An Inexact Sequential Quadratic Programming Method for Learning and Control of Recurrent Neural Networks

Adeyemi D. Adeoye and Alberto Bemporad, *Fellow, IEEE*

***Abstract*— This article considers the two-stage approach to solving a partially observable Markov decision process (POMDP): the *identification stage* and the (optimal) *control stage*. We present an inexact sequential quadratic programming framework for recurrent neural network learning (iSQPRL) for solving the identification stage of the POMDP, in which the true system is approximated by a recurrent neural network (RNN) with dynamically consistent overshooting (DCRNN). We formulate the learning problem as a constrained optimization problem and study the quadratic programming (QP) subproblem with a convergence analysis under a restarted Krylov-subspace iterative scheme that implicitly exploits the structure of the associated Karush–Kuhn–Tucker (KKT) subsystem. In the control stage, where a feedforward neural network (FNN) controller is designed on top of the RNN model, we adapt a generalized Gauss–Newton (GGN) algorithm that exploits useful approximations to the curvature terms of the training data and selects its mini-batch step size using a known property of some regularization function. Simulation results are provided to demonstrate the effectiveness of our approach.**

***Index Terms*— Gauss–Newton methods, Markov decision processes, numerical optimization, recurrent neural networks (RNNs), reinforcement learning (RL), sequential quadratic programming (SQP).**

I. Introduction

REINFORCEMENT learning (RL) or approximate dynamic programming (ADP) involves an *agent* or *decision maker* that interacts with its *environment* based on the *states* of the environment observable by the agent [1], [2]. For each decision that the agent makes, it incurs a cost or, alternatively, gets a reward. Hence, the ultimate goal of the agent is to minimize the total costs, or maximize the sum of its rewards, received from the environment. In most cases, it is assumed that the operations of the agent are in accordance with a finite discrete-time *Markovian decision process* (MDP), which, in turn, assumes that the states of the environment are fully observable by the agent at any given time t, providing all necessary information that the agent can use to decide what action to take. This is not always the case in reality, especially as the environment is mostly complex and its model is unknown. Hence, many practical RL problems are solved within the framework of partially observable Markov decision process (POMDP). In line with control problems, the environment is presented as the state-space model of a dynamical system that encodes the spatial and temporal information about the system, which can be used to predict its future behavior. With the popularity of deep learning methods [3], the paradigm of world model [4] has been recently used to describe RL methods in which a generative and/or predictive model is trained to represent the agent's own imagination of the environment. A control neural network which the agent uses to make decisions and takes actions online is thereafter trained offline on top of the representative model. This approach encompasses an idea that dates back as far as the 1990s where mostly recurrent neural networks (RNNs) are trained to build a predictive model of the environment or complex system [5], [6], [7], [8], helping to possibly develop a meaningful representation of the *Markovian state space* in the face of *partial observability* [8]. Data efficiency becomes an added advantage in the design of the representative RNN model, which is a key element of model-based RL and control methods [2]. These so-called *recurrent control networks* [7] are mostly designed with much focus on their architecture that enable them to capture relevant information about the underlying system dynamics. Schaefer et al. [8] introduce the recurrent control neural network (RCNN) with an architectural design motivated by the two stages of a typical neural network-based complex control method, namely: 1) a system identification stage where an RNN with dynamically consistent overshooting (DCRNN) [9] is designed to train a state-space model of the system using process measurements and 2) an optimal control (OC) policy selection stage where a feedforward neural network (FNN) is concatenated with the RNN to learn an OC policy, which is later used to control the real process.

This unified approach largely benefits from the approximation power of the individual neural networks when trained with an appropriate gradient-based algorithm. Besides their slow convergence, a drawback in learning RNNs for practical applications with first-order gradient-based algorithms is the *possible* problem of *vanishing and exploding gradients* [10], [11], which may lead to their difficulty in learning long-term dependencies, as their operations rely only upon the slope of the loss surface in the Jacobian. The second-order methods, however, can be used to mitigate these drawbacks by utilizing, in their learning procedures, the information contained in the curvature of the loss surface, which provably accelerates convergence. The second-order methods for learning RNNs have historically been used in two ways: 1) the use of

Manuscript received 20 October 2022; revised 21 June 2023 and 7 November 2023; accepted 12 January 2024. Date of publication 31 January 2024; date of current version 6 February 2025. *(Corresponding author: Adeyemi D. Adeoye.)*

The authors are with the IMT School for Advanced Studies Lucca, 55100 Lucca, Italy (e-mail: adeyemi.adeoye@imtlucca.it).

Digital Object Identifier 10.1109/TNNLS.2024.3354855

second-order optimization algorithms, such as generalized Gauss–Newton (GGN), Levenberg–Marquardt (LM), and conjugate gradient (CG) algorithms (see [12], [13], [14], [15], [16], [17], [18], [19]) and 2) using nonlinear sequential state-estimation techniques, such as extended Kalman filter (EKF) method (see [20], [21], [22], [23], [24], [25]). In the first approach, the second-order information is captured through Hessian (or approximate Hessian) computations, while in the second approach, the second-order information is computed recursively as a prediction-error covariance matrix. In any event, these approaches provide ways to capture relevant second-order information about the training loss function as well as help to curtail the possible vanishing/exploding-gradient problem. As the strength of data-based RL and control methods lies in their data efficiency, it is further desirable to capture curvature information about the neural network training data for better approximation and representation capability within the allotted training time. In control applications, recovering a good representation of the underlying system is very important for robustness and reliability. For this reason, many recent works have focused on designing custom training algorithms for their application-specific RNN structures (see the recent works [17], [26], [27], [28], [29]).

Although we discuss the training problem of RCNNs in accordance with the two stages involved, our main focus is on the first stage of the training problem where training a DCRNN is viewed as a constrained optimization problem. While this viewpoint helps us to develop alternative neural network training algorithms, it can also provide tools to derive useful convergence results for the algorithm [30]. In this work, we establish an inexact sequential quadratic programming (SQP) framework for the DCRNN training problem where the quadratic programming (QP) subproblems are solved by a restarted Krylov-subspace iterative scheme that implicitly exploits the structure of the associated Karush–Kuhn–Tucker (KKT) subsystems. As the resulting algorithm is based on the restarted generalized minimal residual (GMRES) Krylov-subspace method, we call it GMRES recurrent learning (GRL) algorithm, or GRL($\hat{m}_r$) to include a given restart parameter $\hat{m}_r$. The inexact sequential QP for recurrent neural network learning (iSQPRL) framework allows us to use sparse iterative methods that exploit the structure of subproblems for fast convergence and to achieve a control- and data-efficient model of the DCRNN. As a back-propagation-through-time (BPTT) algorithm for training RNNs [31], GRL($\hat{m}_r$) also addresses one of the main shortcomings associated with the class of BPTT algorithms, viz., requiring an excessive number of iterations to converge to the true solution, largely increasing the complexity of the algorithm [32]. In the second stage—where we learn an OC law—we simply adapt the GGN with self-concordant regularization (GGN-SCORE) algorithm, established for convex optimization problems in [33]. Because the GGN-SCORE algorithm was established for curvature exploiting, this also improves our overall model-based control decision approach for data efficiency.

The rest of this article is organized as follows. In Section II, we present the state-space RCNN model and the optimization problem involved in each of the two training stages. In Section III, we introduce the SQP framework for the DCRNN learning problem of the first stage and present details of our proposed learning algorithm, which we call GRL($\hat{m}_r$), an algorithm that uses a nonmonotone line search for its globalization (see Algorithm 1). In Section IV, we present the curvature-exploiting GGN-SCORE algorithm for training the control network of the second stage. We analyze the complexity of the proposed algorithm in Section V. Numerical examples in which we demonstrate our approach is presented in Section VI, and a concluding remark is given in Section VII.

II. Recurrent Control Neural Networks

The RCNN training problem involves two stages: 1) identify a DCRNN model and 2) train an FNN on the DCRNN model for OC policy selection. In accordance with this, we first formulate the problem of training an RNN (and DCRNN). Then, we present the RCNN and formulate its training problem on top of an extension of the DCRNN model.

A. RNN for State Reconstruction in RL and Control

Suppose that the state transition and observables of the environment or system process can be described by a nonlinear open-loop dynamical system, for which a model is assumed to be unknown. Let $\{u_0, u_1, \ldots, u_{N-1}\}$ be a sequence of inputs to the dynamical system and $\{y_0, y_1, \ldots, y_{N-1}\}$ the corresponding sequence of observable outputs, with $u_t \in \mathbb{R}^{n_u}$, $y_t \in \mathbb{R}^{n_y}$. System identification is performed by training an RNN model of the form

$$x_{t+1} = f_x(x_t, y_t, u_t, \theta_x) \tag{1a}$$

$$\hat{y}_t = f_y(x_t, \theta_y) \tag{1b}$$

where f_x and f_y describe the Markovian state-space dynamics and are, respectively, parameterized by the vectorized weight/bias terms $\theta_x \in \mathbb{R}^{n_{\theta_x}}$ and $\theta_y \in \mathbb{R}^{n_{\theta_y}}$, $x_t \in \mathbb{R}^{n_x}$ is the RNN hidden state, and $\hat{y}_t \in \mathbb{R}^{n_y}$ is its prediction of the system observable at time t. In the identification task, the RNN is trained to encode meaningful information about the true system in its hidden states x_t. In addition to passing the observables y_t as external inputs to the state-update function (1a), they are also given as the targets for the RNN predictions $\hat{y}_t$. In this work, we propose to learn x_t and the parameter vectors θ_x, θ_y in parallel in the context described in [22], that is, in a way analogous to *direct multiple shooting* approach to OC problems. This is achieved by formulating the learning task as the equality-constrained problem

$$\min_z \quad f(z) := \mathcal{R}(x_0, \theta_x, \theta_y) + \sum_{t=0}^{N-1} \ell(y_t, \hat{y}_t) \tag{2a}$$

$$\text{s.t.} \quad c(z) := \begin{bmatrix} x_1 - f_x(x_0, y_0, u_0, \theta_x) \\ \vdots \\ x_{N-1} - f_x(x_{N-2}, y_{N-2}, u_{N-2}, \theta_x) \\ c_N(x_0, f_x(x_{N-1}, y_{N-1}, u_{N-1}, \theta_x)) \end{bmatrix} = 0 \tag{2b}$$

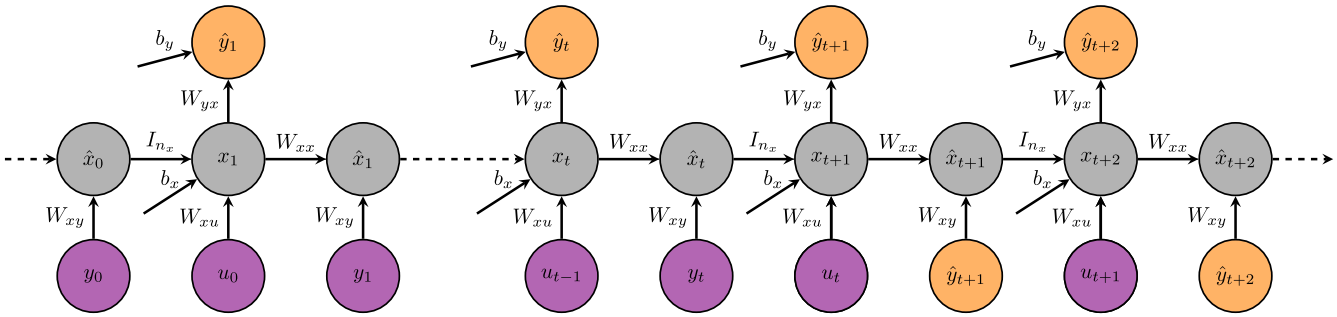


Fig. 1. Unrolled DCRNN.

where $f\colon \mathbb{R}^n \to \mathbb{R}$, $c\colon \mathbb{R}^n \to \mathbb{R}^m$, $z := [x_1^\top \cdots x_{N-1}^\top x_0^\top \theta_y^\top \theta_x^\top]^\top \equiv [z_1\, z_2 \cdots z_n]^\top$, $n = m + n_{\theta_y} + n_{\theta_x}, m = Nn_x$, $\ell\colon \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to \mathbb{R}$ is a loss function, and $\mathcal{R}$ is a regularization term. Both ℓ and $\mathcal{R}$ are assumed to be twice continuously differentiable with respect to their arguments. The last constraint in (2b) represents possible constraints, such as periodicity, $x_0 - f_x(x_{N-1}, y_{N-1}, u_{N-1}, \theta_x) = 0$ [34], or fixed terminal state constraint, where $c_N\colon \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$.

B. Extension to DCRNNs

Since now on, we will consider the special structure of (1) given by

$$x_{t+1} = \sigma_x(W_{xu}u_t + W_{xy}y_t + W_{xx}x_t + b_x) \tag{3a}$$
$$\hat{y}_t = W_{yx}x_t + b_y \tag{3b}$$

where $\sigma_x(\cdot)$ is an elementwise activation function. Let $\theta_x := \text{vec}([W_{xu}\, W_{xy}\, W_{xx}\, b_x])$ and $\theta_y := \text{vec}([W_{yx}\, b_y])$, where $W_{xu} \in \mathbb{R}^{n_u \times n_x}$, $W_{xy} \in \mathbb{R}^{n_x \times n_y}$, $W_{xx} \in \mathbb{R}^{n_x \times n_x}$, and $W_{yx} \in \mathbb{R}^{n_y \times n_x}$ are the RNN weight matrices, and $b_x \in \mathbb{R}^{n_x}$, $b_y \in \mathbb{R}^{n_y}$ are the bias vectors associated with model (3). Formulation (2) does not minimize a pure open-loop simulation error, in that y_t enters the state-update equation rather than $\hat{y}_t$. To handle open-loop simulation terms, we approximate the recurrence of the RNN dynamics by truncating its *unrolling* into the past at a finite time step $t = m_-$. The DCRNN contains in modeling an internal dynamics by *overshooting* the network dynamics in the sense that, we take $m_+ > 1$ overshooting time steps into the future where the network uses its own predictions as the future external inputs [9] (see Fig. 1). Still, in the overshooting part, the network gets the system inputs u_t, as they also influence the network dynamics. This phenomenon results in the following state-space equations of the DCRNN dynamics:

$$x_{t+1} = \sigma_x(I\hat{x}_t + W_{xu}u_t + b_x) \tag{4a}$$
$$\hat{y}_t = W_{yx}x_t + b_y \tag{4b}$$
$$\text{with}\quad \hat{x}_t = \begin{cases} W_{xx}x_t + W_{xy}y_t & \forall 0 \le t \le m_- \\ W_{xx}x_t + W_{xy}\hat{y}_t & \forall m_- < t \le N \end{cases} \tag{4c}$$

where $I \in \mathbb{R}^{n_x \times n_x}$ is an identity matrix. Letting $\ell = (y_t - \hat{y}_t)^2$, $c(z) := [c_0, c_1, \ldots, c_N] \in \mathbb{R}^m$, and $N = m_- + m_+$, the corresponding optimization problem is

$$\min_z \quad f(z) := \mathcal{R}(x_0, \theta_x, \theta_y) + \sum_{t=0}^{N-1} (y_t - \hat{y}_t)^2 \tag{5a}$$
$$\text{s.t.} \quad c_t = 0 \quad \forall t = 0, \ldots, N \tag{5b}$$
$$\text{with} \quad c_t := x_{t+1} - \sigma_x(I\hat{x}_t + W_{xu}u_t + b_x) \tag{5c}$$
$$\hat{x}_t = \begin{cases} W_{xx}x_t + W_{xy}y_t & \forall 0 \le t \le m_- \\ W_{xx}x_t + W_{xy}\hat{y}_t & \forall m_- < t \le N-1. \end{cases} \tag{5d}$$

C. FNN for OC Policy Selection

Given a training dataset $\{u_t, y_t\}, t = 0, \ldots, N-1$, after computing the optimal values of the parameter vectors θ_x and θ_y, we construct an FNN on top of the DCRNN to obtain an RCNN (see Fig. 2). In this work, a three-layer FNN model is considered to obtain new OC inputs $\hat{u}_t$, $\forall t > m_-$. The resulting control network together with the DCRNN forms the following RCNN represented by their state-space equations:

$$\hat{u}_t = \sigma_u(V_{uh}\sigma_h(V_{hx}\hat{x}_t + b_h) + b_u) \quad \forall m_- \le t \le N \tag{6a}$$

$$x_{t+1} = \begin{cases} \sigma_x(I\hat{x}_t + W_{xu}u_t + b_x) & \forall 0 \le t < m_- \\ \sigma_x(I\hat{x}_t + W_{xu}\hat{u}_t + b_x) & \forall m_- < t \le N \end{cases} \tag{6b}$$

$$R_t = G_r\sigma_r(W_{yx}x_t + b_y) \quad \forall m_- < t \le N \tag{6c}$$

$$\text{with} \quad \hat{x}_t = \begin{cases} W_{xx}x_t + W_{xy}y_t, & 0 \le t \le m_- \\ W_{xx}x_t + W_{xy}\hat{y}_t, & m_- < t \le N \end{cases} \tag{6d}$$

where $V_{uh} \in \mathbb{R}^{n_u \times n_h}$, $V_{hx} \in \mathbb{R}^{n_h \times n_x}$, $b_h \in \mathbb{R}^{n_h}$, and $b_u \in \mathbb{R}^{n_u}$ are the control network parameters to be learned, and σ_u and σ_h are elementwise activation functions. G_r is a constant, problem-specific matrix, chosen such that with an appropriate choice of the function σ_r, the network output map $R_t(x_t, \hat{y}_t; \hat{u}_t)$ models the problem's reward function. Here, the optimization problem to solve is

$$\max_{\theta_u} \quad \hat{f}(\theta_u) := \sum_{t=m_-}^{N-1} R_t \tag{7}$$

where $\theta_u := \text{vec}([V_{uh}\, V_{hx}\, b_h\, b_u]) \in \mathbb{R}^{n_{\theta_u}}$.

III. Sequential QP for Recurrent Learning

In the following, we describe the iSQPRL framework, which we use to construct an algorithm to solve the DCRNN

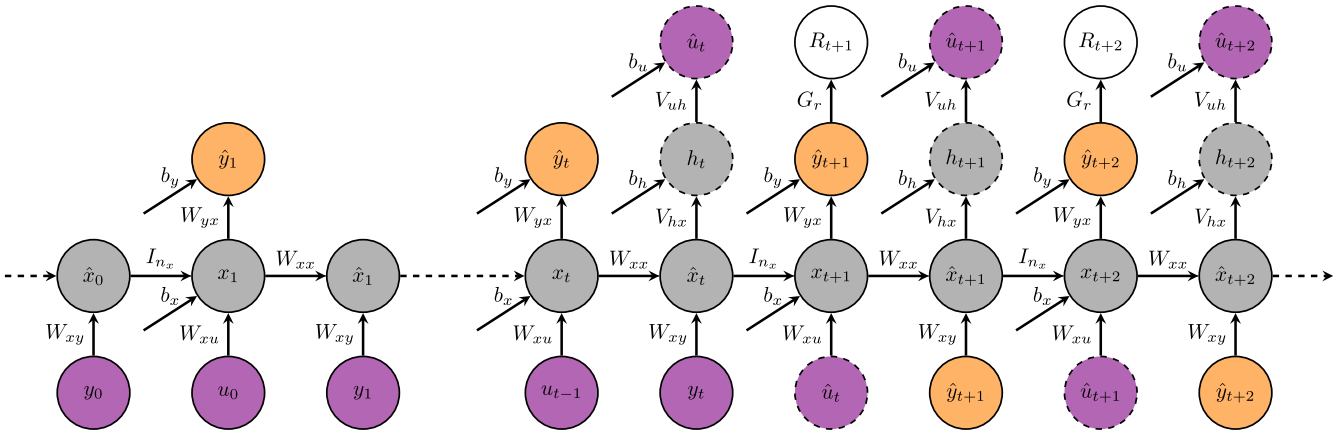


Fig. 2. Architecture of RCNNs.

learning problem. Consider again problem (5) and define the Lagrangian

$$\mathcal{L}(z,\lambda) := f(z) - \lambda^\top c(z) \tag{8}$$

where $\lambda \in \mathbb{R}^m$ is the vector of Lagrange multipliers. The SQP approach for solving problem (5) involves iteratively solving the QP subproblem

$$d_k^z \in \arg\min_{d_z} \quad \frac{1}{2} d_z^\top H(z_k) d_z + \nabla f(z_k)^\top d_z \tag{9a}$$

$$\text{s.t.} \quad \nabla c(z_k) d_z + c(z_k) = 0 \tag{9b}$$

at a given approximate solution z_k, $k = 1, 2, \ldots$, where $\nabla c(z) = [\nabla c_0(z) \cdots \nabla c_{N-1}(z)]^\top \in \mathbb{R}^{m \times n}$ and $H(z_k)$ is the Hessian $\nabla^2 \mathcal{L}(z_k, \lambda_k) := \nabla^2 f(z_k) + \nabla(\nabla c(z_k)^\top \lambda_k)$ of the Lagrangian at step k, or an approximation to it. Suppose at iteration k, the QP solver finds an optimal multiplier $\tilde{\lambda}_k$ (which can change from iteration to iteration). Then, by setting

$$d_k^\lambda = \tilde{\lambda}_k - \lambda_k \tag{10}$$

we update (z, λ) using

$$z_{k+1} = z_k + \alpha_k d_k^z, \quad \lambda_{k+1} = \lambda_k + \alpha_k d_k^\lambda \tag{11}$$

where α_k is a carefully chosen step length or *learning rate*. This iterative process creates a sequence $\{z_k\}$ of approximations, which should be made to converge to a solution z^* of (5). Next, we consider the equivalent Newton viewpoint of the QP subproblem (9), to simplify our analysis of the proposed solution technique.

The necessary (KKT) optimality conditions for the QP subproblem at any step k are given by the nonlinear equation

$$\tilde{\mathcal{F}}(z_k, \lambda_k) := \begin{bmatrix} \nabla f(z_k) - \nabla c(z_k)^\top \lambda_k \\ c(z_k) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{12}$$

For simplicity of notation, we denote by $\tilde{\mathcal{F}}_k := \tilde{\mathcal{F}}(z_k, \lambda_k)$ and similarly for other functional quantities, and set $J_k := \nabla c_k$ and $g_k := \nabla f_k$. By applying Newton's method to (12), we obtain *Newton's equation*

$$\tilde{A}_k \tilde{d}_k = -\tilde{\mathcal{F}}_k, \quad \tilde{A}_k = \nabla \tilde{\mathcal{F}}_k = \begin{bmatrix} H_k & -J_k^\top \\ J_k & 0 \end{bmatrix}, \quad \tilde{d}_k = \begin{bmatrix} d_k^z \\ d_k^\lambda \end{bmatrix}. \tag{13}$$

By treating λ_{k+1} as an unknown and by letting $\mathcal{F}_k := [g_k, c_k]^\top$, $b_k := -\mathcal{F}_k$, we equivalently write (13) as the saddle-point system

$$\underbrace{\begin{bmatrix} H_k & J_k^\top \\ J_k & 0 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} d_k^z \\ -\tilde{\lambda}_k \end{bmatrix}}_{d_k} = \underbrace{\begin{bmatrix} -g_k \\ -c_k \end{bmatrix}}_{b_k}. \tag{14}$$

The solution d_k to (14) constitutes the Newton direction at step k. Depending on the choice of H_k, say $H_k = \nabla^2 \mathcal{L}_k$, and by setting d_k^λ as in (10), we choose an appropriate value for the step-length parameter α_k, say $\alpha_k = 1$, and take the step (11). Once the system in (14) is solved, $\tilde{A}_k$ and $\tilde{d}_k$ satisfying (13) can then be obtained accordingly.

A. Approximating the Lagrangian Hessian

Clearly, the choice $H_k = \nabla^2 \mathcal{L}_k$ in (14) would be computationally prohibitive for the DCRNN training problem in most cases. Besides, we often need H_k to be positive-definite in order to ensure that the QP subproblem solved at each step is convex, thereby preserving the convergence properties provided by the primal step direction. However, $\nabla^2 \mathcal{L}_k$ is not positive-definite in general. In this work, we reduce these computational difficulties by approximating $\nabla^2 \mathcal{L}_k$ by the modified Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula [35], [36], [37], [38] presented in [39].

Starting from an initial positive-definite matrix H_0, the modified BFGS scheme seeks an approximation H_k to the true Hessian $\nabla^2 \mathcal{L}_k$ as follows. At step k, define

$$\bar{\gamma}_k = \theta_k \gamma_k + (1 - \theta_k) H_k \delta_k, \quad 0 \le \theta_k \le 1 \tag{15}$$

where $\gamma_k = \nabla \mathcal{L}_{k+1} - \nabla \mathcal{L}(z_k, \lambda_{k+1})$ and $\delta_k := \alpha_k d_k^z = z_{k+1} - z_k$. Vector $\bar{\gamma}_k$ is geometrically the closest to γ_k, such that the inequality

$$\delta_k^\top \bar{\gamma}_k \ge \eta \delta_k^\top H_k \delta_k \tag{16}$$

is satisfied, where the parameter η is chosen empirically and typically takes the value 0.2 [39], [40], [41], which is just sufficient for our approximations. The key idea is to gradually approach the true Hessian while accounting for the curvature information captured during the most recent step.

Consequently, the convexity parameter θ_k in (15) takes the value

$$\theta_k = \begin{cases} 1, & \text{if } \delta_k^\top \gamma_k \geq \eta \delta_k^\top H_k \delta_k \\ \dfrac{(1-\eta)\delta_k^\top H_k \delta_k}{\delta_k^\top H_k \delta_k - \delta_k^\top \gamma_k}, & \text{otherwise} \end{cases} \tag{17}$$

and a rank-two matrix U_k defined by

$$U_k = \frac{\bar{\gamma}_k \bar{\gamma}_k^\top}{\delta_k^\top \bar{\gamma}_k} - \frac{H_k \delta_k \delta_k^\top H_k}{\delta_k^\top H_k \delta_k} \tag{18}$$

is selected to update H_k as

$$H_{k+1} = H_k + U_k \tag{19}$$

which maintains positive-definiteness of H_k.

B. Numerical Solution of the Saddle-Point System

As the linear system (14) can involve a large number of variables, it becomes natural to present a computationally efficient technique for solving it. In this work, we do not wish to impose restrictive assumptions on iSQPRL as we know that the matrix A_k in (14) is not positive-definite. Hence, we present a solution approach to the system in a general learning framework. The GMRES algorithm [42], which we briefly describe next, is our method of choice for this purpose. Starting from an arbitrary initial guess $d_{k,0} \in \mathbb{R}^{\hat{n}}$, $\hat{n} = n + m$, define the initial residual $r_{k,0} := b_k - A_k d_{k,0}$, and let

$$\mathcal{K}_{\hat{m}}(A_k, r_{k,0}) := \text{span}\left\{r_{k,0}, A_k r_{k,0}, \ldots, A_k^{\hat{m}-1} r_{k,0}\right\} \tag{20}$$

be the $\hat{m}$-dimensional Krylov subspace generated by A_k and $r_{k,0}$. At the $\hat{m}$-th step, GMRES finds an approximation $d_{k,\hat{m}} \in \mathcal{K}_{\hat{m}}$ to the true solution $d_k = A_k^{-1} b_k$. This approximate solution is the value of d_k that minimizes the norm of the residual $r_{k,\hat{m}} = b_k - A_k d_{k,\hat{m}}$. In its implementation, GMRES does not explicitly form the vectors $r_{k,0}, A_k r_{k,0}, A_k^2 r_{k,0}, \ldots, A_k^{\hat{m}-1} r_{k,0}$, as they may be close to being linearly dependent; instead, it uses the Arnoldi iteration to form an orthonormal basis for $\mathcal{K}_{\hat{m}}(A_k, r_{k,0})$. In particular, if $q_{k,1}, q_{k,2}, \ldots, q_{k,\hat{m}}$ are the orthonormal vectors formed by the Arnoldi iteration, and if these vectors form the matrix $Q_{k,\hat{m}} \in \mathbb{R}^{\hat{n} \times \hat{m}}$, then we can write $d_{k,\hat{m}}$ as $d_{k,\hat{m}} = d_{k,0} + Q_{k,\hat{m}} s_{k,\hat{m}}$, $s_{k,\hat{m}} \in \mathbb{R}^{\hat{m}}$.

Let $\tilde{H}_{k,\hat{m}} \in \mathbb{R}^{(\hat{m}+1)\times \hat{m}}$ be the upper Hessenberg matrix generated from the (modified) Gram–Schmidt orthogonalization step of Arnoldi iteration satisfying

$$A_k Q_{k,\hat{m}} = Q_{k,\hat{m}+1} \tilde{H}_{k,\hat{m}}.$$

Let $\beta_k := \|r_{k,0}\|$. Then, $Q_{k,\hat{m}+1} e_1 = q_{k,1} = \|r_{k,0}\|^{-1} r_{k,0}$, where $e_1 := [1\, 0\, 0 \cdots 0]^\top \in \mathbb{R}^{\hat{m}+1}$, and

$$\begin{aligned} r_{k,\hat{m}} &= b_k - A_k d_{k,\hat{m}} \\ &= b_k - A_k (d_{k,0} + Q_{k,\hat{m}} s_{k,\hat{m}}) \\ &= \beta_k q_{k,1} - Q_{k,\hat{m}+1} \tilde{H}_{k,\hat{m}} s_{k,\hat{m}} \\ &= Q_{k,\hat{m}+1} (\beta_k e_1 - \tilde{H}_{k,\hat{m}} s_{k,\hat{m}}). \end{aligned}$$

As $Q_{k,\hat{m}+1}$ has orthonormal columns, we get that

$$\|r_{k,\hat{m}}\| = \|\beta_k e_1 - \tilde{H}_{k,\hat{m}} s_{k,\hat{m}}\|.$$

Therefore, one finds $d_{k,\hat{m}} \equiv d_k$ by solving

$$\min_{s \in \mathbb{R}^{\hat{m}}} \|\beta_k e_1 - \tilde{H}_{k,\hat{m}} s\| \tag{21}$$

whose solution can be obtained efficiently by using the QR factorization of $\tilde{H}_{k,\hat{m}}$, which, in turn, can be cheaply updated from iteration to iteration by exploiting its structure.

While GMRES solves a general linear system, the restarted variant of it, written GMRES($\hat{m}_r$), provides a better convergence speed and helps to curtail the memory issue linked to the storage of the orthonormal vectors formed in the former. Instead of allowing the storage of the entire Krylov subspaces whose size grows quadratically with the number $\hat{m}$ of steps, GMRES($\hat{m}_r$) restricts the dimension of the Krylov subspace to a fixed integer parameter $\hat{m}_r$ and restarts the Arnoldi process using the current approximate solution $d_{k,\hat{m}_r}$ as the new initial guess [42]. As the GMRES($\hat{m}_r$) method only computes an approximate solution d_k of the SQP subproblem, it forms an *inexact SQP method* (see [43]). Hence, we consider (14) with the residual vector $r_k \equiv r_{k,\hat{m}_r} =: (r_k^z, r_k^\lambda)$ that accounts for the error due to this inexactness

$$\begin{bmatrix} H_k & J_k^\top \\ J_k & 0 \end{bmatrix} \begin{bmatrix} d_k^z \\ -\lambda_k \end{bmatrix} = \begin{bmatrix} -g_k \\ -c_k \end{bmatrix} + \begin{bmatrix} r_k^z \\ r_k^\lambda \end{bmatrix}. \tag{22}$$

The addition of this residual vector is useful for deriving relevant bounds on important terms of the problem, as we shall see. In specific terms, the residual vector characterizes a part of the globalization strategy (Steps 9–13 of Algorithm 1) described in Section III-C. As a first step, we now state a result, established in [42], on the convergence of GMRES($\hat{m}_r$), that provides an upper bound on the residual norm of the algorithm.

Proposition 1 ([42], Proposition 4, and Theorem 5): Suppose that A_k is diagonalizable, so that $A_k = P_k D_k P_k^{-1}$, where D_k is the diagonal matrix of eigenvalues of A_k. Let τ be the number of distinct eigenvalues $\varrho_1, \varrho_2, \ldots, \varrho_\tau$ of A_k with nonpositive real parts, and let the other eigenvalues be enclosed in $C_\Theta(\Omega)$, a circle of radius Θ centered at Ω with $\Omega > \Theta > 0$. Then, the residual norm of GMRES($\hat{m}_r$) satisfies

$$\|r_{k,\hat{m}_r}\| \leq \kappa(P_k) \xi_k \|r_{k,0}\| \tag{23}$$

where

$$\kappa(P_k) = \|P_k\| \|P_k^{-1}\|, \quad \xi_k = \left(\frac{\bar{\alpha}}{\bar{\beta}}\right)^\tau \left(\frac{\Theta}{\Omega}\right)^{\hat{m}_r - \tau},$$
$$\bar{\alpha} = \max_{\substack{1 \leq i \leq \tau \\ \tau+1 \leq j \leq \hat{n}}} |\varrho_t - \varrho_j| \quad \text{and} \quad \bar{\beta} = \min_{1 \leq i \leq \tau} |\varrho_t|.$$

C. Globalization of iSQPRL by a Line Search

Many known inexact SQP approaches rely upon a line search or trust region method for global convergence. For the sake of completeness, we discuss these globalization concerns for iSQPRL with a line-search strategy, and with specific reference to the GMRES($\hat{m}_r$) iterative scheme considered in this work. In order to do this, we assume that the sequence $\{H_k\}$ is obtained through the modified BFGS formula described in Section III-A. We also assume that the value of $\hat{m}_r$ is sufficient

to ensure convergence of GMRES($\hat{m}_r$), and that the sequence of iterates $\{z_k, \lambda_k\}$ is generated by the described iSQPRL framework. Furthermore, the following conditions hold (using $\mathcal{L}^*$ to denote $\mathcal{L}(z^*, \lambda^*)$).

- *A.1:* $\{z_k, \lambda_k\}$ is contained in a closed, bounded, convex set S, on which the functions f and c are twice continuously differentiable.
- *A.2:* The columns of J_k are linearly independent, for each k.
- *A.3:* The matrices H_k are uniformly positive-definite on the null spaces of J_k, that is, $\exists \sigma_1 > 0$, such that for each k, $d^\top H_k d \geq \sigma_1 \|d\|^2$ for all $d \in \mathbb{R}^{\hat{n}}$ satisfying $J_k d = 0$.
- *A.4:* The sequence $\{H_k\}$ is bounded in norm, that is, $\exists \sigma_2 > 0$, such that for each k, $\|H_k\| \leq \sigma_2$.
- *A.5:* The matrices H_k have inverses that are bounded in norm, that is, $\exists \sigma_3 > 0$, such that for each k, H_k^{-1} exists and $\|H_k^{-1}\| \leq \sigma_3$.
- *A.6:* $\nabla^2 \mathcal{L}_k$ is Lipschitz continuous for each k in the neighborhood of (z^*, λ^*).
- *A.7:* The primal-dual step d_k *locally* satisfies

$$\lim_{k\to\infty} \frac{\|(P_k(H_k - \nabla^2 \mathcal{L}^*)P_k)d_k^z\|}{\|d_k^z\|} = 0$$

where P_k is the projection matrix $P_k = I - J_k(J_k^\top J_k)^{-1} J_k^\top$.

Assumptions A.1–A.7 are fairly standard for an equality-constrained optimization problem adopting a line-search technique, and in which positive-definiteness of H_k is enforced [44], [45]. Some of the assumptions can, however, be relaxed to meet specific practical demands. The following condition provides a stronger version of Assumptions A.3 and A.4 and will be imposed for the sake of simplicity of our presentation.

- *B.1:* For all $d \in \mathbb{R}^{\hat{n}}$, $\exists \sigma_1, \sigma_2 > 0$, such that $\sigma_1 \|d\|^2 \leq d^\top H_k d \leq \sigma_2 \|d\|^2$ for each k.

An important consequence of Assumption A.7 is that one can recover a local two-step superlinear convergence with positive-definite matrices H_k, requiring only that a projection of each H_k is close to a projection of $\nabla^2 \mathcal{L}^*$ [41], as opposed to the similar convergence result, say for unconstrained problems, that requires $\nabla^2 \mathcal{L}^*$ to be also positive-definite with

$$\lim_{k\to\infty} \frac{\|(H_k - \nabla^2 \mathcal{L}^*)d_k^z\|}{\|d_k^z\|} = 0.$$

We formally state the two-step superlinear convergence result for the constrained problem in the following.

Lemma 2 ([41, Th. 1]): Let Assumptions A.1–A.7 hold for the recurrent learning problem. Then

$$\lim_{l\to\infty} \frac{\|z_{l+1} - z^*\|}{\|z_{l-1} - z^*\|} = 0.$$

In order to ensure superlinear convergence of the kind shown in Lemma 2 from an arbitrary starting point, we equip the SQP with a line-search strategy for choosing a value for the step-length parameter α_k that ensures the value of some well-defined *merit function* is sufficiently reduced for an *acceptable*[1] step to be taken. This strategy provides a way to decide when a progress is made toward a solution. One such merit function commonly used to account for feasibility of the constraints c_k is the ℓ_1 merit function defined as follows:

$$\mathcal{M}_1(z_k; \rho_k) = f_k + \frac{\rho_k}{\omega}\|c_k\|_1, \quad \frac{\rho_k}{\omega} > 0 \tag{24}$$

where $\omega > 1$ is selected heuristically, the motivation for which becomes clear in the ρ_k selection rule of (34). The main goal is to ensure that the (feasible) KKT points of (5) are critical points of $\mathcal{M}_1$.

Upon computing an acceptable step d_k and defining the merit function, we choose a step length α_k, which satisfies the so-called *sufficient decrease* (or Armijo [46]) condition

$$\mathcal{M}_1(z_k + \alpha_k d_k^z; \rho_k) \leq \mathcal{M}_1(z_k; \rho_k) + \nu \alpha_k \nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \tag{25}$$

where $0 < \nu \leq 0.5$ is a small value and $\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k)$ is the directional derivative of $\mathcal{M}_1$ along d_k^z, which we derive in the following lemma.

Lemma 3: Let assumptions A.1–A.5 hold for the iSQPRL problem. Then, the directional derivative of $\mathcal{M}_1(z_k; \rho_k)$ along a step d_k^z satisfies

$$\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \leq g_k d_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1 \tag{26a}$$

$$\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \geq g_k d_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1. \tag{26b}$$

Proof: Applying Taylor's theorem to f and c, we have

$$\begin{aligned}
&\mathcal{M}_1\left(z_k + \alpha_k d_k^z; \rho_k\right) - \mathcal{M}_1(z_k; \rho_k) \\
&= f\left(z_k + \alpha_k d_k^z\right) - f_k + \frac{\rho_k}{\omega}\left\|c\left(z_k + \alpha_k d_k^z\right)\right\|_1 - \frac{\rho_k}{\omega}\|c_k\|_1 \\
&\leq \alpha_k g_k^\top d_k^z + \sigma_2 \alpha_k^2 \left\|d_k^z\right\|^2 + \frac{\alpha_k \rho_k}{\omega}\|c_k\|_1.
\end{aligned}$$

From (22), we have $J_k d_k^z = -c_k + r_k^\lambda$; hence, with $\alpha_k \leq 1$, we get

$$\begin{aligned}
&\mathcal{M}_1\left(z_k + \alpha_k d_k^z; \rho_k\right) - \mathcal{M}_1(z_k; \rho_k) \\
&\quad \leq \alpha_k g_k^\top d_k^z + \sigma_2 \alpha_k^2 \left\|d_k^z\right\|^2 - \frac{\alpha_k \rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1 \\
&\quad = \alpha_k \left(g_k^\top d_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1\right) + \sigma_2 \alpha_k^2 \left\|d_k^z\right\|^2.
\end{aligned}$$

Similar arguments yield the lower bound

$$\begin{aligned}
&\mathcal{M}_1\left(z_k + \alpha_k d_k^z; \rho_k\right) - \mathcal{M}_1(z_k; \rho_k) \\
&\quad \geq \alpha_k \left(g_k^\top d_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1\right) - \sigma_2 \alpha_k^2 \left\|d_k^z\right\|^2.
\end{aligned}$$

Taking the limits

$$\lim_{\alpha_k \to 0} \frac{\mathcal{M}_1\left(z_k + \alpha_k d_k^z; \rho_k\right) - \mathcal{M}_1(z_k; \rho_k)}{\alpha_k} =: \nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k)$$

proves (26). ∎

Consequently, the directional derivative of $\mathcal{M}_1$ along d_k^z is given by

$$\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) = g_k^\top d_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1. \tag{27}$$

[1] An inexact solution d_k is considered acceptable if, together with some penalty term ρ_k of the merit function, it causes a sufficient reduction in the value of the merit function.

As with inexact Newton methods for a general nonlinear system of equations [43], inexact SQP methods require that the residual norm is reduced strictly at each step to allow for the update (z_{k+1}, λ_{k+1}) in (11). This requirement, in our case, is provided in the following conditions.

M.1: The primal step computed in the iSQPRL framework is a descent direction for $\mathcal{M}_1$.

M.2: The residual r_k satisfies $\|r_k\| \leq \sigma_4 \|\mathcal{F}_k\|, 0 < \sigma_4 < 1$.

Indeed, that condition M.2 holds with the primal steps evaluated via GMRES($\hat{m}_r$) is a simple deduction from Proposition 1.

Corollary 1 (of Proposition 1): The residual norm at step k satisfies condition M.2 with $\sigma_4 = \kappa(P_k)\xi_k$ for an appropriate choice of $\hat{m}_r$ assuming the initial guess $d_{k,0} = 0$.

Proof: With $d_{k,0} = 0$, we get from (23) and the definition of $r_{k,\hat{m}}$ that $\|r_k\| \leq \kappa(P_k)\xi_k \|r_{k,0}\| = \kappa(P_k)\xi_k \|b_k - A_k d_{k,0}\| = \kappa(P_k)\xi_k \|b_k\| = \kappa(P_k)\xi_k \|\mathcal{F}_k\|$. ∎

The following result tells us what conditions are sufficient, in particular the choice of ρ_k, such that condition M.1 holds.

Proposition 4: Let Assumptions A.1–A.5 and B.1 hold for the iSQPRL problem. Then, the directional derivative of $\mathcal{M}_1(z_k; \rho_k)$ along a step d_k^z satisfies

$$\begin{aligned}&\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \\ &\leq -{d_k^z}^\top H_k d_k^z + \left(\frac{\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty\right)\left(\|c_k\|_1 - \left\|r_k^\lambda\right\|_1\right) + {d_k^z}^\top r_k^z. \end{aligned} \tag{28}$$

Moreover, if we choose

$$\rho_k > \omega \|\tilde{\lambda}_k\|_\infty \tag{29}$$

and ${d_k^z}^\top r_k^z \leq 0$, then d_k^z is guaranteed to be a descent direction for $\mathcal{M}_1$, and $\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) < 0$ at nonstationary points of (5).

Proof: From (22), we obtain

$$\begin{aligned} g_k^\top d_k^z &= -{d_k^z}^\top H_k d_k^z + {d_k^z}^\top J_k \tilde{\lambda}_k + {d_k^z}^\top r_k^z \\ {d_k^z}^\top J_k \tilde{\lambda}_k &= -c_k^\top \tilde{\lambda}_k + {r_k^\lambda}^\top \tilde{\lambda}_k. \end{aligned}$$

Substituting these into (27) gives

$$\begin{aligned}&\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \\ &= -{d_k^z}^\top H_k d_k^z + {d_k^z}^\top J_k \tilde{\lambda}_k + {d_k^z}^\top r_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1 \\ &= -{d_k^z}^\top H_k d_k^z - \left(c_k - r_k^\lambda\right)^T \tilde{\lambda}_k + {d_k^z}^\top r_k^z - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1. \end{aligned}$$

Using the relation $-(c_k - r_k^\lambda)^\top \tilde{\lambda}_k \leq \|\tilde{\lambda}_k\|_\infty \|c_k - r_k^\lambda\|_1$ from Hölder's inequality, we obtain

$$\begin{aligned}&\nabla_{d_k^z} \mathcal{M}_1(z_k; \rho_k) \\ &\leq -{d_k^z}^\top H_k d_k^z - \left(\frac{\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty\right)\left\|c_k - r_k^\lambda\right\|_1 + {d_k^z}^\top r_k^z \end{aligned} \tag{30}$$

which proves (28) by applying the reverse triangle inequality on $-\|c_k - r_k^\lambda\|_1$. By assumption B.1, and if $d_k^z \neq 0$ solves (9), it is sufficient to show that

$$\Delta\mathcal{M}_1 := {d_k^z}^\top r_k^z - \left(\frac{\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty\right)\left\|c_k - r_k^\lambda\right\|_1 \leq 0 \tag{31}$$

in order to show that d_k^z is a descent direction for $\mathcal{M}_1$, with $\rho_k > \omega \|\tilde{\lambda}_k\|_\infty$. To do this, we define

$$r_k^\top \bar{d}_k = {r_k^\lambda}^\top \tilde{\lambda}_k + {d_k^z}^\top r_k^z = {d_k^z}^\top H_k d_k^z + g_k^\top d_k^z + c_k^\top \tilde{\lambda}_k$$

where $\bar{d}_k = (d_k^z, \tilde{\lambda}_k)$. By noting the relation

$$-\frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1 < -\|\tilde{\lambda}_k\|_\infty \left\|c_k - r_k^\lambda\right\|_1$$

we have

$$\begin{aligned}&{d_k^z}^\top r_k^z - \left(\frac{\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty\right)\left\|c_k - r_k^\lambda\right\|_1 \\ &= \underbrace{{d_k^z}^\top H_k d_k^z + g_k^\top d_k^z + c_k^\top \tilde{\lambda}_k - {r_k^\lambda}^\top \tilde{\lambda}_k}_{= {d_k^z}^\top r_k^z} - \frac{\rho_k}{\omega}\left\|c_k - r_k^\lambda\right\|_1 \\ &\quad + \|\tilde{\lambda}_k\|_\infty \left\|c_k - r_k^\lambda\right\|_1 \\ &r_k^\top \bar{d}_k - r_k^\lambda \tilde{\lambda}_k \leq 0. \end{aligned}$$

∎

Remark 1: Alternatives to the merit reduction condition (29) exist in the literature (see [47]), with their specificity linked to the choice of merit function. The use of a particular merit reduction condition is, however, often well motivated, say, as a way to quantify an appropriate steepness of the directional derivative of the merit function with the residual terms of the inexact SQP step [44].

Corollary 2: Let the assumptions of Proposition 4 hold, and let d_k^z be such that (28) is satisfied. Then, the following property holds:

$$\rho_k \geq \frac{\omega}{2}\left[\frac{g_k^\top d_k^z + {d_k^z}^\top H_k d_k^z - {d_k^z}^\top r_k^z}{\|c_k\|_1 - \|r_\lambda\|_1} + \|\tilde{\lambda}_k\|_\infty\right]. \tag{32}$$

Proof: See Appendix A. ∎

From the property shown in Corollary 2, we see that a necessary requirement for condition (29) is that we choose $\omega \geq 2$ (see [44], [45]). In order to maintain (29) at each step k, thereby ensuring the exactness of $\mathcal{M}_1$ at convergence, a common approach is to *safeguard* the inequality with some positive constant $\bar{\rho}$ in a way that

$$\rho_k \geq \omega \|\tilde{\lambda}_k\|_\infty + \bar{\rho}. \tag{33}$$

The selection rule for ρ_k, proposed in [39], is slightly modified for the inexact SQP problem of this work

$$\rho_k = \begin{cases} \max\{\omega \|\tilde{\lambda}_k\|_\infty + \bar{\rho}, \\ \quad \frac{1}{\omega}(\rho_{k-1} + \omega \|\tilde{\lambda}_k\|_\infty + \bar{\rho})\} & \forall k \neq 1 \\ \omega \|\tilde{\lambda}_k\|_\infty + \bar{\rho}, & \text{if } k = 1. \end{cases} \tag{34}$$

The rationale behind (34) is that it provides relevant bounds on the sequence $\{\rho_k\}$ and allows, for k large enough and toward convergence, the choice $\rho_k = \rho_{k-1}$, provided that $\rho_{k-1} \geq \omega \|\tilde{\lambda}_k\|_\infty + \bar{\rho}$. Coupled with a backtracking line-search procedure, a step length α_k, such that the Armijo condition (25) holds, is chosen.

Many known results for the global convergence of line-search SQP algorithms in constrained optimization are based upon the exactness and stationarity of $\mathcal{M}_1$ at KKT optimality points of the problem and are direct consequences of the descent property of $\mathcal{M}_1$ (when shown to hold) at d_k, such that the line search succeeds for all k. In the same spirit, and with condition M.2 assumed to hold, we obtain the following result (which we prove by following the reasoning in [48, Th. 17.2]).

Theorem 5: Let the assumptions of Proposition 4 hold for the iSQPRL problem, and let ρ_k be chosen according to the rule (34), so that $\rho_k > \omega\|\tilde{\lambda}_k\|_\infty$ holds. Suppose further that the sequence $\{\lambda_k\}$ is bounded. Then, if α_k is bounded below by some positive constant, the sequence of iterates $\{z_k\}$, starting from an arbitrary point, converges to a stationary point of $\mathcal{M}_1$.

Proof: Since $\{\lambda_k\}$ is bounded, and hence $\{\rho_k\}$, then it is easy to show that there exists an index k_1, such that $\rho_k \approx \rho \geq \omega\|\tilde{\lambda}_k\|_\infty + \bar{\rho}$ for all $k \geq k_1$. The adaptation rule (34) indeed ensures that ρ_k does not increase unnecessarily in attaining this ρ, since then $\{\tilde{\lambda}_k\}$ is bounded. Therefore, by the descent property of $\mathcal{M}_1$ in Proposition 4, we get that $\mathcal{M}_1$ remains exact.

Consider the set of indices $\mathcal{K} := \{k \geq k_1 : \alpha_k < 1\}$. Then, each of the step $k \in \mathcal{K}$ causes a sufficient decrease in $\mathcal{M}_1$ by the backtracking line-search procedure. Moreover, with $\mathcal{M}_1(z_k;\rho) \geq K > -\infty$ for some constant K, the sufficient decrease condition (25) shows that

$$\alpha_k \nabla_{d_k^z} \mathcal{M}_1(z_k;\rho_k) \to 0. \tag{35}$$

Now let $\bar{\alpha} \leq \alpha_k$ be some constant for which the backtracking line-search procedure fails. Then, with $k \in \mathcal{K}$, at least one of the following holds:

$$z_k + \bar{\alpha} d_k^z \notin S \tag{36a}$$

$$\mathcal{M}_1(z_k + \bar{\alpha} d_k^z;\rho_k) > \mathcal{M}_1(z_k;\rho_k) + \nu\bar{\alpha}\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k). \tag{36b}$$

However, by Assumption A.1 and with $d_k^z \neq 0$, (36a) does not hold. Hence, we have (36b). From the proof of Lemma 3 and (27), we get

$$\mathcal{M}_1(z_k + \bar{\alpha} d_k^z;\rho_k) - \mathcal{M}_1(z_k;\rho_k) \leq \bar{\alpha}\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k) + \sigma_2\bar{\alpha}^2\|d_k^z\|^2$$

which together with (36b) gives

$$-(1-\nu)\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k) \leq \sigma_2\bar{\alpha}\|d_k^z\|^2.$$

From (30) and Assumption B.1, we have

$$\sigma_1\|d_k^z\|^2 \leq -\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k)$$

and since $\nu > 1$, we get from the above two inequalities

$$\alpha_k \geq \bar{\alpha} \geq \frac{\sigma_1}{\sigma_2}(1-\nu) > 0. \tag{37}$$

Therefore, (35) implies $\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k) \to 0$. Consequently, by (30) and $\rho_k \geq \omega\|\tilde{\lambda}_k\|_\infty + \bar{\rho}$, we have

$$d_k^{z\top} H_k d_k^z \to 0, \quad c_k \to 0.$$

By our assumptions on H_k, the above implies $d_k^z \to 0$, which we know to hold if and only if z_k is a feasible point satisfying the KKT optimality conditions in (12). ∎

D. Practical Aspects

We discuss two practical issues that are considered in the implementation of our algorithm with the goal of enabling the convergence results discussed so far.

Algorithm 1 GRL($\hat{m}_r$) With Nonmonotone Line Search

Input: data $\{u_t, y_t\}_{t=0}^{N-1}$, initial guess for $(x_0, \theta_x, \theta_y)$, number of epochs N_e, line-search parameters $0 < \mu < 1$, ν, $\bar{\rho}$, initial Lagrangian multiplier vector λ_0, initial BFGS matrix H_0
Output: updated parameters $(x_0, \theta_x, \theta_y)$
Evolve (1). Set $z_0 \leftarrow [x_1^\top \cdots x_{N-1}^\top x_0^\top \theta_y^\top \theta_x^\top]^\top$
Compute $f_0, c_0, J_0, g_0, \mathcal{L}_0$
$k \leftarrow 0$
repeat
  Construct KKT system (14)
  Solve KKT system with preconditioning (see Section III-B and Section III-D1). Get $(d_k^z, \tilde{\lambda}_k)$. Get d_k^λ as in (10)
  Get ρ_k using (34). Define the merit function $\mathcal{M}_1$ (24). Define m_k (39)
  if $\mathcal{M}_1(z_k + d_k^z;\rho_k) \leq m_k + \nu\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k)$ **then**
    $z_{k+1} \leftarrow z_k + d_k^z$
    $\lambda_{k+1} \leftarrow \lambda_k + d_k^\lambda$
  else
    Construct KKT system (41)
    Solve KKT system with preconditioning (see Section III-B and Section III-D1). Get $(\bar{d}_k^z, \bar{\lambda}_k)$
    $\bar{d}_\lambda \leftarrow \bar{\lambda}_k - \lambda_k$
    if $\|\bar{d}_k^z\| > \|d_k^z\|$ **then**
      $\bar{d}_k^z \leftarrow 0, \bar{d}_\lambda \leftarrow 0$
    end if
    $\alpha_k \leftarrow 1$
    while $\mathcal{M}_1(z_k + \alpha_k d_k^z + \alpha_k^2 \bar{d}_k^z) > m_k + \nu\alpha_k\nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k)$ **do**
      $\alpha_k \leftarrow \mu\alpha_k$
    end while
    $z_{k+1} \leftarrow z_k + \alpha_k d_k^z + \alpha_k^2 \bar{d}_k^z$
    $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k d_k^\lambda + \alpha_k^2 \bar{d}_\lambda$
  end if
  Compute $f_{k+1}, c_{k+1}, J_{k+1}, g_{k+1}, \mathcal{L}_{k+1}$
  Get U_k as in (18). $H_{k+1} \leftarrow H_k + U_k$
  $k \leftarrow k+1$
until $k \geq N_e$

1) Preconditioning for GMRES($\hat{m}_r$): The (left) preconditioned GMRES($\hat{m}_r$) is the GMRES($\hat{m}_r$) applied to the system

$$M_k^{-1} A_k d_k = M_k^{-1} b_k$$

where $M_k \in \mathbb{R}^{\hat{n}\times\hat{m}}$ is a suitably chosen, problem-dependent preconditioner. In this work, we implement GMRES($\hat{m}_r$) for the solution of (14) at each step k, left-preconditioned with the nonsingular preconditioner

$$M_k = \begin{bmatrix} G_k & J_k^\top \\ J_k & 0 \end{bmatrix} \tag{38}$$

where $G_k \neq H_k$ is an $n \times n$ matrix chosen so that M_k is invertible. This preconditioning method, known in the literature as *constraint preconditioning* [49], has been used extensively to achieve improved convergence results for problems of the kind (14). There are various options for the choice of G_k (see [49] for an overview).

2) Enforcing Superlinear Convergence: An important requirement for the superlinear convergence of SQP algorithms is that the line-search strategy accepts a unit step length[2] for all k. This superlinear step, however, may not be accepted for the merit function $\mathcal{M}_1$, thereby inhibiting fast convergence.

[2]It is remarked that obtaining a unit step length does not always hold [41].

This condition, known as the *Maratos effect*, is avoided in this work by incorporating the nonmonotone line-search procedure introduced in [50] into our algorithm allowing to occasionally take nonmonotone steps toward progress (see lines 14–25 in Algorithm 1). The approach is based on the observation that with the number m_k defined by

$$m_k := \max_{j=0,\ldots,h_m} \mathcal{M}_1(z_{k-j};\rho_k) \tag{39}$$

over a nonmonotone horizon h_m (typically, $h_m = 3$), the function $\mathcal{M}_1$ "eventually" satisfies the *modified* sufficient decrease condition

$$\mathcal{M}_1(z_k + \alpha_k d_k^z;\rho_k) \le m_k + \nu\alpha_k \nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k). \tag{40}$$

In specific terms, it is proposed in [50] to perform the variable update

$$z_{k+1} = z_k + \hat{\alpha}_k d_k^z + \hat{\alpha}_k^2 \bar{d}_k^z$$

whenever the merit function does not accept a unit step length in the test (40), where $0 < \hat{\alpha}_k \le 1$ is the number returned by a backtracking line search performed on

$$\mathcal{M}_1\big(z_k + \alpha_k d_k^z + \alpha_k^2 \bar{d}_k^z;\rho_k\big) \le m_k + \nu\alpha_k \nabla_{d_k^z}\mathcal{M}_1(z_k;\rho_k)$$

and $\bar{d}_k^z$ is defined by

$$\begin{bmatrix} H_k & J_k^\top \\ J_k & 0 \end{bmatrix}\begin{bmatrix} \bar{d}_k^z \\ -\bar{\lambda}_k \end{bmatrix} = \begin{bmatrix} -g_k \\ -\bar{c}_k \end{bmatrix} \tag{41}$$

with $\bar{c}_k = c(z_k + d_k^z)$. It is shown that under suitable conditions (such as those assumed in this work), this "correction plus arc-search" procedure globally preserves Q-superlinear convergence of the SQP algorithm, even though now the merit function $\mathcal{M}_1$ is not forced to reduce at each step k [50].

IV. Offline Learning of the Control Network

A key element considered in discussing our proposed algorithm for the optimization problem of the second stage of training RCNN is the structure of the Hessian terms which the method efficiently exploits. Although an FNN is often a simpler network to train than a recurrent network, it is still desirable to exploit curvature information about the training data to improve learning. Adeoye and Bemporad [33] establish a GGN algorithm that ultimately exploits the idea of self-concordance [51] for approximating batchwise parameter updates in an unconstrained optimization problem. In this work, we adapt the GGN-SCORE algorithm[3] of [33] for training the control network as described next.

Suppose that in stage 2 of learning the RCNN, we process mini-batches of sample streams in a multistep ahead fashion, composed of the current and future stacks $(\hat{u}_k, \hat{y}_k)$, $\hat{u}_k \in \mathbb{R}^{n_u \cdot n_b}$, and $\hat{y}_k \in \mathbb{R}^{n_y \cdot n_b}$, where n_b is the mini-batch size, $1 < n_b < n_{\theta_u}$ (possibly $1 < n_b \ll n_{\theta_u}$), $k = 1, 2, \ldots, \lceil (N_u/n_b) \rceil$, and $N_u \ge n_b$ is a desired number of training samples. Let $\bar{r}: \mathbb{R}^{n_{\theta_u}} \to \mathbb{R}$ be a self-concordant function[4] with parameter $M_{\bar{r}}$, that is, the inequality

$$\left|\left\langle v, \left(\nabla^3 \bar{r}(\theta)[v]\right) v\right\rangle\right| \le 2M_{\bar{r}}\left\langle v, \nabla^2 \bar{r}(\theta) v\right\rangle^{3/2} \tag{42}$$

holds for any θ in the closed convex set $\mathbb{W} \subseteq \mathbb{R}^{n_{\theta_u}}$ and $v \in \mathbb{R}^{\theta_u}$, where $\nabla^3 \bar{r}(\theta)[v] \in \mathbb{R}^{n_{\theta_u} \times n_{\theta_u}}$ denotes the limit

$$\nabla^3 \bar{r}(\theta)[v] := \lim_{t \to 0} \frac{1}{s}\left[\nabla^2 \bar{r}(\theta + sv) - \nabla^2 \bar{r}(\theta)\right], \quad s \in \mathbb{R}.$$

Note that any self-concordant function can be scaled to have $M_{\bar{r}} = 1$ in (42) [52, Corollary 5.1.3].

Let us regularize problem (7) with $\bar{r}(\theta_u)$ and a penalty parameter $\lambda_{\bar{r}} \in \mathbb{R}$ to have the regularized minimization problem[5]

$$\min_{\theta_u} \quad \bar{f}(\theta_u) + \lambda_{\bar{r}} \bar{r}(\theta_u) \tag{43}$$

where $\bar{f} \equiv -\hat{f}$. Then, letting $\bar{z} := \theta_u$ and $n_d := n_y \cdot n_b$, the mini-batch GGN update rule for optimizing over θ_u is

$$\bar{z}_{k+1} = \bar{z}_k - \left(J_{\hat{y}}^\top Q_{\bar{f}} J_{\hat{y}} + \lambda_{\bar{r}} \bar{H}_k\right)^{-1} J_{\hat{y}}^\top g_{\bar{f}} \tag{44}$$

where at step k, $\bar{H}_k \in \mathbb{R}^{n_{\theta_u} \times n_{\theta_u}}$ is the Hessian of $\bar{r}$ with respect to θ_u, $J_{\hat{y}} \in \mathbb{R}^{n_d \times n_{\theta_u}}$ is the Jacobian of $\hat{y}_k$ with respect to θ_u, $g_{\bar{f}} \in \mathbb{R}^{n_d}$ is the residual vector defined as the gradient of $\bar{f}$ with respect to $\hat{y}_k$, and $Q_{\bar{f}} \in \mathbb{R}^{n_d \times n_d}$ is the Hessian of $\bar{f}$ with respect to $\hat{y}_k$. Note that dependence of the terms on k is ignored for notational convenience.

Clearly, the computations involved in (44) are highly prohibitive. Adeoye and Bemporad [33] use a generalized inverse identity on (44) to derive a more computationally convenient way to update $\bar{z}$ as follows:

$$\bar{B}_k := \lambda_{\bar{r}} I + \bar{Q}_k \bar{J}_k \bar{H}_k^{-1} \bar{J}_k^\top \tag{45a}$$

$$\bar{z}_{k+1} = \bar{z}_k - \bar{H}_k^{-1} \bar{J}_k \bar{B}_k^{-1} \bar{e}_k \tag{45b}$$

where $I \in \mathbb{R}^{n_d \times n_d}$ is an identity matrix, $\bar{J}_k \in \mathbb{R}^{(n_d+1) \times n_{\theta_u}}$ is $J_{\hat{y}}$ augmented with $\lambda_{\bar{r}} \bar{g}_k$ the gradient of $\lambda_{\bar{r}} \bar{r}$ with respect to θ_u, $\bar{e}_k \in \mathbb{R}^{n_d+1}$ is $g_{\bar{f}}$ augmented with a unit term, and $\bar{Q}_k \in \mathbb{R}^{(n_d+1)\times(n_d+1)}$ is a diagonal matrix with diagonal terms defined by $Q_{\bar{f}}$ with an additional zero diagonal term. That is,

$$\bar{J}_k^\top = \left[\begin{array}{c:c} J_{\hat{y}}^\top & \lambda_{\bar{r}} \bar{g}_k \end{array}\right], \bar{Q}_k = \left[\begin{array}{c:c} Q_{\bar{f}} & 0 \\ \hdashline 0 & 0 \end{array}\right], \bar{e}_k = \left[\begin{array}{c} g_{\bar{f}} \\ \hdashline 1 \end{array}\right].$$

We note that the Hessian $\bar{H}_k$ is a diagonal matrix with diagonal elements that can be quite cheap to obtain, and hence, its inverse $\bar{H}_k^{-1}$ can clearly be computed efficiently.

Furthermore, with motivation from the *Newton decrement* framework in convex optimization, a learning rate $\sigma/(1 + M_{\bar{r}} \bar{\eta}_k)$ was introduced in (45) to obtain the full GGN-SCORE update step

$$\bar{B}_k := \lambda_{\bar{r}} I + \bar{Q}_k \bar{J}_k \bar{H}_k^{-1} \bar{J}_k^\top \tag{46a}$$

$$\bar{z}_{k+1} = \bar{z}_k - \frac{\sigma}{1 + M_{\bar{r}} \bar{\eta}_k} \bar{H}_k^{-1} \bar{J}_k \bar{B}_k^{-1} \bar{e}_k \tag{46b}$$

where $0 < \sigma \le 1$ and $\bar{\eta}_k = \langle \bar{g}_k, \bar{H}_k^{-1} \bar{g}_k \rangle^{1/2}$. Hence, the self-concordance of $\bar{r}$ helps to control the rate at which its second derivatives change within the region of convergence, thereby giving the problem an affine-invariant structure [33], as well as to select a learning rate at step k without a line-search technique.

An efficient way to compute vector $\bar{B}_k^{-1} \bar{e}_k$ in (46b) is to solve a linear system, say, by QR factorization, and not by

[3] An implementation of GGN-SCORE is available in the Julia package SelfConcordantSmoothOptimization.jl [68].

[4] For details on self-concordant functions, we refer the interested reader to the references provided in [33].

[5] Note that $\bar{f} \equiv \hat{f}$ is used if the problem defines R_t in (7) as a cost to be minimized rather than a reward to be maximized.

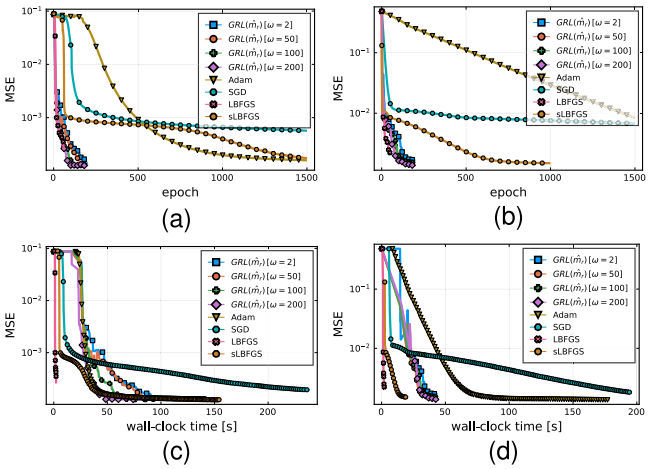


Fig. 3. Convergence curves for the function approximation in DCRNN training using GRL($\hat{m}_r$) with $\bar{\rho} = 0.8$, first-order methods (Adam [55] and stochastic gradient descent (SGD) [56]), LBFGS (best memory size: 1) with line search [57, Sec. 3.5] (training stopped at line-search failure), and sLBFGS (best memory size: 10) with adaptive step size [58]. Hidden state vector is initialized to zero. MSE stands for the mean-squared error. (a) Mountain car. (b) Ethylene oxidation. (c) Mountain car. (d) Ethylene oxidation.

a direct matrix inversion. However, in our adaptation of the algorithm, it is observed that, by construction, the diagonal matrix $\underline{\mathrm{B}}_k \in \mathbb{R}^{n_{\theta_u} \times n_{\theta_u}}$, with diagonal terms defined by

$$\underline{\mathrm{b}}_{i,j} = \bar{\delta}_{i,j}/\bar{b}_{i,i} \quad i, j = 1, \ldots, n_{\theta_u}$$

provides a good approximation to $\bar{B}_k^{-1}$, where $\bar{\delta}_{i,j}$ is the Kronecker delta function and $\bar{b}_{i,i}$ are the diagonal entries of $\bar{B}_k$. Hence, for the training of the control network in this work, we propose the approximation $\bar{B}_k^{-1} \approx \underline{\mathrm{B}}_k$, so that (46b) becomes

$$\bar{z}_{k+1} = \bar{z}_k - \frac{\sigma}{1 + M_{\bar{r}}\bar{\eta}_k} \bar{H}_k^{-1} \bar{J}_k \underline{\mathrm{B}}_k \bar{e}_k.$$

Once the RCNN is trained on the representative DCRNN model to a desired accuracy, the control network parameters become fixed and are used to perform simulation tasks on the true system. Simulation results obtained from a classical RL problem and a model predictive control (MPC) steady-state regulation problem are presented in Section VI to demonstrate the efficiency of our approach.

V. Complexity Analysis

In this section, we estimate the computational complexity of the method proposed in this article. The proposed GRL($\hat{m}_r$) algorithm is summarized into three main steps, viz., constructing the Newton–KKT system, solving the Newton–KKT system, and updating the optimization variables.

1) Constructing the KKT System: Constructing the Newton–KKT system at each iteration k involves evaluating the constraints and their gradients with a combined worst-case complexity of $\mathcal{O}(2mn)$. Hence, in addition to evaluating the modified BFGS Hessian matrix, constructing the KKT system has an overall complexity of $\mathcal{O}(n^2 + 2mn)$ per iteration in the worst-case scenario.

2) Solving the KKT System: The KKT system is solved using the restarted GMRES scheme, which depends on both the number of restarts required to reach a given tolerance and the restart parameter $\hat{m}_r$. Let us denote by N_1, the number of GMRES restarts required to reach a specified tolerance. Then, the matrix-vector multiplications and vector operations in the restarted GMRES procedure have a (fixed) worst-case complexity of $\mathcal{O}(N_1 n \hat{m}_r^2)$, which involves the construction of the upper Hessenberg matrix $\tilde{H}_{k,\hat{m}_r}$ in the Arnoldi iteration. Here, we assume the use of an updating QR algorithm in the solution of (21). Note that $\hat{m}_r$ is typically small, say between 10 and 50, compared with n.

3) Updating the Optimization Variables: The line-search method using the Armijo update rule with the ℓ_1 merit function has a worst-case complexity of $\mathcal{O}(n)$ per iteration.

Overall, the complexity of the proposed GRL($\hat{m}_r$) algorithm for training the RNN model considered in this article is estimated as $\mathcal{O}(2n^2 N_e + 2nN_e + 2mnN_e)$. Since the number of iterations, N_e, is generally small for GRL($\hat{m}_r$), which also provides a better solution quality, a compromise is made on computational complexity in favor of GRL($\hat{m}_r$) for convergence speed in terms of small iteration number and solution quality, against benchmark (first-order) BPTT algorithms.

As we mentioned before, we may use any convenient gradient-based algorithm for learning the control FNN. Hence, the effective complexity involved in training the full RCNN will be the complexity of the GRL($\hat{m}_r$) algorithm plus the complexity of the algorithm used to train the control FNN.

VI. Numerical Examples

In the following examples, the RCNN is trained with σ_x and σ_h set, respectively, to the hyperbolic tangent (tanh) and the rectified linear unit (ReLU) functions. The RCNN is structured as described in Section II. The entries of the DCRNN weights are initialized to a normally distributed number randomly generated by the Mersenne–Twister pseudorandom number generator, while the bias vectors, the hidden states, and the Lagrangian parameter are all initialized to the zero array. $\bar{\rho} = 0.8, \omega = 2, N_e = 200, \mu = 0.8$, and $\nu = 0.5$. We set $\mathcal{R}(x_0, \theta_x, \theta_y) = (Q_\alpha/2)\|x_0\|_2^2 + (Q_\beta/2)(\|\theta_x\|_2^2 + \|\theta_y\|_2^2)$ with $Q_\alpha = 1.0$ and $Q_\beta = 10^{-5}$. The restart parameter $\hat{m}_r$ in GRL($\hat{m}_r$) is upper bounded by 50 for solving the resulting saddle-point systems with $G_k = I$, the identity matrix, in (38). The control network weights V_{uh} and V_{hx} are initialized according to the Kaiming uniform initialization scheme described in [53]; that is, we initialize each entry of the matrices to a number generated randomly from a uniform distribution in the interval $[-d, d]$ where $d = 4(3/\hat{n}_w)^{1/2}$, and $\hat{n}_w$ represents n_u or n_h accordingly. The bias vectors here are initialized to the zero array. The control network is trained with $\bar{r}$ set to the pseudo-Huber function $r_{\bar{\mu}}$ parameterized by $\bar{\mu}$, $r_{\bar{\mu}}(\theta_u) := (\bar{\mu}^2 + \theta_u^2)^{1/2} - \bar{\mu}$, and $\lambda_{\bar{r}} = 10^{-2}, \sigma = M_{\bar{r}} = 1$. All bound constrained continuous variables $a_i \le x_i \le b_i$ are transformed to [54]

$$x_i = t_{a_i,b_i}(\hat{x}_i) = \frac{b_i + a_i}{2} + \frac{b_i - a_i}{2}\left(\frac{2\hat{x}_i}{1 + \hat{x}_i}\right), \quad i = 1, 2, \ldots, n. \tag{47}$$

First, in Figs. 3 and 4, we visualize the convergence of Algorithm 1 using the two examples considered. The approximation errors and CPU time (in seconds) are compared with those of standard BPTT algorithms in Tables I and II. The

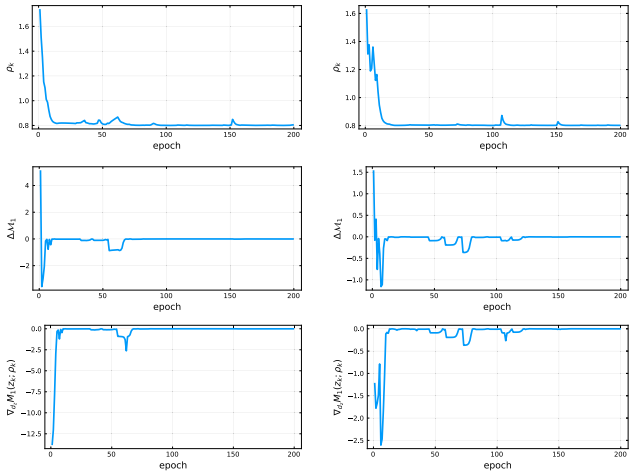


Fig. 4. Convergence of Algorithm 1 for terms defined in Proposition 4 and Theorem 5 ($\bar{\rho} = 0.8$ and $\omega = 2$). Left: mountain car dynamics. Right: ethylene oxidation process.

TABLE I

SOLUTION COMPARISON BETWEEN THE PROPOSED METHOD AND SGD, ADAM, LBFGS, AND SLBFGS (MOUNTAIN CAR PROBLEM)

BPTT algorithm	Approximation error (MSE)	Iter	CPU time [s]
SGD	5.6828×10^{-4}	1500	5.6584×10
Adam	1.5821×10^{-4}	1500	4.9532×10
LBFGS	7.1389×10^{-4}	30	2.8670×10^{0}
sLBFGS	3.9884×10^{-4}	1000	2.8071×10
GRL($\hat{m}_r$), $\omega = 2$	1.3451×10^{-4}	200	6.4956×10
GRL($\hat{m}_r$), $\omega = 50$	$\mathbf{1.2702 \times 10^{-4}}$	200	1.0091×10^{2}
GRL($\hat{m}_r$), $\omega = 100$	1.3164×10^{-4}	200	5.6220×10
GRL($\hat{m}_r$), $\omega = 200$	1.2707×10^{-4}	200	1.1914×10^{2}

results shown in Figs. 3 and 4 corroborate with our discussions in Section III-C, in particular, Proposition 4 and Theorem 5. We observe small variations in ρ_k, that is, $\rho_k \approx \rho$, as $\nabla_{d_k^z} \mathcal{M}_1 \to 0$.

A. Classical RL Example

The RL problem considered in this example is the classical mountain car problem described in [2]. The task is to drive an underpowered car to the top of a steep mountain.

The continuous observables here are the states of the car given by $y_t = [\bar{y}_t, \dot{\bar{y}}_t]$, where $\bar{y}_t$ is the car's position and $\dot{\bar{y}}_t$ is its velocity at any given time t. For each time the car reaches the top of the mountain, it gets a positive reward R_t

$$R_t = \text{logistic}_{10}(\hat{y}_t - y^{\text{ref}}) \tag{48}$$

in which σ_r has been set to a steep logistic function $\text{logistic}_{10}(x) := (1/1 + e^{-10x})$ [59], and $y^{\text{ref}} = 0.5$. The function R_t returns a number that is approximately equal to 1 whenever $\hat{y}_t > y^{\text{ref}}$ and 0 otherwise. The control action is the applied force $u_t \in \{-1, 0, 1\}$. The car's motion is described by the set of equations

$$\begin{aligned} \bar{y}_{t+1} &= \bar{y}_t + \dot{\bar{y}}_t \\ \dot{\bar{y}}_{t+1} &= \dot{\bar{y}}_t + 0.001 u_t - 0.0025 \cos(3\bar{y}_t) \end{aligned}$$

with the bound constraints $-1.2 \le \bar{y}_t \le 0.5$ and $-0.07 \le \dot{\bar{y}}_t \le 0.07$. The task is episodic in that, when $\bar{y}_t$ has reached its right bound, the goal was reached, and an episode is completed. However, when it reached its left bound, $\dot{\bar{y}}_t$ was reset to zero.

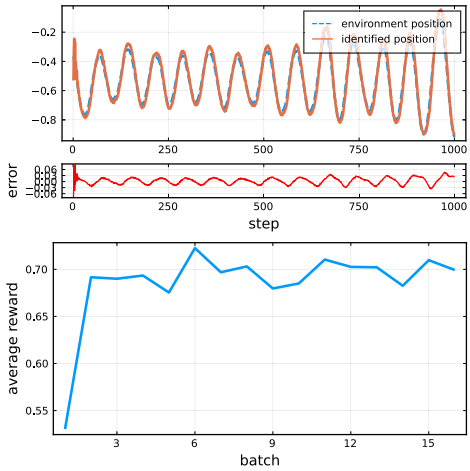


Fig. 5. RCNN offline training results for the mountain car environment. Top: DCRNN identified model in stage I (testing via open-loop simulation); error $= y - \hat{y}$. Bottom: stage II training of the RCNN.

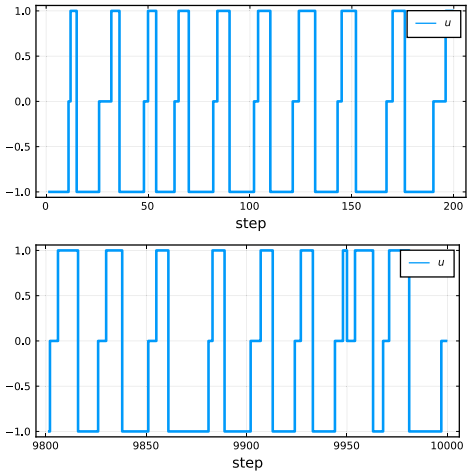


Fig. 6. Real-time RCNN control actions $u_t \in \{-1, 0, 1\}$. Top: control actions for the first 200 steps. Bottom: control actions for the last 200 steps.

Each episode starts from an initial random position $\bar{y}_t \in [-0.6, -0.4]$ and a zero velocity. We trained the RCNN using the algorithms proposed in this article, where the control inputs u_t and environment observables y_t were given to the network as inputs and targets accordingly. Here, $n_u = 1$ and $n_y = 2$, and we used $m_- = m_+ = 50, n_x = 2, n_h = 128, N_u = 1000$, and $n_b = 64$.

Given the characteristics of the problem, we chose σ_u as the tanh function that keeps the variables $\hat{u}_t$ in the interval $[-1, 1]$ during training. However, for our simulations, we follow the standard approach of bucketing the continuous outputs of σ_u into the discrete set $\{-1, 0, 1\}$. This approach for ensuring the input bound constraints are satisfied can be used in different problem settings by using an appropriate σ_u or by using the variable transform (47). Furthermore, we set $G_r = [1, 0]$, and R_t is defined in (48).

The RCNN training results are shown in Fig. 5. In the identification stage, the *effective* DCRNN model was trained on three randomly generated datasets with trained hidden states from one dataset used in the next initialization.[6]

[6]This way, we are able to generate a good start value for the hidden state to improve the representation power of the DCRNN.

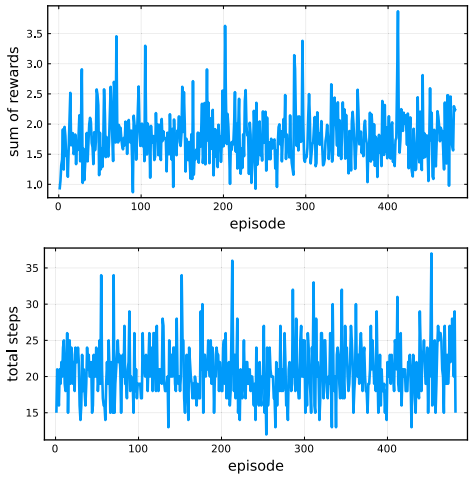


Fig. 7. Real-time performance of the trained RCNN control actions in the mountain car environment.

TABLE II

SOLUTION COMPARISON BETWEEN THE PROPOSED METHOD AND SGD, ADAM, LBFGS, AND SLBFGS (ETHYLENE OXIDATION)

BPTT algorithm	Approximation error (MSE)	Iter	CPU time [s]
SGD	6.7298×10^{-3}	1500	5.3596×10
Adam	8.3706×10^{-3}	1500	4.4071×10
LBFGS	1.5866×10^{-3}	46	1.6200×10^{0}
sLBFGS	1.4902×10^{-3}	1000	1.9693×10
GRL($\hat{m}_r$), $\omega = 2$	1.6720×10^{-3}	200	4.3819×10
GRL($\hat{m}_r$), $\omega = 50$	1.4948×10^{-3}	200	4.2956×10
GRL($\hat{m}_r$), $\omega = 100$	1.5043×10^{-3}	200	4.4190×10
GRL($\hat{m}_r$), $\omega = 200$	$\mathbf{1.3642 \times 10^{-3}}$	200	4.5000×10

In the second stage, the control network was trained with 1000 noisy observations (5% Gaussian noise over ten seeds). Simulation performance on the true system with 10 000 noisy observations (5% Gaussian noise over ten seeds) are shown in Figs. 6 and 7. Adding noise at these stages creates a more realistic situation [60], [61]. As displayed, we were able to train the RCNN with our algorithms to keep the agent's rewards and steps taken per episode within a reasonably good threshold in the true environment.

B. Chemical Process Example

The ethylene oxidation process in a nonisothermal continuously stirred tank reactor is considered in this example. The oxidation process is described by the following dimensionless equations [62]:

$$\begin{aligned}
\dot{\bar{y}}_1 &= u_1(1 - \bar{y}_1\bar{y}_4) \\
\dot{\bar{y}}_2 &= u_1(u_2 - \bar{y}_2\bar{y}_4) - A_1 e^{\gamma_1/\bar{y}_4}(\bar{y}_2\bar{y}_4)^{0.5} \\
&\quad - A_2 e^{\gamma_2/\bar{y}_4}(\bar{y}_2\bar{y}_4)^{0.25} \\
\dot{\bar{y}}_3 &= -u_1\bar{y}_3\bar{y}_4 + A_1 e^{\gamma_1/\bar{y}_4}(\bar{y}_2\bar{y}_4)^{0.5} - A_3 e^{\gamma_3/\bar{y}_4}(\bar{y}_3\bar{y}_4)^{0.5} \\
\dot{\bar{y}}_4 &= \frac{u_1}{\bar{y}_1}(1 - \bar{y}_4) + \frac{B_1}{\bar{y}_1} e^{\gamma_1/\bar{y}_4}(\bar{y}_2\bar{y}_4)^{0.5} + \frac{B_2}{\bar{y}_1} e^{\gamma_2/\bar{y}_4}(\bar{y}_2\bar{y}_4)^{0.25} \\
&\quad + \frac{B_3}{\bar{y}_1} e^{\gamma_3/\bar{y}_4}(\bar{y}_3\bar{y}_4)^{0.5} - \frac{B_4}{\bar{y}_1}(\bar{y}_4 - T_c)
\end{aligned}$$

where the continuous observables are given by the dimensionless state variables $y_t = [\bar{y}_1, \bar{y}_2, \bar{y}_3, \bar{y}_4]$, $\bar{y}_1, \bar{y}_2, \bar{y}_3$, and $\bar{y}_4$ represent the gas density, ethylene concentration, ethylene oxide concentration, and temperature in the reactor, respectively, at any given time t. The control action is $u_t = [u_1, u_2]$, where u_1 is the feed volumetric flow rate and u_2 is the concentration of ethylene in the feed, with bound constraints $0.0704 \leq u_1 \leq 0.7042$ and $0.2465 \leq u_2 \leq 2.4648$. The parameters in the equations above are given values adopted from [62] (see Appendix B).

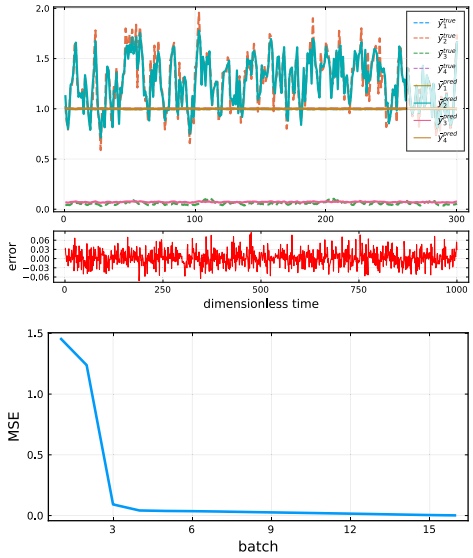


Fig. 8. RCNN training results for ethylene oxidation process dynamics. Top: DCRNN identified model in stage I (testing via open-loop simulation); error $= y - \hat{y}$. Bottom: stage II training of the RCNN. MSE stands for the mean-squared error.

The goal of the process is to maximize the production of ethylene oxide for a limited reactant feedstock, starting from an initial state $y_0 = [0.997, 1.264, 0.209, 1.004]$ with a sampling period of $\Delta t = 9.36$. Again, we trained the RCNN using the algorithms proposed in this article, where the control inputs u_t and environment observables y_t were given to the network as inputs and targets accordingly. Here, $n_u = 2$ and $n_y = 4$, and we used $m_- = m_+ = 40, n_x = 2, n_h = 128, N_u = 1000$, and $n_b = 64$. Here, σ_u was taken to be the identity map with bound constraints enforced through (47), $G_r = [1, 1, 1, 1]$, and R_t is defined by taking the squared difference between $\hat{y}_t$ and a reference point y^{ref}, with σ_r and σ_y both set to the identity map, so that we have

$$R_t = (\hat{y}_t - y^{\text{ref}})^2$$

where we set $y^{\text{ref}} \equiv [\bar{y}_{1s}, \bar{y}_{2s}, \bar{y}_{3s}, \bar{y}_{4s}] = [0.998, 0.424, 0.032, 1.002]$, the open-loop asymptotically stable steady state of the process provided in [62]. Note that in this problem, R_t is specified as a cost to be minimized.

The training and identification results are displayed in Fig. 8. In the first identification stage, the *effective* DCRNN model was trained on five randomly generated datasets with trained hidden states from the first dataset used in the second initialization. Simulation performance on the true system is shown in Fig. 9. As shown, the constant control inputs reported in [62] to bring the process states to the asymptotically stable steady-state value resemble with what is obtained by our approach, that is, $u_s = [u_{1s}, u_{2s}] = [0.35, 0.5]$ in [62].

Regarding the comparison between the proposed GRL($\hat{m}_r$) algorithm for training the DCRNN with respect to first-order SGD and Adam optimizers and two quasi-Newton methods [limited-memory BFGS (LBFGS) and stochastic LBFGS

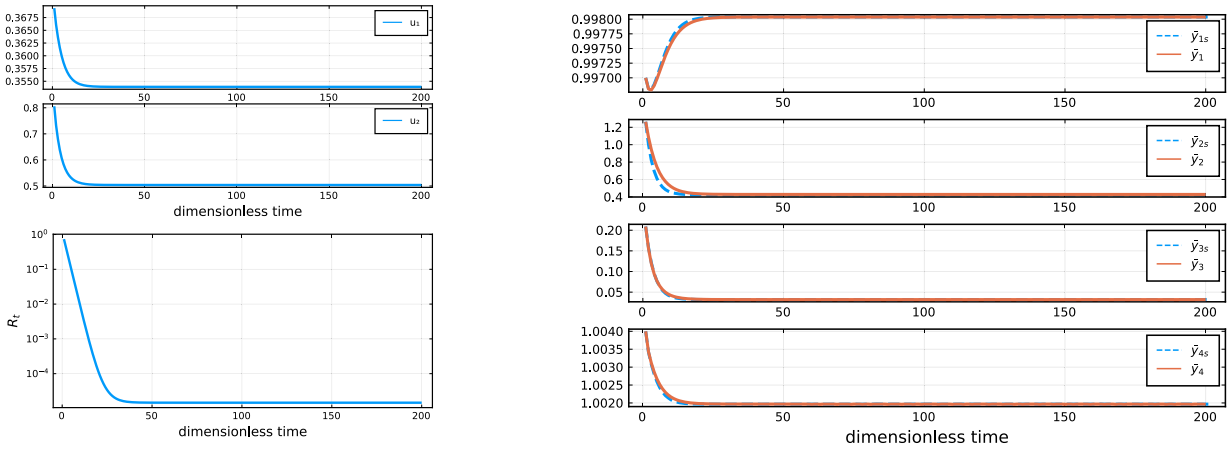


Fig. 9. Performance of the trained RCNN control actions in ethylene oxidization process. Top left: RCNN control inputs $u_t = [u_1, u_2]$ with bound constraints $0.0704 \leq u_1 \leq 0.7042$ and $0.2465 \leq u_2 \leq 2.4648$. Bottom left: Simulation squared errors $R_t = (\hat{y}_t - y^{\text{ref}})^2$. Right: $\hat{y}_t = [\bar{y}_1, \bar{y}_2, \bar{y}_3, \bar{y}_4]$ are the RCNN predictions of the steady state $y^{\text{ref}} = [\bar{y}_{1s}, \bar{y}_{1s}, \bar{y}_{3s}, \bar{y}_{4s}]$.

TABLE III
COMPARISON BETWEEN THE PROPOSED METHOD AND SELECTED RNN STRUCTURES IN THE MOUNTAIN CAR PROBLEM

Method	Approximation error (MSE)	Iter	CPU time [s]
Vanilla RNN + SGD	2.0034×10^{-4}	8000	1.0377×10^{2}
DRRRNN + SGD + ℓ_1 [63]	3.0245×10^{-4}	8000	1.5338×10^{2}
DRRGRU + SGD + ℓ_1 [63]	1.7761×10^{-4}	8000	1.6378×10^{2}
Ours (GRL($\hat{m}_r$), $\omega = 50$)	1.3689×10^{-4}	**250**	5.4477×10
Ours (GRL($\hat{m}_r$), $\omega = 200$)	$\mathbf{1.3435 \times 10^{-4}}$	**250**	$\mathbf{5.3771 \times 10}$

(sLBFGS)], in Fig. 3, we compare the corresponding convergence curves. Although shown to sometimes converge faster than GRL($\hat{m}_r$), LBFGS (for which we report the best results obtained after careful tuning) yields less numerically robust function approximation results in our examples, as the backtracking line search fails most of the time. In particular, failure occurs frequently when the memory size is as large as 4. The sLBFGS method seems numerically more robust than the LBFGS in our experiments for a small memory size, but its performance depends heavily on selecting a good adaptive step size. For a large memory size, and with the adaptive step-size technique adopted in our experiments, sLBFGS tends to converge as slow as a first-order method, such as SGD. In summary, GRL($\hat{m}_r$) compares favorably overall with respect to the alternative state-of-the-art methods tested.

In Table III, we also compare our approach with the vanilla RNN and recently proposed RNN structures and training methods. In particular, we compare with two variants of the recently proposed dynamic recurrent routing neural networks (DRRNets) [63]: DRRRNN and DRRGRU. In order to have a feel of the low-rank regularization proposed in [63], which leads to a (convex-)cardinality problem, we trained the respective DRRNets with an ℓ_1 regularization (i.e., DRRNets + SGD + ℓ_1). Our choice of these variants for comparison is based on the results and discussions in the DRRNets paper. As shown, our approach produces smaller approximation error in fewer number of iterations and reduced runtime.

All parts of the experiments, including the RCNN models and the proposed algorithms, were implemented in Julia v1.7.2 on a laptop with 16 × 2.30 GHz Intel Core i7-11800H CPU and 16-GB RAM. Gradient and Jacobian computations were performed by Julia's open-source `ForwardDiff.jl` [64] package. The first-order algorithms used for comparison in Fig. 3 were provided by Julia's open-source `Flux.jl` [65], [66] package. The continuous dynamical system simulations inSection VI-B were performed within Julia's open-source `DynamicalSystems.jl` [67] library.

VII. CONCLUSION

In this work, we have proposed an efficient training approach for RCNNs, an RNN architecture for data-efficient RL and control. We presented an approximate Newton framework for training the state-space model of DCRNNs with convergence guarantee, through the lens of inexact SQP and numerical linear algebra techniques. The proposed GRL($\hat{m}_r$) algorithm efficiently addresses one of the main shortcoming associated with benchmark BPTT algorithms, viz., requiring excessive number of iterations to converge to the true solution. The method does this by exploiting approximate second-order information about the training data to speed up convergence, with minimal parameters to tune—we provided a detailed insight into the choice of the main parameter ω in our algorithm. Numerical examples have shown the efficiency of our proposed method.

In future works, we will explore the success of the proposed framework in more complex RL and control tasks. With control applications in focus, it would also be interesting to explore more creative and reliable ways to handle input and output constraints, for example, directly incorporating these constraints into the optimization framework proposed in this article. Another future research direction is in the use of more

TABLE IV
PARAMETERS USED IN THE ETHYLENE OXIDATION PROCESS [62]

Parameter	Value	Parameter	Value
γ_1	-8.13	B_1	7.32
γ_2	-7.12	B_2	10.39
γ_3	-11.07	B_3	2170.57
A_1	92.80	B_4	7.02
A_2	12.66	T_c	1.0
A_3	2412.71		

computationally efficient iterative schemes in the solution of the KKT subproblems where our algorithm spends most of its running time.

APPENDIX A
PROOF OF COROLLARY 2

From (28) and (27), we get

$$g_k^\top d_k^z + d_k^{z\top} H_k d_k^z + \left(\frac{2\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty\right) \|c_k - r_k^\lambda\|_1 - d_k^{z\top} r_k^z \le 0$$

and applying the reverse triangle inequality on $\|c_k - r_k^\lambda\|_1$ gives

$$\frac{2\rho_k}{\omega} - \|\tilde{\lambda}_k\|_\infty \ge \frac{g_k^\top d_k^z + d_k^{z\top} H_k d_k^z - d_k^{z\top} r_k^z}{\|c_k\|_1 - \|r_k^\lambda\|_1}$$

$$\rho_k \ge \frac{\omega}{2}\left[\frac{g_k^\top d_k^z + d_k^{z\top} H_k d_k^z - d_k^{z\top} r_k^z}{\|c_k\|_1 - \|r_\lambda\|_1} + \|\tilde{\lambda}_k\|_\infty\right].$$

APPENDIX B
PARAMETERS USED IN THE EXAMPLES

See Table IV.

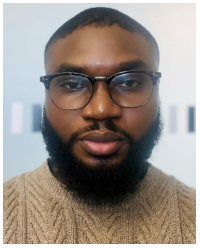
ACKNOWLEDGMENT

The authors thank Mario Zanon, IMT School for Advanced Studies Lucca, Lucca, Italy, for providing very useful comments on an initial version of this article, which helped to improve the overall presentation.

REFERENCES

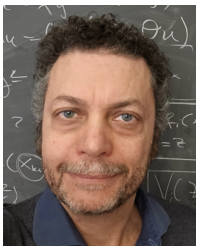
[1] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Uttar Pradesh, India: Pearson Education, 2009.
[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[4] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.
[5] J. Schmidhuber, "Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments," Dept. Comput. Sci., Tech. Univ. Munich, Munich, Germany, Tech. Rep. FKI-126-90, 1990.
[6] J. Schmidhuber, "An on-line algorithm for dynamic reinforcement learning and planning in reactive environments," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2, Jun. 1990, pp. 253–258.
[7] J. Schmidhuber, "Reinforcement learning in Markovian and non-Markovian environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 3, 1990.
[8] A. M. Schaefer, S. Udluft, and H.-G. Zimmermann, "A recurrent control neural network for data efficient reinforcement learning," in *Proc. IEEE Int. Symp. Approx. Dyn. Program. Reinforcement Learn.*, Apr. 2007, pp. 151–157.
[9] H.-G. Zimmermann, R. Grothmann, A. Schaefer, and C. Tietz, "Identification and forecasting of large dynamical systems by dynamical consistent neural networks," in *New Directions in Statistical Signal Processing: From Systems To Brain*. Cambridge, MA, USA: MIT Press, 2006, pp. 203–242.
[10] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[11] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *Int. J. Uncertainity, Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, 1998.
[12] E. P. dos Santos and F. J. Von Zuben, "Improved second-order training algorithms for globally and partially recurrent neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3, Jul. 1999, pp. 1501–1506.
[13] A. A. Vartak, M. Georgiopoulos, and G. C. Anagnostopoulos, "On-line Gauss–Newton-based learning for fully recurrent neural networks," *Nonlinear Anal., Theory, Methods Appl.*, vol. 63, nos. 5–7, pp. e867–e876, Nov. 2005.
[14] J. Martens and I. Sutskever, "Learning recurrent neural networks with Hessian-free optimization," in *Proc. ICML*, 2011.
[15] L.-W. Chan and C.-C. Szeto, "Training recurrent network with block-diagonal approximated Levenberg–Marquardt algorithm," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3, Jul. 1999, pp. 1521–1526.
[16] D. Mirikitani and N. Nikolaev, "Recursive Bayesian Levenberg–Marquardt training of recurrent neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2007, pp. 282–287.
[17] X. Fu, S. Li, M. Fairbank, D. C. Wunsch, and E. Alonso, "Training recurrent neural networks with the Levenberg–Marquardt algorithm for optimal control of a grid-connected converter," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 9, pp. 1900–1912, Sep. 2015.
[18] W.-F. Chang and M.-W. Mak, "A conjugate gradient learning algorithm for recurrent neural networks," *Neurocomputing*, vol. 24, nos. 1–3, pp. 173–189, Feb. 1999.
[19] A. Bemporad, "Training recurrent neural networks by sequential least squares and the alternating direction method of multipliers," *Automatica*, vol. 156, Oct. 2023, Art. no. 111183.
[20] M. M. Livstone, J. A. Farrell, and W. L. Baker, "A computationally efficient algorithm for training recurrent connectionist networks," in *Proc. Amer. Control Conf.*, Jun. 1992, pp. 555–561.
[21] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 4, Jun. 1992, pp. 241–246.
[22] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 279–297, Mar. 1994.
[23] X. Wang and Y. Huang, "Convergence study in extended Kalman filter-based training of recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 22, no. 4, pp. 588–600, Apr. 2011.
[24] T. Ergen and S. S. Kozat, "Efficient online learning algorithms based on LSTM neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3772–3783, Aug. 2018.
[25] A. Bemporad, "Recurrent neural network training with convex loss and regularization functions by extended Kalman filtering," *IEEE Trans. Autom. Control*, vol. 68, no. 9, pp. 5661–5668, Sep. 2023, doi: 10.1109/TAC.2022.3222750.
[26] A. Ororbia, A. Mali, C. L. Giles, and D. Kifer, "Continual learning of recurrent neural networks by locally aligning distributed representations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4267–4278, Oct. 2020.
[27] Y. Zhang, "Neural network algorithm with reinforcement learning for parameters extraction of photovoltaic models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 6, pp. 2806–2816, Jun. 2023, doi: 10.1109/TNNLS.2021.3109565.
[28] Z. Li and S. Li, "Neural network model-based control for manipulator: An autoencoder perspective," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 6, pp. 2854–2868, Jun. 2023, doi: 10.1109/TNNLS.2021.3109953.
[29] W. Cai, T. Wang, J. Wang, and C. Sun, "Learning a world model with multitimescale memory augmentation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 11, pp. 8493–8502, Nov. 2023, doi: 10.1109/TNNLS.2022.3151412.
[30] Q. Li, L. Chen, C. Tai, and E. Weinan, "Maximum principle based algorithms for deep learning," 2017, *arXiv:1710.09513*.
[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," La Jolla Inst. Cogn. Sci., San Diego, CA, USA, Tech. Rep. ADA164453, 1985.

[32] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: Unifying the algorithms and accelerating convergence," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 697–709, May 2000.
[33] A. D. Adeoye and A. Bemporad, "SCORE: Approximating curvature information under self-concordant regularization," *Comput. Optim. Appl.*, vol. 86, no. 2, pp. 599–626, Nov. 2023.
[34] A. Van Mulders, J. Schoukens, M. Volckaert, and M. Diehl, "Two nonlinear optimization methods for black box identification compared," *Automatica*, vol. 46, no. 10, pp. 1675–1681, Oct. 2010.
[35] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms 1. General considerations," *IMA J. Appl. Math.*, vol. 6, no. 1, pp. 76–90, 1970.
[36] R. Fletcher, "A new approach to variable metric algorithms," *Comput. J.*, vol. 13, no. 3, pp. 317–322, Mar. 1970.
[37] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Math. Comput.*, vol. 24, no. 109, pp. 23–26, 1970.
[38] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1970.
[39] M. J. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*. Berlin, Germany: Springer, 1978, pp. 144–157.
[40] M. J. D. Powell, "Algorithms for nonlinear constraints that use Lagrangian functions," *Math. Program.*, vol. 14, no. 1, pp. 224–248, Dec. 1978.
[41] M. J. D. Powell, "The convergence of variable metric methods for nonlinearly constrained optimization calculations," in *Nonlinear Programming*, 3rd ed. Amsterdam, The Netherlands: Elsevier, 1978, pp. 27–63.
[42] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, Jul. 1986.
[43] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM J. Numer. Anal.*, vol. 19, no. 2, pp. 400–408, 1982.
[44] R. H. Byrd, F. E. Curtis, and J. Nocedal, "An inexact SQP method for equality constrained optimization," *SIAM J. Optim.*, vol. 19, no. 1, pp. 351–369, Jan. 2008.
[45] R. H. Byrd, F. E. Curtis, and J. Nocedal, "An inexact Newton method for nonconvex equality constrained optimization," *Math. Program.*, vol. 122, no. 2, pp. 273–299, Apr. 2010.
[46] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific J. Math.*, vol. 16, no. 1, pp. 1–3, Jan. 1966.
[47] R. Choquet and J. Erhel, "Some convergence results for the Newton-GMRES algorithm," Ph.D. dissertation, INRIA, Rocquencourt, France, 1993.
[48] J.-F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization: Theoretical and Practical Aspects*. Berlin, Germany: Springer, 2006.
[49] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, vol. 14, pp. 1–137, May 2005.
[50] E. R. Panier and A. L. Tits, "Avoiding the Maratos effect by means of a nonmonotone line search I. General constrained problems," *SIAM J. Numer. Anal.*, vol. 28, no. 4, pp. 1183–1195, Aug. 1991.
[51] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia, PA, USA: SIAM, 1994.
[52] Y. Nesterov, *Lectures on Convex Optimization*, vol. 137. Cham, Switzerland: Springer, 2018.
[53] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1026–1034.
[54] S. K. Park, "A transformation method for constrained-function minimization," NASA Langley Res. Center, Hampton, VA, USA, Tech. Rep. L-10178 and NASA-TN-D-7983, 1975.
[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[56] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999.
[57] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 1999.
[58] A. Wills and T. Schön, "Stochastic quasi-Newton with adaptive step lengths for large-scale problems," 2018, *arXiv:1802.04310*.
[59] A. M. Schäfer, "Reinforcement learning with recurrent neural networks," Ph.D. dissertation, Dept. Math., Universität Osnabrück, Osnabrück, Germany, Oct. 2008.
[60] V. Heidrich-Meisner and C. Igel, "Variable metric reinforcement learning methods applied to the noisy mountain car problem," in *Proc. Eur. Workshop Reinforcement Learn.* Berlin, Germany: Springer, 2008, pp. 136–150.
[61] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.
[62] H. Durand, M. Ellis, and P. D. Christofides, "Economic model predictive control designs for input rate-of-change constraint handling and guaranteed economic performance," *Comput. Chem. Eng.*, vol. 92, pp. 18–36, Sep. 2016.
[63] D. Shan, Y. Luo, X. Zhang, and C. Zhang, "DRRNets: Dynamic recurrent routing via low-rank regularization in recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 2057–2067, Apr. 2023.
[64] J. Revels, M. Lubin, and T. Papamarkou, "Forward-mode automatic differentiation in Julia," 2016, *arXiv:1607.07892*.
[65] M. Innes et al., "Fashionable modelling with flux," 2018, *arXiv:1811.01457*.
[66] M. Innes, "Flux: Elegant machine learning with Julia," *J. Open Source Softw.*, vol. 3, no. 25, p. 602, May 2018.
[67] G. Datseris, "DynamicalSystems.Jl: A Julia software library for chaos and nonlinear dynamics," *J. Open Source Softw.*, vol. 3, no. 23, p. 598, Mar. 2018, doi: 10.21105/joss.00598.
[68] A. D. Adeoye and A. Bemporad, "Self-concordant smoothing for convex composite optimization," 2023, *arXiv:2309.01781*.



Adeyemi D. Adeoye received the bachelor's degree (Hons.) in mathematics from the University of Ilorin, Ilorin, Nigeria, in 2016, the master's degree in mathematical sciences from the African Institute for Mathematical Sciences, Limbe, Cameroon, in 2018, and the master's degree in machine intelligence from the African Institute for Mathematical Sciences, Kigali, Rwanda, in 2021. He is currently pursuing the Ph.D. degree with the Dynamical Systems, Control, and Optimization Research Unit, IMT School for Advanced Studies Lucca, Lucca, Italy.

His research interests include mathematical optimization, data-driven control, and neural networks.



Alberto Bemporad (Fellow, IEEE) received the master's degree (cum laude) in electrical engineering and the Ph.D. degree in control engineering from the University of Florence, Florence, Italy, in 1993 and 1997, respectively.

From 1996 to 1997, he was with the Center for Robotics and Automation, Department of Systems Science and Mathematics, Washington University, St. Louis, MO, USA. From 1997 to 1999, he held a postdoctoral position at the Automatic Control Laboratory, ETH Zürich, Zürich, Switzerland, where he collaborated as a Senior Researcher until 2002. From 1999 to 2009, he was with the Department of Information Engineering, University of Siena, Siena, Italy, where he became an Associate Professor in 2005. From 2010 to 2011, he was with the Department of Mechanical and Structural Engineering, University of Trento, Trento, Italy. He spent visiting periods at Stanford University, Stanford, CA, USA; University of Michigan, Ann Arbor, MI, USA; and Zhejiang University, Hangzhou, China. In 2011, he co-founded ODYS S.r.l., Lucca, Italy, a company specialized in developing model predictive control systems for industrial production. From 2012 to 2015, he served as the Director of the institute at the IMT School for Advanced Studies Lucca, Lucca, where he has been a Full Professor since 2011. He has published more than 400 papers in the areas of model predictive control, hybrid systems, optimization, and automotive control, and is the co-inventor of 21 patents. He has authored or coauthored various software packages for model predictive control design and implementation, including the Model Predictive Control Toolbox (The Mathworks Inc., Natick, MA, USA) and the Hybrid Toolbox for MATLAB.

Prof. Bemporad received the International Federation of Automatic Control (IFAC) High-Impact Paper Award for the 2011–2014 triennial, the IEEE Control Systems Society (CSS) Transition to Practice Award in 2019, and the 2021 Society of Automobile Engineers (SAE) Environmental Excellence in Transportation Award. He was the Chair of the Technical Committee on Hybrid Systems of the IEEE Control Systems Society from 2002 to 2010. He was an Associate Editor of IEEE TRANSACTIONS ON AUTOMATIC CONTROL from 2001 to 2004.

Open Access funding provided by 'Scuola IMT Alti Studi Lucca' within the CRUI CARE Agreement