2020 59th IEEE Conference on Decision and Control (CDC)
Jeju Island, Republic of Korea, December 14-18, 2020

Learning nonlinear feedback controllers from data via optimal policy search and stochastic gradient descent

Laura Ferrarotti* and Alberto Bemporad*

***Abstract*— This paper proposes a technique for synthesizing smooth nonlinear controllers by optimal policy search and stochastic gradient descent. After choosing an appropriate parameterization of the control law, mini-batch stochastic gradient descent steps are used to iteratively optimize the parameters of the control law. The gradients of the expected future closed-loop performance required for the descent are approximated by using simple local linear models, as introduced earlier by the authors for optimal policy search of linear feedback controllers. In this way, the method does not require a full nonlinear model of the process. The algorithm can be applied offline, on a previously collected dataset, or online, while controlling the plant itself with the most updated policy. We apply the method in a numerical example in which we solve an output-tracking problem for a Continuously Stirred Tank Reactor (CSTR) using a neural-network parameterization with differentiable activation function of the controller. In the offline setting the performance of the resulting neural controller is compared to the one of a linear feedback controller trained on the same dataset. In the online setting, instead, we show how the learning procedure can be designed, combining on-policy and off-policy learning, to increase safety and improve performance.**

I. Introduction

Data-driven control has been recently investigated as a viable alternative to classic model-based control [1], that often require rather time-consuming identification procedures to get an open-loop model of the process [2]. Among others, data-driven control techniques based on Reinforcement Learning (RL) exploit experience gathered from process/environment interactions, see [3]–[6]. Following the RL taxonomy, the methods can be divided in actor-only, critic-only, and actor-critic. Critic-only and actor-critic methods rely on approximations of the value function [7], [8]. Actor-only methods, also called policy-search methods, instead, do not require such an approximation, as they work with a parameterized family of policies and directly optimize the policy parameters [9]. In particular, policy-search methods that optimize the parameterized policies following the direction indicated by the gradient of the cost-function are denoted as policy-gradient methods (see [10], [11]).

Actor-only methods, critic-only methods, and actor-critic methods were recently combined with the use of neural networks (NNs), exploited for their extreme flexibility and generalization capability. NNs are powerful nonlinear function approximators, composed by a sequence of layers of "neurons", that result in the composition of a linear function, associated with the weights W_i and bias b_i, and a nonlinear activation function σ, i.e., $f_i(x) = \sigma(W_i\, x + b_i)$. The result of combining NNs with RL is indicated as Deep RL and is the subject of several recent publications [12]. An important example is the Deep Q-learning algorithm (DQN) [13], [14], that adapts the Q-learning algorithm [7] in order to make effective use of large NNs as function approximators. DQN saw many attempts of improvement, surveyed and combined in [15], to obtain an algorithm with improved performance. Another milestone in the field is represented by the Deep Deterministic Policy Gradient (DDPG) algorithm [16], an actor-critic, model-free method based on the deterministic policy gradient [17], that can operate over continuous action spaces. Attempts were also made to combine deep policy learning and deep Q-learning, as in [18], where the Q-values are estimated from the action preferences of the policy, and Q-learning updates are applied to the policy. In [19], instead, a mix of policy gradient and actor-critic updates is shown to achieve promising results, together with the use of a parameterized family of policy gradient methods that interpolate between on-policy and off-policy learning.

In this paper we focus on synthesizing nonlinear smooth controllers via mini-batch stochastic gradient descent optimization of the control policy parameters. The gradients of a given closed-loop performance index required for the descent are approximated using simple local linear open-loop models, so that the method does not require a full model of the plant from which input/output data are collected. The approach, here described in its most general setup, was introduced in [20], where it was tailored for the learning of linear policies for output-tracking. A heuristic for assisted input selection during online learning is introduced in this paper, to cope with the risk of implementing the policy directly on the plant, without loosing the advantages of an on-policy approach. A different extension of [20] was recently proposed in [21] to support piecewise-affine policy parameterizations, in which both the linear control policies and the switching law are learned from data. To show the performance of the approach proposed in this paper for the synthesis of nonlinear smooth controllers, we provide a simulation example in which an output-tracking problem for a Continuously Stirred Tank Reactor (CSTR) is solved by training a NN-parameterized policy by SGD. The paper is organized as follows: the optimal policy search problem is formulated in Section II and an optimal policy search algorithm is described in Section III. Section IV details the algorithmic setup for solving nonlinear output tracking problems, which is used in Section V to show numerical results. Section VI summarizes and concludes the paper.

* IMT School for Advanced Studies Lucca, P.za San Francesco 19, 55100 Lucca (IT), laura.ferrarotti@imtlucca.it, alberto.bemporad@imtlucca.it

978-1-7281-7446-4/20/$31.00 ©2020 IEEE

Notation: Let $\mathbb{R}^n$ be the set of real vectors of dimension n. For each vector x belonging to $\mathbb{R}^n$, x_i is the i-th element of the vector. Given a matrix $A \in \mathbb{R}^{n\times m}$ we denote its transpose by A'. We denote by I the identity matrix and by e_i its i-th column. Given a matrix $Q \in \mathbb{R}^{n\times n}$, $\|x\|_Q^2 = x'\,Q\,x$. With $\text{rem}(x, y)$ we indicate the reminder of the division of x by y.

II. Optimal policy search

Let $s_t \in \mathbb{R}^{n_s}$ be a Markovian signal collecting the variables that describe the dynamics of the strictly causal plant P to be controlled and its interactions with the environment. The components of s_t include the state vector x_t of P and may also include states required by the control policy, for example the integral of tracking errors. In this paper we focus on input/output models, that is x_t is a collection of a finite number of past samples of the input u_t and output y_t of the plant, although the approach could be immediately extended to state-space models. The temporal evolution of s_t can be described by the (unknown) model

$$s_{t+1} = f(s_t, p_t, u_t, d_t), \tag{1}$$

which captures the dependence of s_t on the decision variable (action) $u_t \in \mathbb{R}^{n_u}$, on unmeasured disturbances $d_t \in \mathbb{R}^{n_d}$, and on a vector $p_t \in \mathbb{R}^{n_p}$ of non-Markovian exogenous signals, such as measured disturbances, time-varying parameters, and reference signals to track.

The cost of applying an action u_t to P while in $(s_t,\, p_t)$ is quantified by the *stage cost* $\rho(s_t, p_t, u_t)$, where $\rho : \mathbb{R}^{n_s+n_p+n_u} \to \mathbb{R}$. Given the values $p_0,\, d_0,\, p_1,\, d_1, \ldots$ and the initial condition s_0, the cost of applying a sequence of control actions $u_0,\, u_1, \ldots$ over an infinite horizon can be obtained by summing up the stage costs along the trajectory produced by (1)

$$J_\infty(s_0, \{p_\ell, d_\ell, u_\ell\}_{\ell=0}^{\infty}) = \sum_{\ell=0}^{\infty} \rho(s_\ell, p_\ell, u_\ell). \tag{2}$$

Our aim is to design an optimal nonlinear feedback controller generating the command inputs $u_0, u_1, \ldots$ to the plant P so to minimize the closed-loop performance index (2), for every possible sequence $\{p_\ell,\, d_\ell \,|\, \ell = 0, 1, \ldots\}$ and initial condition s_0, without an explicit knowledge of model f in (1).

We represent the controller as a deterministic policy $\pi : \mathbb{R}^{n_s+n_p} \to \mathbb{R}^{n_u}$ that associates to each s_t and p_t an action $u_t = \pi(s_t, p_t)$. To evaluate the performance of a policy π, we derive from (2) the cost

$$J(\pi) = \mathbb{E}[J_\infty(S_0,\, \{P_\ell,\, D_\ell,\, \pi(S_\ell,\, P_\ell)\}_{\ell=0}^{\infty})], \tag{3}$$

where the expectation of J_∞ is taken with respect to the random variables S_0 and P_ℓ, D_ℓ, $\ell = 0, 1, \ldots$, representing all the possible initial states of the trajectory and values of the signals p_ℓ, d_ℓ at step ℓ, respectively.

Based on (3), the optimal feedback controller corresponds to the policy π^* such that

$$\pi^* = \arg \min_{\pi \in \mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})} J(\pi) \tag{4}$$

where $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ is the set of functions of $n_s + n_p$ real variables taking values in $\mathbb{R}^{n_u}$. Equation (4) represents a general abstract optimal policy search (OPS) problem.

Problem (4) is computationally and theoretically difficult to tackle, due to $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ being completely generic. For this reason, we restrict the shape of the controller we want to synthesize by parameterizing the policy with a vector $H \in \mathbb{R}^h$ of parameters to select, and denote the resulting policy by $\pi_H(s_t, p_t)$. The optimization problem (4) is therefore transformed into

$$H^* = \arg \min_{H \in \mathbb{R}^h} J(\pi_H). \tag{5}$$

We consider nonlinear policy parameterizations, that is $\pi_H(s_t, p_t) = g(s_t, p_t, H)$, where g is a nonlinear function, differentiable with respect to H for every (s_t, p_t) in $\mathbb{R}^{n_s+n_p}$. Problem (5) still involves the optimization of the expected value of an infinite horizon cost. We therefore approximate it with a finite trajectory cost of length L. The resulting cost of π_H , given s_0, $\{p_\ell \,|\, \ell = 0, \ldots, L\}$ and $\{d_\ell \,|\, \ell = 0, \ldots, L-1\}$ is

$$\begin{aligned} &J_L(\, H,\, s_0,\, \{p_\ell\}_{\ell=0}^{L},\, \{d_\ell\}_{\ell=0}^{L-1}\,) = \\ &\quad = \sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, \pi_H(s_\ell, p_\ell)) + \rho_L(s_L, p_L). \end{aligned} \tag{6}$$

Then, the considered approximate OPS problem is

$$H^* = \arg \min_{H \in \mathbb{R}^h} \mathbb{E}[J_L(H,\, S_0,\, \{P_\ell\}_{\ell=0}^{L},\, \{D_\ell\}_{\ell=0}^{L-1})], \tag{7}$$

where the expected value is taken with respect to the random variables S_0, $\{P_\ell\}_{\ell=0}^{L}$, $\{D_\ell\}_{\ell=0}^{L-1}$.
Stochastic optimization methods can be employed to optimize an expected value. In this paper we use the mini-batch Stochastic Gradient Descent (SGD) algorithm [22]. The application of mini-batch SGD to problem (7) requires at every step t computing the gradients $\nabla J_L(H_{t-1}, s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1})$ for each $(s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1})$ in the current mini-batch of data. Function J_L, though, depends on (1), and so does its gradient. Since in our setup the dynamics (1) are unknown, for each evaluation of $\nabla J_L(H_{t-1}, s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1})$ we replace (1) with a local linear model that approximates the behaviour of the system in a neighborhood of the initial point s_0^i, as in [20]. We indicate the gradient computed using the local linear model as $\nabla \hat{J}_L(H_{t-1}, s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1})$.

III. Optimal policy search algorithm

In this section we generalize the optimal policy search algorithm introduced in [20] to nonlinear policies. This algorithm employs mini-batch SGD iterations to attempt solving (7), given the initial value H_{-1} of the parameters.

We consider two possible settings: an *offline* setting, in which π_H is synthesized from open-loop data previously collected from the plant, and an *online* setting, in which new data are collected from the plant during the iterative policy synthesis, and employed in further SGD iterations. In both settings, the t-th iteration requires one to approximate the gradients $\{\nabla J_L(H_{t-1}, s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1})\}_{i,j,k}$.

Being J_L dependent on the unknown dynamics (1), a set of local linear models $\{\Theta_i\}_i$, each of them fitted in a neighborhood of the associated sampled initial state s_0^i, is employed. This local linear model is obtained recursively, as described in Section III-A. The sampling procedure of the mini-batches is summarized in Section III-B and the gradient approximation and policy update method is detailed in Section III-C. A summary of the overall proposed method is given in Algorithm 1.

Algorithm 1 Optimal nonlinear policy search

Input: Initial policy parameterization H_{-1}, number N_{learn} of learning steps.
Online setting: Initial state s_0 and exogenous signal p_0, history $X = \emptyset$ of P, initial local linear model Θ_0
Offline setting: state trajectory $X_N = \{x_0, .., x_{N-1}\}$ and associated local linear models $\Theta = \{\Theta_0, .., \Theta_{N-1}\}$

Output: Policy parameter vector H_{SGD}.

for $t = 0, \dots, N_{\text{learn}} - 1$ **do**
  online setting: build x_t as in (11)
    $X \leftarrow X \cup \{x_t\}$;
    $u_t \leftarrow \pi_{H_t}(s_t, p_t)$;
    apply u_t to the plant;
    collect y_{t+1};
    Update Θ_{t+1} from Θ_t (see Sec. III-A)
    $\Theta \leftarrow \Theta \cup \{\Theta_{t+1}\}$;
  for $h = 1, 2, ..., N_b$ **do**
    sample $w_h = (s_0^i, \{p_\ell^j\}_{\ell=0}^{L}, \{d_\ell^k\}_{\ell=0}^{L-1})$;
    retrieve model coefficients Θ_i associated with s_0^i;
    calculate $\nabla_H \hat{J}_L(H_{t-1}, w_h)$;
  end for

$$\mathcal{D}(H_{t-1}) \leftarrow \frac{1}{N_b} \sum_{h=0}^{N_b} \nabla_H \hat{J}_L(H_{t-1}, w_h); \tag{8}$$

  Policy update:

$$H_t = H_{t-1} - \alpha_t \mathcal{D}(H_{t-1}); \tag{9}$$

  online setting: build s_{t+1} and measure p_{t+1};
end for
$H_{\text{SGD}} \leftarrow H_{N_{\text{learn}}-1}$;
end.

A. Local model

To evaluate the gradient $\nabla_H \hat{J}_L$ in (8) without knowing the dynamics (1), we use the local linear model

$$y_{t+1} = \Theta_{t+1} \begin{bmatrix} x_t \\ u_t \end{bmatrix} + d_t \tag{10}$$

where

$$x_t = [y_t' \ \dots \ y_{t-n_o+1}' \ u_{t-1}' \ \dots \ u_{t-n_i+1}']', \tag{11}$$

and $\Theta_t \in \mathbb{R}^{n_y \times (n_x + n_u)}$ is estimated recursively, $n_x = n_o n_y + (n_i - 1) n_u$. We use Kalman filtering (KF) for updating Θ_t, assuming that $\Theta_{t+1} = \Theta_t + \xi_t$ where ξ_t is Gaussian white noise with covariance Q_k and d_t in (10) is a Gaussian white noise with covariance matrix R_k.

B. Sampling procedure

At time instant t, the iterative optimization procedure starts with sampling a mini-batch $\{w_h = (s_0^i, \{p_\ell^j\}_{\ell=0}^{L}, \{d_\ell^k\}_{\ell=0}^{L-1})\}_{h=1}^{N_b}$ of initial states, exogenous signals, and disturbances. We adopt the sampling procedure suggested in [20], summarized next.

We first sample the initial state s_0^i: this vector is composed by a vector x_0^i representing the initial state of the plant P, and possibly by additional states that one wants to include in the control policy, denoted by vector z_0^i. The state history of P is collected in the set X. In the *online* setting at time instant t, the history X is the set of all the states x_t visited by the plant up to t. In the *offline* setting, instead, X contains N states obtained from a previous open-loop data collection session.

To obtain the values s_0^i, we sample a set of N_0 states from the history X and perturb them by a (small) random noise v_n from a given set V to explore the neighborhood of the trajectory collected in X. Combining it with N_z realizations z_0^m of the chosen additional states, randomly sampled from a given set Z, we form a set of $N_s = N_0 \, N_z$ initial states

$$s_0^i = s_0^{n,m} = \begin{bmatrix} x_0^n + v_n \\ z_0^m \end{bmatrix}, \tag{12}$$

for $n = 1, \dots, N_0$ and $m = 1, \dots, N_z$.

The exogenous signals $\{p_\ell^j\}_{\ell=0}^{L}$ are randomly generated from $[p_{\min}, p_{\max}]^{L+1}$ for $j = 1, \dots, N_p$. Analogously, the disturbance trajectories $\{d_\ell^k\}_{\ell=0}^{L-1}$ are randomly generated for $k = 1, \dots, N_d$, from a given interval $[-d_{\max}, d_{\max}]^L$. All the possible combinations of the sampled states, exogenous signals, and disturbances are built; each of them becomes an element $w_h = w_{i,j,k} = (s_0^i, \{p_\ell^j\}_{\ell=0}^{L}, \{d_\ell^k\}_{\ell=0}^{L-1})$ of the mini-batch of cardinality $N_b = N_s \, N_p \, N_d$.

C. Policy update and online implementation

The direction used to update H_t at step t is given by (8). To evaluate $\nabla_H \hat{J}_L(H_{t-1}, w_h)$, the local model Θ_i associated with the initial state s_0^i specified in $w_h = (s_0^i, \{p_\ell^j\}_{\ell=0}^{L}, \{d_\ell^k\}_{\ell=0}^{L-1})$ for each h is used. The policy update performed by the mini-batch SGD algorithm with learning rate α_t using direction (8) is given by (9).

In the *online setting*, the current policy is also applied to P by setting $u_t = \pi_{H_t}(s_t, p_t)$. Alternatively, a safer but possibly suboptimal policy π_b (called *behavioral* policy) could be employed to drive the system for data collection, while π_{H_t} is optimized (see [23], [24]).

IV. OPS FOR OUTPUT TRACKING

In Section III, the optimal policy search algorithm has been described in its most general form. To illustrate its performance, we apply it to an output-tracking task. To learn a nonlinear policy that makes the output y_t of the plant P track a reference signal r_t from input/output data, we consider $s_t = [\ x_t', \quad q_t'\ ]', \quad p_t = r_t$, where the state s_t is composed by x_t as in (11) and by the integral term q_t,

Algorithm 2 Input selection during online learning with assisted control of the plant

```
on-policy(t) ← False;
if t > T_2
    on-policy(t) ← True;
else
    if t ≥ T_1
        if rem(t, M) = 0
            u_H ← π_{H_t}(s_t, r_t);
            y_pred = Θ_t [ x_t ; u_H ];
            err = ‖y_pred − r_t‖_∞;
            if err ≤ ε
                on-policy(t) ← True;
        else
            on-policy(t) ← on-policy(t − 1)
if on-policy(t)
    u_t ← π_{H_t}(s_t, r_t);
else
    u_t ← π_b(s_t, r_t);
```

$q_t = q_{t-1} + (y_t - r_t)$. To achieve the output-tracking task we consider the following stage cost and terminal cost

$$\rho(s_t,\, r_t,\, u_t) = \|y_t - r_t\|^2_{Q_y} + \|\Delta u_t\|^2_R, \tag{13a}$$

$$\rho_L(s_L,\, r_L) = \|y_L - r_L\|^2_{Q_y}, \tag{13b}$$

where y_t is a component of x_t, and therefore of s_t, and $\Delta u_t = u_t - u_{t-1}$ is the input increment. Here we assume $n_i \geq 2$, so u_{t-1} is contained in x_t, as defined in (11), and in s_t. In case $n_i = 1$, it is nonetheless possible to add u_{t-1} as additional state in s_t. For offset-free tracking of constant set-points in steady-state we add the penalty $\|q_t\|^2_{Q_q}$ in (13). By treating the reference as an exogenous signal, the N_p reference trajectories $\{r^j_\ell\}_{\ell=0}^{L-1}$ are sampled such that $r^j_\ell = r^j$, with r^j randomly taken from the interval $[r_{\min}, r_{\max}]$ of references of interest.

The online implementation of the optimal policy search algorithm can benefit from the use of a behavioral policy π_b, as expressed by the steps summarized in Algorithm 2, that substitutes the computation of u_t in Step 2 of Algorithm 1. In Algorithm 2, π_b is used in a first phase of *off-policy* learning, for $t \in \{0, \ldots, T_1 - 1\}$, to generate the input to the plant, to track the reference r_t, and to collect input/output data for the ongoing optimization of H_t. After T_1 steps, online learning with assisted control of the plant is performed, for $t \in \{T_1, \ldots, T_2\}$: every $M \geq 1$ steps, the input $u_t = \pi_{H_t}(s_t, r_t)$ is generated from the current policy π_{H_t}. Then, the result of applying u_t to the plant is predicted using the last updated local model Θ_t. If the predicted output is sufficiently close to the desired set-point value r_t, we assign the control of the plant for the next M iterations to π_{H_t} while it is being optimized. Otherwise the task is performed by the behavioral policy π_b for M steps. After T_2 steps, learning is performed completely in *on-policy* setting, the policy π_{H_t} being employed at every successive iteration.

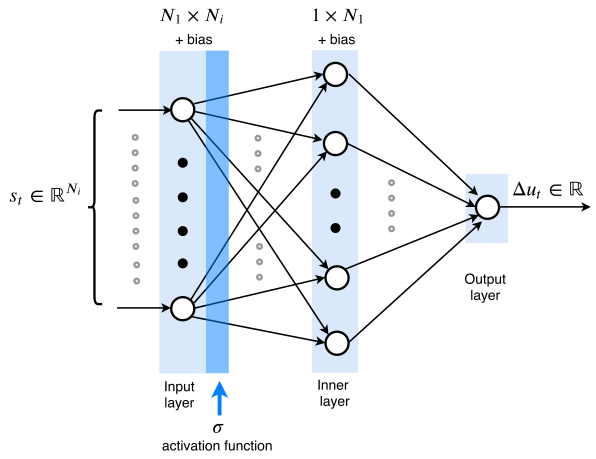

Fig. 1. Neural network architecture. In our case $N_i = 9$, $N_1 = 10$ and the activation function σ is the swish function

V. Numerical results

We consider the classical Continuous Stirred Tank Reactor (CSTR) benchmark problem [25]. We denote as C_A and T the concentration of reactant A and the temperature inside the tank, respectively. The signal T_c is the temperature of the cooling jacket. The signals are discretized with sampling time $T_s = 6$ s. The control objective is to optimally make C^t_A track a desired set point r_t by manipulating the temperature T^t_c of the cooling jacket. Our goal is to satisfy the control objective by learning and employing a nonlinear control policy, synthesized as described in Section III, treating the plant as a "black box", producing input/output data. The nonlinear parameterization of the control policy is $u_t = u_{t-1} + \mathcal{N}(s_t, r_t)$, where $\mathcal{N}$ is a neural network with two fully connected layers generating Δu_t. The first layer has $N_i = 9$ neurons with nonlinear differentiable swish activation function $\sigma(x) = \frac{x}{1+e^{-x}}$, the second layer is linear with $N_1 = 10$ neurons. The neural network used in this example is described in Figure 1.

The RMSProp algorithm [26], a fast version of SGD, is used for the learning, with $\beta_1 = 0.9$ and a decreasing sequence of learning rates. We consider the scaled signals

$$y^C_t = \frac{\bar{C}^t_A - c_m}{c_M - c_m}, \quad y^T_t = \frac{\bar{T}_t - t_m}{t_M - t_m}, \quad u_t = \frac{T^t_c - u_m}{u_M - u_m},$$

as input/output data, where $[c_m,\, c_M] = [0,\, 10]$, $[t_m,\, t_M] = [300,\, 400]$, $[u_m,\, u_M] = [240,\, 360]$. The quantities $\bar{C}^t_A$ and $\bar{T}_t$ represent the noisy measurements of C^t_A and T_t, considering a Gaussian white noise with standard deviation 0.01 units for both signals. The stage-cost (13) is used, with weights

$$Q_y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 0.1, \quad Q_q = \begin{bmatrix} 0.009 & 0 \\ 0 & 0 \end{bmatrix}.$$

Model (10) is taken as a local linear model, without considering the disturbance ($d_t = 0$).

A. Offline learning

The algorithm is run in offline mode on a dataset of $N = 29990$ samples, for $N_{\text{learn}} = 10000$ iterations. Table I

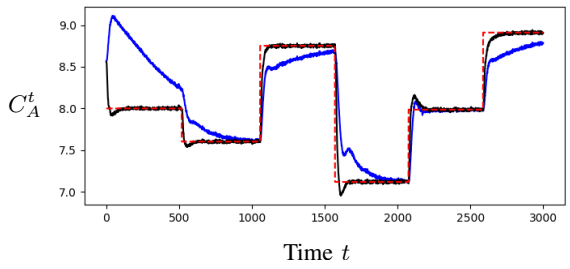

Fig. 2. Tracking task performed in validation by the nonlinear neural policy (black line) and the linear policy (blue line) on a piecewise constant task expressed by the reference signal r_t (red dashed line).

TABLE I

POLICY SEARCH OFFLINE PARAMETERS

n_y	n_u	n_i	n_o	Q_k	R_k	L
2	1	2	3	I	0.1	10
N_0	N_p	N_z	$r_{\min}$	$r_{\max}$	Z	V
50	20	20	1.5	9.5	$[0, 10]$	$[0, 0.1]$

contains the policy search parameters used in this example.

Using the same dataset, parameters, and number of iterations, we train a linear policy $\pi_K(s_t, r_t) = K_s s_t + K_r r_t$ using the Adam algorithm [27], another fast version of SGD, with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\alpha = 0.001$.

Both the nonlinear and the linear controller learning procedures are performed starting from random initial parameters H_{-1} and (K_s^{-1}, K_r^{-1}), sampled from a Gaussian random variable with zero mean and standard deviation 0.0001. The closed-loop performance of the resulting policies H_{learn} and (K_s, K_r) is compared in validation, while performing a piecewise tracking task of $N_{\text{valid}} = 3000$ steps. The results are shown in Figure 2. Considering the sum of the stage costs $\rho(s_t, r_t, u_t)$ defined as in (13) over N_{valid} steps as performance index, the cost obtained using the linear policy is 4.319 while, using the neural network is 0.647. As it is visible from the plots and the costs, the neural network is more efficient than the linear parameterization in dealing with the nonlinearity of the plant.

B. Online learning

We use now the parameters H_{learn} learnt offline to warm-start a new learning phase of $N_{\text{learn}} = 15000$ steps, in which the optimization of the neural network obtained offline is refined online, while controlling the plant, i.e., we take $H_{-1} = H_{\text{learn}}$. In this case we sample the initial values $q_0^i \sim \mathcal{N}(0, \sigma_q^2)$ of the integral action as well as the perturbations on the states $v_n \sim \mathcal{N}(0, \sigma_x^2)$. Table II contains the parameters tuned for the online learning. The procedure described in Algorithm 2 is followed to control the plant online and collect new data, using $M = 10$

TABLE II

POLICY SEARCH ONLINE PARAMETERS

n_y	n_u	n_i	n_o	Q_k	R_k	L
2	1	2	3	I	0.1	10
N_0	N_p	N_z	$r_{\min}$	$r_{\max}$	σ_q	σ_x
32	20	1	4	9.5	5	0.01

TABLE III

ONLINE LEARNING, PERFORMANCE INDEX

	cost $0 \le t \le T_2$	cost $T_2 < t \le N_{\text{learn}}$
π_{H_t}	47.768	34.359
π_b	23.922	60.033

and $\epsilon = 0.3$. The policy trained offline is employed as behavioral policy, i.e., $\pi_b = \pi_{H_{\text{learn}}}$. The tracking of a reference signal r_t is performed at time-instant t by the current policy π_{H_t}, with the assistance of the behavioral policy for the first 5000 iterations, and independently for the remaining steps ($T_1 = 0, T_2 = 5000$). The top plot of Figure 3 shows the reference r_t and the concentration values obtained by controlling the plant with the described algorithm. The gradual improvement of π_{H_t} in tracking is visible throughout the online learning, as well as the final improvement with respect to the performance obtained by using $\pi_b = \pi_{H_{\text{learn}}}$ on the same tracking task, presented in the bottom plot. The costs of the online task are shown in Table III. The first column contains the cost cumulated during steps $0 \le t \le T_2$, corresponding to the assisted control phase for π_{H_t}, while the second column shows the cost corresponding to the remaining steps $T_2 < t \le N_{\text{learn}}$, where π_{H_t} is applied. In both intervals we consider the sum of the stage costs $\rho(s_t, r_t, u_t)$ defined as in (13) as performance index. We see that controlling the plant using the assisted control procedure is more expensive than simply using π_b but it pays off in the sense of learning, permitting one to refine the initial policy and obtain a sensibly better performance in the long run.

VI. CONCLUSIONS

We have shown that the proposed optimal policy-search algorithm is effective for learning nonlinear smooth feedback control laws from data. Although we have used neural networks to define the structure of the policy, the approach is applicable to any other differentiable nonlinear architectures. Thanks to using simple stochastic gradient descent iterations, the identification of a full nonlinear model of the process dynamics is avoided, as simple local linear models, recursively identified with forgetting factor, are enough to compute the gradients required at every step of the descent. The reported example shows that the method is able to learn nontrivial nonlinear policies, such as neural networks, both in offline and online setting. In the online case the method permits one to delegate the exploration of the state space to the reference signal used during the learning phase, making the design of the experiment required to collect data relatively easy. Moreover, the combination of on-policy and off-policy learning permits on one hand to base the parameters optimization on data belonging to the trajectory induced by the current policy of the agent, and on the other hand to mitigate the risk of implementing the policy directly on the plant, particularly during the initial phases of the learning. Current research is devoted to extending the approach in several directions, including the extension of the method to handle constraints, and the study of new frameworks that can enforce stability and/or safety. Additionally, the

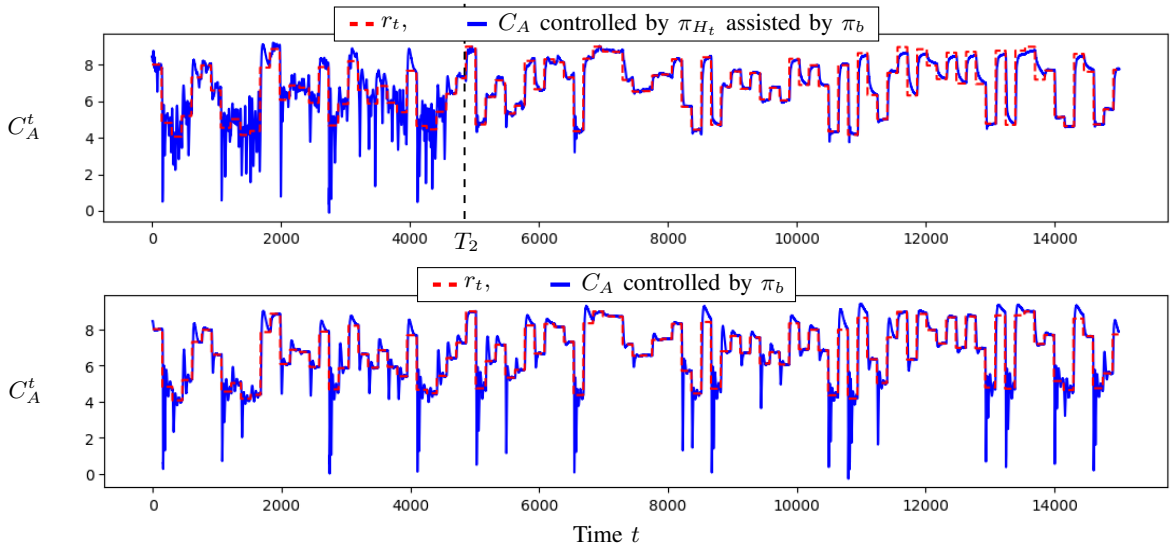

Fig. 3. Online tracking task. In both plots the task is expressed by the online reference signal r_t (red dashed line). The top plot presents the tracking performed by the policy π_{H_t} with assistance of the behavioral policy $\pi_b = \pi_{H_{\text{learn}}}$, following the scheme proposed in Algorithm 2 (with $T_1 = 0$, $T_2 = 5000$, $M = 10$ and $\epsilon = 0.3$). In the bottom plot, instead, $\pi_b = \pi_{H_{\text{learn}}}$ is implemented at every instant on the plant.

training of nonlinear parameterizations in a collaborative learning framework using the presented approach is under investigation.

REFERENCES

[1] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, p. 335, 06 2013.

[2] S. Formentin, K. Heusden, and A. Karimi, "A comparison of model-based and data-driven controller tuning," *International Journal of Adaptive Control and Signal Processing*, vol. 28, 10 2014.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[4] B. Recht, "A Tour of Reinforcement Learning: The View from Continuous Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, May 2019.

[5] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using Reinforcement Learning: a survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 2042–2062, Jun 2018.

[6] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[7] C. J. C. H. Watkins and P. Dyan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[8] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal of Optimal Control*, vol. 42, pp. 1143–1166, 2003.

[9] M. P. Desienroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011.

[10] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist Reinforcement Learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[11] J. Peters and S. Schaal, "Reinforcement Learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, May 2008.

[12] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing ATARI with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[15] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *CoRR*, vol. abs/1710.02298, 2017.

[16] T. P. Lillicrap, J. H. Jonathan, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, 22–24 Jun 2014.

[18] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "PGQ: combining policy gradient and Q-learning," *CoRR*, vol. abs/1611.01626, 2016.

[19] S. S. Gu, T. Lillicrap, R. E. Turner, Z. Ghahramani, B. Schölkopf, and S. Levine, "Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning," in *Advances in Neural Information Processing Systems 30*, pp. 3846–3855, 2017.

[20] L. Ferrarotti and A. Bemporad, "Synthesis of optimal feedback controllers from data via stochastic gradient descent," in *Proceedings of the 18th European Control Conference (ECC)*, pp. 2486–2491, June 2019.

[21] L. Ferrarotti and A. Bemporad, "Learning optimal switching feedback controllers from data," in *Proceedings of the 21st IFAC World Congress, 2020*, to appear.

[22] H. Robbins and S. Monoro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[23] L. Busoniu, T. de Bruin, D. Toli, J. Kober, and I. Palunko, "Reinforcement Learning for Control: performance, stability, and deep approximators," *Annual Reviews in Control*, Oct 2018.

[24] J. García and F. Fernández, "A comprehensive survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[25] D. Seborg, T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control, 2nd ed.* Wiley, 2004.

[26] T. Tieleman and G. Hinton, "Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[27] D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," in *Proc. International Conference on Learning Representation*, (San Diego, CA, USA), May 7-9 2015.