to obtain the Jacobian J_{xf}.

4.1.1 Offline implementation

In the offline implementation approach, the estimation of $\bar{\theta}$ is carried out once the complete transfer dataset $\mathcal{D}_{\text{xf}}$ is available. The main challenge is to compute the full Jacobian matrix J_{xf}. Once available, the system of linear equations (10b) is constructed and solved to obtain $\bar{\theta}$. Details are provided below.

Computation of the Jacobian matrix J_{xf}. A straightforward approach to compute the Jacobian $J_{\mathrm{xf}} \in \mathbb{R}^{N \times n_\theta}$ consists in invoking N independent reverse-mode automatic differentiation (back-propagation) operations to construct the N rows of the matrix. The computational cost of this naïve approach is $\mathcal{O}(N^2 n_\theta)$.

In this paper, we introduce an alternative approach tailored to the RNN model (3), based on a recursive construction of the Jacobian matrix, which exploits forward sensitivity equations [20]. This approach has a computational cost $\mathcal{O}(N(n_x + n_y)n_\theta)$.

Let us introduce the *state sensitivities* $s_k = \frac{\partial x_k}{\partial \theta} \in R^{n_x \times n_\theta}$. By taking the derivatives of the left- and right-hand side of (3a) w.r.t. the model parameters θ, we obtain a recursive equation describing the evolution of s_k:

$$s_{k+1} = J_k^{fx} s_k + J_k^{f\theta}, \tag{14a}$$

where $J_k^{fx} \in \mathbb{R}^{n_x \times n_x}$ and $J_k^{f\theta} \in \mathbb{R}^{n_x \times n_\theta}$ are the Jacobians of $\mathcal{F}(x_k, u_k; \theta)$ w.r.t. x_k and θ, respectively.

Let us now take the derivative of (3b) w.r.t. θ, which corresponds by definition to the k-th row of J_{xf}:

$$\frac{\partial y_k}{\partial \theta} = J_k^{gx} s_k + J_k^{g\theta}, \tag{14b}$$

where $J_k^{gx} \in \mathbb{R}^{n_y \times n_x}$ and $J_k^{g\theta} \in \mathbb{R}^{ny \times n_\theta}$ are the Jacobians of $\mathcal{G}(x_k, u_k; \theta)$ w.r.t. x_k and θ, respectively.

The Jacobians J_k^{fx} and $J_k^{f\theta}$ can be obtained through n_x back-propagation operations through $\mathcal{F}$, thus at cost $\mathcal{O}(n_x n_\theta)$. Similarly, J_k^{gx} and $J_k^{g\theta}$ can be obtained through n_y back-propagation operations through $\mathcal{G}$ at cost $n_y n_\theta$. Overall, the computational cost of obtaining $\frac{\partial y_k}{\partial \theta}$ in (14b) (given the previous sensitivity s_{k-1}) is $\mathcal{O}((n_x + n_y)n_\theta)$. Thus, J_{xf} is obtained at a cost $\mathcal{O}(N(n_x + n_y)n_\theta)$.

Solution of the linear system. The linear system to be solved in order to obtain $\bar{\theta}$ in (10b) is:

$$\overbrace{(J_{\mathrm{xf}}^\top J_{\mathrm{xf}} + \sigma^2 I_{n_\theta})}^{=A} \bar{\theta} = \overbrace{J_{\mathrm{xf}}^\top \mathbf{y}_{\mathrm{xf}}}^{=b}, \tag{15}$$

where the system matrix $A \in \mathbb{R}^{n_\theta \times n_\theta}$ is symmetric and positive definite. The standard approach to solve such a linear system involves application of the Cholesky factorization [21, Appendix A], which requires approximately $\frac{{n_\theta}^3}{3}$ operations. Thus, the computational cost is $\mathcal{O}({n_\theta}^3)$, while the memory cost is $\mathcal{O}({n_\theta}^2)$, since the system matrix A has to be stored explicitly into memory.

4.1.2 Online implementation

Thanks to the linear-in-the-parameters structure of the adapted model $\mathcal{M}_\ell$ in (8), the parameter vector $\bar{\theta}$ may be estimated in real time by applying the Recursive Least Squares (RLS) algorithm [22, Chapter 11], as soon as a new sample is measured.

Implementation of RLS requires, at each time step, simple linear algebra operations (sums and multiplications) on matrices of size $n_\theta \times n_\theta$ and vectors of size n_θ. Furthermore, the Jacobian feature $\frac{\partial y_k}{\partial \theta}$ (which is the regressor required in RLS at time step k) is recursively obtained by exploiting the forward sensitivities equations (14), thus with computational cost $\mathcal{O}((n_x + n_y)n_\theta)$.

4.1.3 Limited-memory implementation

In certain cases, the number of parameters n_θ may be too large to adopt the approaches discussed above. In particular, the matrices A in (15) and the matrices to be updated in RLS, all having size $n_\theta \times n_\theta$, may exceed the computing device's available memory for large-scale RNN models.

In this case, as already mentioned at the beginning of Section 4.1, for a given vector v of compatible size, it is still possible to compute the Jacobian-vector product $J_{\mathrm{xf}} v$ (and actually also the *transposed* Jacobian-vector product $J_{\mathrm{xf}}^\top v$) using the very same machinery of a gradient evaluation through AD without explicitly computing and storing the Jacobian matrix J_{xf}, at computational cost $\mathcal{O}(N n_\theta)$.

For limited-memory computation of $\bar{\theta}$, it is then convenient to reformulate (10b) as the solution of the optimization problem:

$$\bar{\theta} = \arg\min_\theta \overbrace{\|\mathbf{y}_{\mathrm{xf}} - J_{\mathrm{xf}}\theta\|^2 + \sigma^2 \theta^\top \theta}^{=\mathcal{L}(\mathcal{D}_{\mathrm{xf}}, \theta)}. \tag{16}$$

Evaluation of the loss $\mathcal{L}(\mathcal{D}_{\text{xf}}, \theta)$ in (16) may be performed by computing the Jacobian-vector product $J_{\text{xf}}\theta$ through AD. Furthermore, the gradients $\frac{\partial \mathcal{L}(\mathcal{D}_{\text{xf}},\theta)}{\partial \theta}$ of the loss $\mathcal{L}$ w.r.t. θ is:

$$\frac{\partial \mathcal{L}(\mathcal{D}_{\text{xf}}, \theta)}{\partial \theta} = 2\; J_{\text{xf}}^{\top}(J_{\text{xf}}\theta - \mathbf{y}_{\text{xf}}) + 2\;\sigma^2\theta,$$

and it can be obtained through the additional transposed Jacobian-vector product $J_{\text{xf}}^{\top}(J_{\text{xf}}\theta - \mathbf{y}_{\text{xf}})$. Thus, an approximate numerical solution to (16) may be obtained by means of *iterative* gradient-based optimization techniques, where each iteration has computational cost $\mathcal{O}(Nn_{\theta})$ and the same memory cost of back-propagation through the RNN model.

4.2 Function-space inference

As shown in [14], the function-space formulation (13) can leverage the fast Jacobian-vector products, similar to our parametric limited-memory approach. The final model correction is computed by means of the conjugate gradient method [21, Chapter 5], for an overall effort of $\mathcal{O}(N^2)$ for each iteration of the conjugate gradient. Thus, the approach can become favourable for very large models and small datasets, provided that convergence occurs within few iterations.

The implementation of [14] is used for the RNTK GP. The network state is initialized first using the nominal model, then a NTK GP is created which accepts 2D tensors as the network input and output. For compatibility, given the estimated RNN inital state, the model is wrapped in an interface that allows the IO sequence length to be collapsed over the batch size. Note that the contribution of past states and inputs is accounted for when computing the backward passes used for Jacobian-vector products. In this respect, the proposed RNTK extends the approach of [14].

5 Examples

Two demonstrators are presented, respectively, from the chemical and electrical domain. For both examples, we have defined a nominal and a perturbed data-generating system. The nominal system is used to generate the training and the test datasets, while the perturbed system is used to generate the transfer and the evaluation datasets. First, a nominal model is estimated on the training dataset using standard gradient-based approaches, and its performance is evaluated on the test dataset. Then, the nominal model is adapted to the perturbed configuration using data from the transfer dataset, following the approach described in Section 4. On the evaluation dataset, we can finally measure the performance of the adapted model in the perturbed configuration and compare it to the performance of the nominal model.

The performance of the estimated nominal and adapted model is assessed through the R^2 *performance index*:

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\|\mathbf{y} - \overline{\mathbf{y}}\|^2}, \tag{17}$$

where $\overline{\mathbf{y}}$ denotes the mean of the measured output $\mathbf{y}$, and $\hat{\mathbf{y}}$ is the predicted output from the model.

The developed software is based on the PyTorch DL framework [3]. The implementation for function-space inference builds upon the codes of the paper [14], adapted to handle a RNN nominal model. Instead, the parameter-space methods have been developed from scratch. All our codes are available on the GitHub repository `https://github.com/forgi86/RNN-adaptation` and allow full reproduction of the results in this paper.

We run all the experiments PC equipped with an Intel i7-8550U CPU and 16 GB of RAM.

5.1 Continuous-flow Stirred-Tank Reactor (CSTR)

As a first case study, we consider the CSTR system [23], with continuous-time dynamics described by the following *ordinary differential equations*:

$$\begin{aligned}\dot{C}_A &= q(C_A^0 - C_A) - k^1 C_A + k^4 C_R \\ \dot{C}_R &= q(1 - C_A^0 - C_R) + k^1 C_R + k^3(1 - C_A - C_R) \\ &\quad -(k^2 + k^4)C_R,\end{aligned} \tag{18}$$

with inputs T and q denoting, the temperature and the flow rate, respectively; output concentrations C_A and C_R; and coefficient k^i defined as

$$k^i = k_0^i \exp\left(-E^i\left(\frac{1}{T} - 1\right)\right), \qquad i = 1, 2, 3, 4. \tag{19}$$

The nominal and perturbed system parameters are reported in Table 1.

The model (18) is discretized with sampling time $T_s = 0.1$ s. We collect a training dataset consisting of 64 sequences containing 256 samples, and a test, transfer, and evaluation dataset each consisting of a single sequence of length 1024. In all datasets, the input temperature T is a triangular wave and the feed rate q is a step signal with random amplitude values.

We model the system with an LSTM network [24, 25] having two inputs; one hidden layer with 16 units; and 2 output units. The nominal LSTM model parameters are obtained by minimizing the mean square simulation error on the training dataset using standard gradient descent techniques. Moreover, to assist with the tracking of the signal transient, we initialize the LSTM with a context, as illustrated in Section 3.2.

The achieved R^2 indexes in estimating the two outputs C_A and C_R are reported in Table 3, where we can observe that the performance of the nominal LSTM model is satisfactory on both the training and test datasets. However, the performance index drops significantly on the transfer and evaluation datasets. This shows that the nominal LSTM model cannot accurately describe the perturbed system dynamics.

We now utilize the transfer dataset to adapt the nominal model using the following three different approaches described in Sections 3 and 4:

- Jacobian Feature Regression (JFR) with standard offline implementation described in Section 3.3 and based on eqs. (14) and (15);
- Limited-memory Jacobian Feature Regression (LM-JFR) based on an iterative gradient-based minimization of (16), exploiting Jacobian-vector products for gradient computation, as described in Section 4.1.3;
- Function-space approach based on a Gaussian Process with kernel designed from the Jacobian of the LSTM (GP-LSTM), as described in Sections 3.4 and 4.2.

Fig. 2 compares the (normalized) ground truth output with the nominal LSTM output and the outputs of the JFR, LM-JFR, and GP-LSTM model adaptation approaches on the evaluation dataset. The outputs of the three model adaptation approaches are shown to be equal (up to small numerical deviations). The R^2 index of the nominal LSTM and of the adapted model (equivalent for the three approaches) is reported in Table 3. We observe that on the evaluation dataset the adapted model achieves excellent predictive performances, with an R^2 index above 0.99 for both output channels. Conversely, the performance of the nominal LSTM decreases dramatically on the evaluation dataset (R^2 index of 0.5/-0.74 for the two output channels).

The run time of the three model adaptation approaches is reported in Table 2, which show that, for this particular example, JFR turned out to be the fastest one.

5.2 Non-linear RLC

We consider the non-linear RLC series circuit benchmark introduced in [5] and described by:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ \frac{-1}{L(i_L)} & \frac{-R}{L(i_L)} \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L(i_L)} \end{bmatrix} v_{in}, \tag{20}$$

where v_{in} (V) is the input voltage; v_C (V) is the capacitor voltage; and i_L (A) is the inductor current. The resistance R and capacitance C are constant, while the inductance L depends on i_L according to:

$$L(i_L) = L_0 \left[\left(\frac{0.9}{\pi} \arctan\big(-5(|i_L| - 5\big) + 0.5 \right) + 0.1 \right].$$

The coefficients characterizing the nominal and the perturbed systems are reported in Table 4. All datasets contain 2000 samples obtained by simulating the (nominal or perturbed) system with discretization step $T_s = 1$ μs. The input is a filtered white noise with bandwidth $80, 90, 100, 100$ kHz and standard deviation $80, 70, 70, 70$ V for the training, test, transfer, and evaluation datasets, respectively. In the training and transfer datasets, the output v_C is corrupted by an additive white noise term. The corresponding signal-to-noise ratios are 20.0 and 18.8 dB, respectively.

Table 1: CSTR example: nominal and perturbed system parameters.

System	**Parameters**								
	C_A^0	k_0^1	k_0^2	k_0^3	k_0^4	E^1	E^2	E^3	E^4
nominal	0.8	1	0.7	0.1	0.006	8.33	10	50	83.3
perturbed	0.8	1	0.7	0.1	0.006	7.33	9	60	93.3

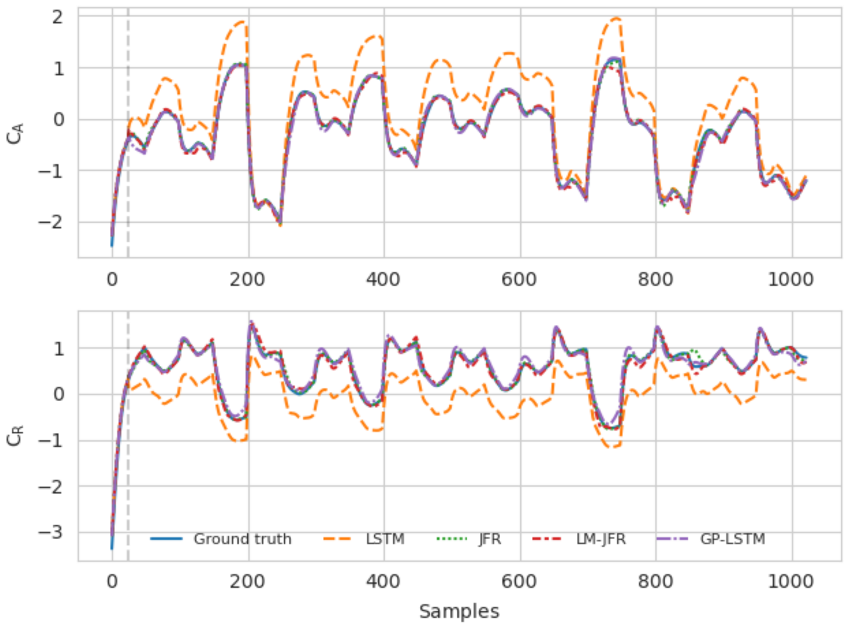

Figure 2: CSTR example. Normalized output concentrations C_A and C_R on the evaluation dataset: ground truth (blue); nominal LSTM (orange); Jacobian Feature Regression (green); Limited Memory Jacobian-Vector Product implementation (red); non-parametric GP-LSTM (purple). The gray vertical lines represent the context window consisting of the first 25 datapoints.

Table 2: CSTR example: Computational time of performing transfer for proposed methods JFR, LM-JFR, GP-LSTM.

Model	**Computational Time (s)**
JFR	13.24
LM-JFR	295.37
GP-LSTM	1321.05

By considering as state vector $x = [v_C \;\; i_L]^\top$ and input $u = v_{in}$, we adopt for this system the following nominal state-space model structure:

$$\dot{x} = \mathcal{F}(x, u; \theta)$$
$$y = v_C,$$

where $\mathcal{F}$ is a feed-forward neural network with three input units (corresponding to v_C, i_L, and v_{in}); a hidden layer with 64 linear units followed by tanh nonlinearity; and two linear output units corresponding to the components of the state equation to be learned.

The model above is discretized with the forward-Euler method at sampling time $T_s = 1\ \mu$s, resulting in a discrete-time RNN model. The parameters θ are then estimated on the training dataset using the truncated simulation error minimization method described in [5]. In particular, training is performed over 10000 iterations of gradient-based optimization on minibatches, each one containing 16 sequences of length 256 extracted from the training dataset.

Model adaptation is performed in the parameter-space setting, through the Bayesian Linear Regression approach discussed in the paper.

Table 3: CSTR example: R^2 performance index of nominal and adapted model on the two output channel C_A/C_R for training, test, transfer and evaluation datasets.

Model	**Dataset**			
	Train	Test	Transfer	Evaluation
nominal	0.99/0.99	0.99/0.99	0.74/0.08	0.50/-0.74
adapted	-	-	0.99/0.99	0.99/0.99

Table 4: RLC example: nominal and perturbed system coefficients.

System	**Parameters**		
	R (Ω)	L_0 (μH)	C (nF)
nominal	3	50	270
perturbed	4	50	350

Table 5: RLC example: R^2 performance index of nominal and adapted model for training, test, transfer and evaluation datasets.

Model	**Dataset**			
	Train	Test	Transfer	Eval
nominal	0.99	0.98	0.92	0.93
adapted	-	-	0.99	0.97

As shown in Table 5, the performance of the nominal model is excellent on both the training and the test datasets, but drops significantly on the transfer and evaluation datasets where the system is simulated in its perturbed configuration. On the other hand, the JFR model adapted on the transfer dataset achieves very high performance both on the transfer and on the evaluation datasets. Fig. 3 shows the (normalized) output voltage v_C on the evaluation dataset for: the nominal state-space network; the JFR model; and an adapted model iteratively estimated through an Extended Kalman Filter (EKF). The obtained results clearly show that the JFR approach proposed in the paper outperforms EKF.

Table 6: RLC example. (i) R^2 performance index on evaluation dataset achieved by: nominal model; Bayesian Linear Regression; Extended Kalman Filter; complete retraining from transfer dataset after full convergence and after 15 seconds of training; (ii) CPU time T required to compute adapted models through: Bayesian Linear Regression with naïve and sensitivity-based (eq. (14)) computation of the Jacobian; Extended Kalman Filter; full retraining.

L_o = 60 μH , C = 550 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2
4	0.80	0.95	13.52	0.95	0.98	0.68	538.80	0.98	158.02	0.07
7	0.73	0.94	13.14	0.94	0.97	0.83	542.21	0.99	158.32	0.20
10	0.66	0.91	13.06	0.91	0.99	0.85	548.64	0.99	176.42	0.15

L_o = 75 μH , C = 550 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2
4	0.73	0.90	13.88	0.90	1.01	0.63	550.62	0.97	144.50	0.12
7	0.67	0.90	13.22	0.90	1.01	0.74	544.81	0.98	140.43	0.19
10	0.61	0.88	12.57	0.88	0.97	0.72	537.06	0.98	159.24	0.24

L_o = 90 μH , C = 550 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2
4	0.66	0.85	12.90	0.85	0.97	-82.44	537.25	0.96	390.15	0.13
7	0.60	0.85	13.17	0.85	0.97	0.78	540.22	0.98	133.40	0.17
10	0.56	0.84	12.67	0.84	0.98	0.77	538.96	0.98	147.66	0.17

L_o = 60 μH , C = 1050 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T(s)	R^2
4	0.64	0.88	13.59	0.88	1.00	0.64	535.40	0.98	157.59	0.22
7	0.50	0.82	12.73	0.82	0.97	0.65	538.85	0.99	169.94	0.28
10	0.36	0.76	12.62	0.76	0.97	0.81	537.95	0.99	175.31	0.45

L_o = 75 μH , C = 1050 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2
4	0.57	0.82	13.07	0.82	0.97	0.76	540.73	0.98	390.67	0.22
7	0.45	0.78	12.62	0.78	0.99	0.76	537.78	0.98	148.42	0.32
10	0.32	0.73	12.71	0.73	0.98	0.75	543.11	0.99	166.83	0.47

L_o = 90 μH , C = 1050 nF

	Nominal	JFR with naïve Jacobian comp.		JFR with recursive Jacobian comp.		EKF		Retrain		
								full-convergence		$15s$
R (Ω)	R^2	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2	T (s)	R^2
4	0.52	0.77	13.27	0.77	0.97	0.80	539.87	0.97	391.41	0.22
7	0.41	0.74	13.51	0.74	0.96	0.78	536.80	0.98	133.35	0.37
10	0.23	0.69	13.26	0.69	0.96	0.81	538.65	0.98	153.58	0.47

Table 6 summarizes results of a more extensive validation of the proposed methodology both in terms of model performance and run time. In particular, resistance R, inductance L, and capacitance C are all varied with respect to their nominal values. We can observe a drop in the performance of the nominal model on the evaluation dataset with respect to the nominal performance reported in Table 5 on the test dataset. On the other hand, the proposed JFR approach for model adaptation achieves consistently better performance, with a degradation only for large nominal/perturbed system mismatch. It is worth noticing that JFR with recurrent Jacobian computation based on sensitivity equations (14)

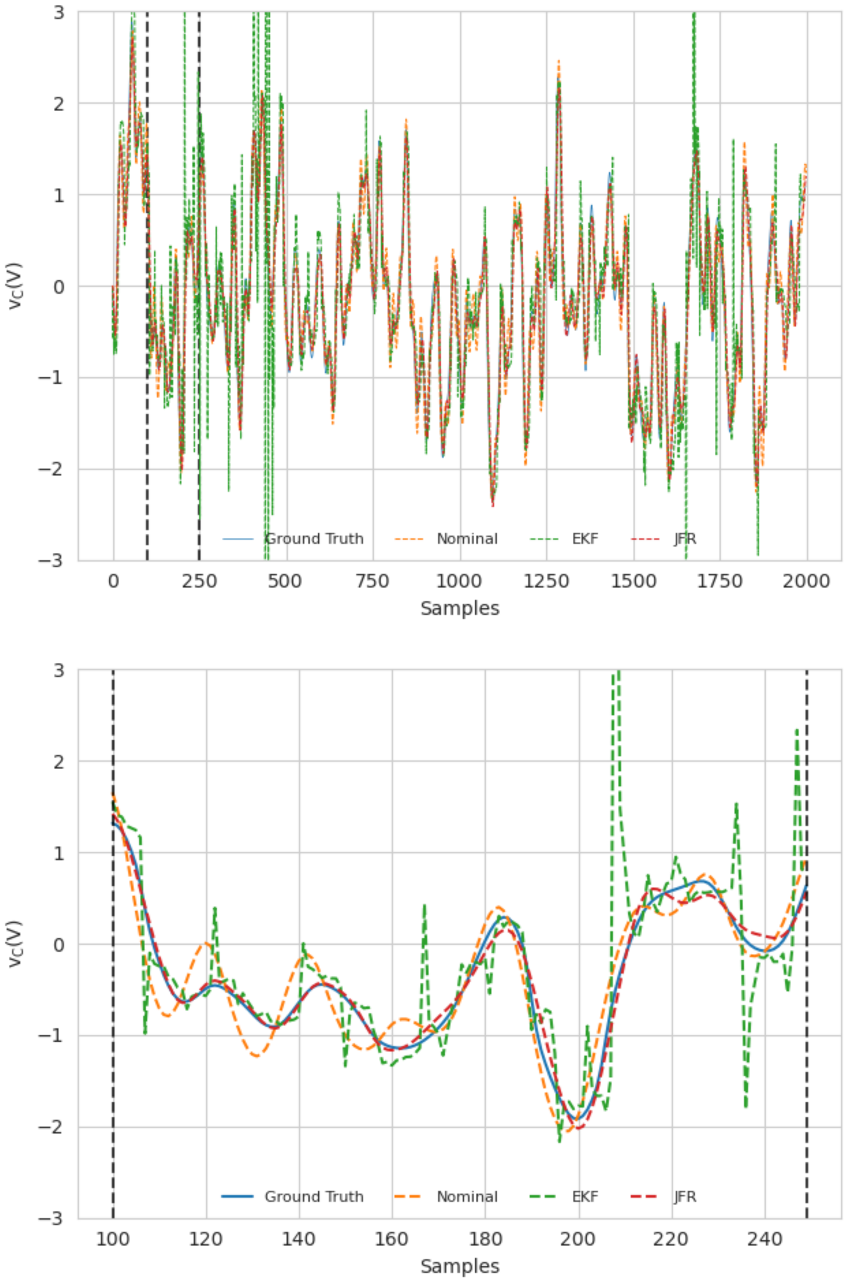

Figure 3: RLC example. **(Top)** Normalized output voltage v_C on the evaluation dataset: ground truth (blue), nominal state-space neural network [5] (orange); Jacobian Feature Regression (red); Extended Kalman filter (green). **(Bottom)** Zoom for samples between the dotted vertical lines (black) in the top figure.

is about 13x faster than the naïve implementation. Furthermore, the performance of JFR are consistently better than the ones achieved by the EKF, both in terms of prediction accuracy and run time.

Finally, results achieved by simply re-training a RNN model from the transfer dataset are also provided. To appreciate the computational efficiency of the JFR, we compare its performance with the one achieved by a model retrained for the same amount of time as it takes the JFR with a naïve computation of the Jacobian (∼15 seconds). We also report results obtained by retraining the RNN model until full convergence is achieved (either after 10000 iterations or when the loss is smaller than 1% of its initial value). Although the predictive performance of the re-trained RNN at full convergence is better than JFR, the run time is about 150x larger. Furthermore, by constraining the training time to 15 seconds, we observe a significant drop in the performance of the retrained model. This shows the potential advantage of our JFR approach in presence of time constraints, e.g., in adaptive control applications.

6 Conclusion

We have presented a transfer learning methodology for fast and efficient adaptation of Recurrent Neural Network models. The identified RNN is augmented by an additive linear-in-the-parameters correction term trained on fresh data using the model's Jacobian features with respect to its nominal parameters. The adaptation term is then obtained as the solution of a (non-singular) linear least-squares problem.

The proposed transfer learning approach is shown to compare favorably with respect to EKF in terms of predictive performance and run time. We also compare our methodology with a full re-identification of the RNN model from scratch. While in some cases full re-identification deliver superior predictive performance, our approach is significantly faster. For a limited computational budget, our approach is actually shown to outperform full plant re-identification also in terms of predictive performance. Furthermore, the optimal correction term is guaranteed to exist and to be unique, owing to the linear least-squares formulation.

Current investigations are devoted to the application of our transfer learning methodologies in adaptive control, where the fast run time, the possibility of online implementation, and the theoretical guarantees of the proposed algorithms may lead to significant advantages with respect to existing model updating techniques.

Acknowledgments

The activities of Marco Forgione and Dario Piga have been supported by HASLER STIFTUNG under the project DEALING: DEep learning for dynamicAL systems and dynamical systems for deep learnING.

References

[1] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[2] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[3] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[4] Gerben Beintema, Roland Toth, and Maarten Schoukens. Nonlinear state-space identification using deep encoder networks. In *Learning for Dynamics and Control*, pages 241–250. PMLR, 2021.

[5] M. Forgione and D. Piga. Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control*, 59:69–81, 2021.

[6] Carl Andersson, Antônio H Ribeiro, Koen Tiels, Niklas Wahlström, and Thomas B Schön. Deep convolutional networks in system identification. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 3670–3676. IEEE, 2019.

[7] Marco Forgione and Dario Piga. dynoNet: A neural network architecture for learning dynamical systems. *International Journal of Adaptive Control and Signal Processing*, 35(4), 2021.

[8] Lucian Cristian Iacob, Gerben Izaak Beintema, Maarten Schoukens, and Roland Tóth. Deep Identification of Nonlinear Systems in Koopman form. *arXiv preprint arXiv:2110.02583*, 2021.

[9] Alexandre Mauroy, Igor Mezić, and Yoshihiko Susuki. *The Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*, volume 484. Springer Nature, 2020.

[10] Bojan Mavkov, Marco Forgione, and Dario Piga. Integrated neural networks for nonlinear continuous-time system identification. *IEEE Control Systems Letters*, 4(4):851–856, 2020.

[11] Simone Pozzoli, Marco Gallieri, and Riccardo Scattolini. Tustin neural networks: a class of recurrent nets for adaptive mpc of mechanical systems. *IFAC-PapersOnLine*, 53(2):5171–5176, 2020.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[13] Lennart Ljung. *System Identification: Theory for the User*. Prentice-Hall, Inc., USA, 1986.

[14] Wesley Maddox, Shuai Tang, Pablo Moreno, Andrew Gordon Wilson, and Andreas Damianou. Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2737–2745. PMLR, 2021.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Jan Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2000.

[17] Luca Bugliari Armenio, Enrico Terzi, Marcello Farina, and Riccardo Scattolini. Echo state networks: analysis, training and predictive control. In *2019 18th European Control Conference (ECC)*. IEEE, June 2019.

[18] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[19] Randall Balestriero and Richard Baraniuk. Fast Jacobian-Vector product for Deep Networks. *arXiv preprint arXiv:2104.00219*, 2021.

[20] Herschel Rabitz, Mark Kramer, and D Dacol. Sensitivity analysis in chemical kinetics. *Annual review of physical chemistry*, 34(1):419–461, 1983.

[21] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[22] Lennart Ljung, editor. *System Identification: Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 edition, 1999.

[23] *Chemical Reactor Design and Control*, chapter 2, pages 31–106. John Wiley & Sons, Ltd, 2007.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[25] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017. arXiv: 1503.04069.