Received: 2 January 2019 | Revised: 23 May 2019 | Accepted: 28 May 2019

DOI: 10.1002/aic.16689

PROCESS SYSTEMS ENGINEERING

AIChE JOURNAL

Toward self-driving processes: A deep reinforcement learning approach to control

Steven Spielberg[1] | Aditya Tulsyan[1] | Nathan P. Lawrence[2] | Philip D. Loewen[2] | R. Bhushan Gopaluni[1]

[1]Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, British Columbia, Canada

[2]Department of Mathematics, University of British Columbia, Vancouver, British Columbia, Canada

Correspondence
R. Bhushan Gopaluni, Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, British Columbia BC V6T 1Z3, Canada.
Email: bhushan.gopaluni@ubc.ca

Funding information
Natural Sciences and Engineering Research Council of Canada

Abstract

Advanced model-based controllers are well established in process industries. However, such controllers require regular maintenance to maintain acceptable performance. It is a common practice to monitor controller performance continuously and to initiate a remedial model re-identification procedure in the event of performance degradation. Such procedures are typically complicated and resource intensive, and they often cause costly interruptions to normal operations. In this article, we exploit recent developments in reinforcement learning and deep learning to develop a novel adaptive, model-free controller for general discrete-time processes. The deep reinforcement learning (DRL) controller we propose is a data-based controller that learns the control policy in real time by merely interacting with the process. The effectiveness and benefits of the DRL controller are demonstrated through many simulations.

KEYWORDS

actor–critic networks, deep learning, model-free learning, process control, reinforcement learning

1 | INTRODUCTION

Industrial process control is a large and diverse field; its broad range of applications call for a correspondingly wide range of controllers—including single and multiloop proportional integral and derivative (PID) controllers, model predictive controllers (MPCs), and a variety of nonlinear controllers. Many deployed controllers achieve robustness at the expense of performance. The overall performance of a controlled process depends on the characteristics of the process itself, the controller's overall architecture, and the tuning parameters that are employed. Even if a controller is well tuned at the time of installation, a drift in process characteristics or deliberate set-point changes can cause performance to deteriorate over time.[1-3] Maintaining system performance over a long term is essential. Unfortunately, it is typically also both complicated and expensive.

Most modern industrial controllers are model based, so good performance calls for a high-quality process model. It is a standard practice to continuously monitor system performance and initiate a remedial model re-identification exercise in the event of performance degradation. Model re-identification can require 2 weeks or more,[4] and typically involves the injection of external excitations,[5] which introduce an expensive interruption to the normal operation of the process. Re-identification is particularly complicated for multivariable processes, which require a model for every input–output combination.

Most classical controllers in industry are linear and nonadaptive. Although extensive work has been done in nonlinear adaptive control,[6,7] it has not yet established a significant position in the process industry, beyond several niche applications.[8,9] The difficulty of learning (or estimating) reliable multivariable models in an online fashion is partly responsible for this situation. Other contributing factors include the presence of hidden states, process dimensionality, and in some cases computational complexity.

Given the limitations of existing industrial controllers, we seek a new design that can learn the control policy for discrete-time nonlinear stochastic processes in real time, in a model-free and adaptive environment. This article continues our recent investigations[10] on

https://doi.org/10.1002/aic.16689

the same topic. The idea of reinforcement learning (RL) has been around for several decades; however, its application to process control has been somewhat recent. Next, we provide a short introduction to RL and its methods and also highlight existing RL-based approaches for control problems.

1.1 | RL and process control

RL is an active area of research in artificial intelligence. It originated in computer science and operations research to solve complex sequential decision-making problems.[11-14] The RL framework comprises an agent (e.g., a controller) interacting with a stochastic environment (e.g., a plant) modeled as a Markov decision process (MDP). The goal in an RL problem is to find a policy (or feedback controller) that is optimal in a certain sense.[15]

Over the last three decades, several methods such as dynamic programming (DP), Monte Carlo (MC) learning, and temporal difference (TD) learning, have been proposed to solve the RL problem.[11] Most of these compute the optimal policy using policy iteration. This is an iterative approach in which every step involves both policy estimation and policy improvement. The policy estimation step aims at making the value function consistent with the current policy; and the policy improvement step makes the policy greedy with respect to the current estimate of the value function. Alternating between the two steps produces a sequence of value function estimates and sub-optimal policies which, in the limit, converge to the optimal value function and the optimal policy, respectively.[11]

The choice of a solution strategy for an RL problem is primarily driven by the assumptions on the environment and the agent. For example, under the perfect model assumption for the environment, classical DP methods for Markov decision problems with finite state and action spaces, provide a closed-form solution to the optimal value function, and are known to converge to the optimal policy in polynomial time.[13,16] Despite their strong convergence properties, however, classical DP methods have limited practical applications because of their stringent requirement for a perfect model of the environment, which is seldom available in practical problems.

MC algorithms belong to a class of approximate RL methods that can be used to estimate the value function using experiences (i.e., sample sequences of states, actions, and rewards accumulated through the agent's interaction with an environment). MC algorithms offer several advantages over DP. First, MC methods allow an agent to learn the optimal behavior directly by interacting with the environment, thereby eliminating the need for an exact model. Second, MC methods can be focused to estimate value functions for a small subset of the states of interest rather than evaluating them for the entire state space as with DP methods. This significantly reduces the computational burden, since value function estimates for less relevant states need not be updated. Third, MC methods may be less sensitive to the violations of the Markov property of the system. Despite the advantages mentioned above, MC methods are difficult to implement in real time, as the value function estimates can only be updated at the end of an experiment. Furthermore, MC methods are also known to exhibit slow convergence as they do not bootstrap, that is, they do not update their value function from other value estimates.[11] Finally, the efficacy of MC methods in RL remains unsettled and is a subject of ongoing research.[11]

TD learning is another class of approximate methods for solving RL problems that combine ideas from DP and MC. Like MC algorithms, TD methods can learn directly from raw experiences without requiring a model; and like DP, TD methods update the value function in real time without having to wait until the end of the experiment. For a detailed exposition on RL solutions, the reader is referred to Sutton and Barto[11] and the references therein.

RL has achieved remarkable success in robotics,[17,18] computer games,[19,20] online advertising,[21] and board games[22,23]; however, its adaptation to process control has been limited (see Reference 24 for recent survey of RL methods in process control)—even though many optimal scheduling and control problems can be formulated as MDPs.[25] This is primarily due to lack of efficient RL algorithms to deal with infinite MDPs (i.e., MDPs with continuous state and action spaces) that define most modern control systems. Although existing RL methods apply to infinite MDPs, exact solutions are possible only in special cases, such as in linear quadratic (Gaussian) control problem, where DP provides a closed-form solution.[26,27] It is plausible to discretize infinite MDPs and use DP to estimate the value function; however, this leads to an exponential growth in the computational complexity with respect to the states and actions, which is referred to as curse of dimensionality.[13] The computational and storage requirements for discretization methods applied to most problems of practical interest in process control remain unwieldy even with today's computing hardware.[28]

The first successful implementation of RL in process control appeared in the series of papers published in the early 2000s,[25,28-32] where the authors proposed approximate dynamic programming (ADP) for optimal control of discrete-time nonlinear systems. The idea of ADP is rooted in the formalism of DP but uses simulations and function approximators (FAs) to alleviate the curse of dimensionality. Other RL methods based on heuristic dynamic programming (HDP),[33] direct HDP,[34] dual heuristic programming,[35] and globalized DHP [36] have also been proposed for optimal control of discrete-time nonlinear systems. RL methods have also been proposed for optimal control of continuous-time nonlinear systems.[37-39] However, unlike discrete-time systems, controlling continuous-time systems with RL has proven to be considerably more difficult and fewer results are available.[26] While the contributions mentioned above establish the feasibility and adaptability of RL in controlling discrete-time and continuous-time nonlinear processes, most of these methods assume complete or partial access to process models.[25,28-31,38,39] This limits existing RL methods to processes for which high-accuracy models are either available or can be derived through system identification.

Recently, several data-based approaches have been proposed to address the limitations of model-based RL in control. In References 32 and 40, a data-based learning algorithm was proposed to derive an improved control policy for discrete-time nonlinear systems using ADP with an identified process model, as opposed to an exact model.

Similarly, Lee and Lee[32] proposed a Q-learning algorithm to learn an improved control policy in a model-free manner using only input–output data. Although these methods remove the requirement for having an exact model (as in RL), they still present several issues. For example, the learning method proposed in References 32 and 40 is still based on ADP, so its performance relies on the accuracy of the identified model. For complex, nonlinear, stochastic systems, identifying a reliable process model may be nontrivial as it often requires running multiple carefully designed experiments. Similarly, the policy derived from Q-learning by Lee and Lee[32] may converge to a suboptimal policy as it avoids adequate exploration of the state and action spaces. Further, calculating the policy with Q-learning for infinite MDPs requires solving a nonconvex optimization problem over the continuous action space at each sampling time, which may render it unsuitable for online deployment for processes with small time constants or modest computational resources. Note that data-based RL methods have also been proposed for continuous-time nonlinear systems.[41,42] Most of these data-based methods approximate the solution to the Hamilton–Jacobi–Bellman (HJB) equation derived for a class of continuous-time affine nonlinear systems using policy iteration. For more information on RL-based optimal control of continuous-time nonlinear systems, the reader is referred to Luo et al,[41] Wang et al,[42]Tang and Daoutidis,[43] and the references therein.

The promise of RL to deliver a real-time, self-learning controller in a model-free and adaptive environment has long motivated the process systems' community to explore novel approaches to apply RL in process control applications. While the plethora of studies from over last two decades has provided significant insights to help connect the RL paradigm with process control, they also highlight the nontrivial nature of this connection and the limitations of existing RL methods in control applications. Motivated by recent developments in the area of DRL, we revisit the problem of RL-based process control and explore the feasibility of using DRL to bring these goals one step closer.

1.2 | Deep reinforcement learning

The recent resurgence of interest in RL results from the successful combination of RL with deep learning that allows for effective generalization of RL to MDPs with continuous state spaces. The deep-Q-network (DQN) proposed recently by Mnih et al[19] combines deep learning for sensory processing [44] with RL to achieve human-level performance on many `Atari` video games. Using unprocessed pixels as inputs, simply through interactions, the DQN can learn in real time the optimal strategy to play `Atari`. This was made possible using a deep neural network FA to estimate the action–value function over the continuous state space, which was then maximized over the action space to find the optimal policy. Before DQN, learning the action–value function using FAs was widely considered difficult and unstable. Two innovations account for this latest gain in stability and robustness: (a) the network is trained off-policy with samples from a replay buffer to minimize correlations between samples, and (b) the network is trained with a target Q-network to give consistent targets during TD backups.

In this article, we propose an off-policy actor–critic algorithm, referred to as a DRL controller, for controlling of discrete-time nonlinear processes. The proposed DRL controller is a model-free controller designed based on TD learning. As a data-based controller, the DRL controller uses two independent deep neural networks to generalize the actor and critic to continuous state and action spaces. The DRL controller is based on the deterministic policy gradient (DPG) algorithm proposed by Silver et al[45] that combines an actor–critic method with insights from DQN. The DRL controller uses ideas similar to those in Lillicrap et al[18] modified to make learning suitable for process control applications. Several simulation examples of different complexities are presented to demonstrate the efficacy of the DRL controller in set-point tracking problems.

The rest of the article is organized as follows: in Section 2, we introduce the basics of MDP and derive a control policy based on value functions. Motivated by the intractability of optimal policy calculations for infinite MDPs, we introduce Q-learning in Section 3 for solving RL problem over continuous state space. A policy gradient algorithm is discussed in Section 4 to extend the RL solution to continuous action space. Combining the developments in Sections 3 and 4, a novel actor–critic algorithm is discussed in Section 5. In Section 6, a DRL framework is proposed for data-based control of discrete-time nonlinear processes. The efficacy of a DRL controller is demonstrated in several examples in Section 7. Finally, Section 8 compares a DRL controller to an MPC.

This article follows a tutorial-style presentation to assist readers unfamiliar with the theory of RL. The material is systematically introduced to highlight the challenges with existing RL solutions in process control applications and to motivate the development of the proposed DRL controller. The background material presented here is only introductory, and readers are encouraged to refer to the cited references for a detailed exposition on these topics.

2 | THE RL PROBLEM

The RL framework consists of a learning agent (e.g., a controller) interacting with a stochastic environment (e.g., a plant or process), denoted by $\mathscr{E}$, in discrete time steps. The objective in an RL problem is to identify a policy (or control actions) to maximize the expected cumulative reward the agent receives in the long run.[11,15] The RL problem is a sequential decision-making problem, in which the agent incrementally learns how to optimally interact with the environment by maximizing the expected reward it receives. Intuitively, the agent–environment interaction is to be understood as follows. Given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, the agent at time step $t \in \mathbb{N}$ observes some representation of the environment's state $s_t \in \mathcal{S}$ and on that basis selects an action $a_t \in \mathcal{A}$. One time step later, in part as a consequence of its action, the agent finds itself in a new state $s_{t+1} \in \mathcal{S}$ and receives a scalar reward $r_t \in \mathbb{R}$ from the environment indicating how well the agent performed at $t \in \mathbb{N}$. This procedure repeats for all $t \in \mathbb{N}$, as in the case of a continuing task or until the end of an episode. The agent–environment interactions are illustrated in Figure 1.

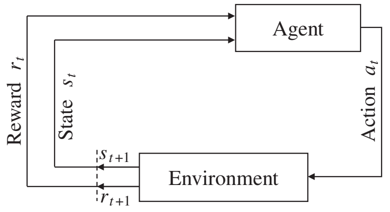

FIGURE 1 A schematic of the agent-environment interactions in a standard RL problem

2.1 | Markov decision process

An MDP consists of the following: a state space $\mathcal{S}$; an action space $\mathcal{A}$; an initial state distribution $p(s_1)$; and a transition distribution $p(s_{t+1}|s_t, a_t)$[1] satisfying the following Markov property:

$$p(s_{t+1}|s_t,a_t,\ldots,s_1,a_1) = p(s_{t+1}|s_t,a_t), \tag{1}$$

for any trajectory s_1, a_1, …, s_T, a_T generated in the state–action space $\mathcal{S}\times\mathcal{A}$ (for continuing tasks, $T\to\infty$, while for episodic tasks, $T\in\mathbb{N}$ is the terminal time); and a reward function $r_t:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$. In an MDP, the transition function (1) is the likelihood of observing a state $s_{t+1}\in\mathcal{S}$ after the agent takes an action $a_t\in\mathcal{A}$ in state $s_t\in\mathcal{S}$. Thus, $\int_{\mathcal{S}}p(s|s_t,a_t)ds = 1$ for all $(s_t,a_t)\in\mathcal{S}\times\mathcal{A}$.

A policy is used to describe the actions of an agent in the MDP. A stochastic policy is denoted by $\pi:\mathcal{S}\to\mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on $\mathcal{A}$. Thus, $\pi(a_t|s_t)$ is the probability of taking action a_t in state s_t; we have $\int_{\mathcal{A}}\pi(a|s_t)da = 1$ for each $s_t\in\mathcal{S}$. This general formulation allows any deterministic policy $\mu:\mathcal{S}\to\mathcal{A}$, through the definition

$$\pi(a|s_t) = \begin{cases} 1 & \text{if } a = \mu(s_t), \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The agent uses π to sequentially interact with the environment to generate a sequence of states, actions, and rewards in $\mathcal{S}\times\mathcal{A}\times\mathcal{R}$, denoted generically as $h = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$, where $(H = h)\sim p^\pi(h)$ is an arbitrary history generated under policy π and distributed according to the probability density function (PDF) $p^\pi(\cdot)$. Note that under the Markov assumption of the MDP, the PDF for the history can be decomposed as follows:

$$p^\pi(h) = p(s_1)\prod_{t=1}^{T}p(s_{t+1}|s_t,a_t)\pi(a_t|s_t).$$

The total reward accumulated by the agent in a task from time $t\in\mathbb{N}$ onward is

$$R_t(h) = \sum_{k=t}^{\infty}\gamma^{k-t}r_t(s_k,a_k), \tag{3}$$

Here $\gamma\in[0,1]$ is a (user-specified) discount factor: a unit reward has the value 1 right now, γ after one time step, and γ^τ after τ sampling intervals. If we specify $\gamma = 0$, only the immediate reward has any influence; if we let $\gamma\to 1$, future rewards are considered more strongly—informally, the controller becomes "farsighted." Although Equation (3) is defined for continuing tasks, that is, $T\to\infty$, the notation can be applied also for episodic tasks by introducing a special absorbing terminal state that transitions only to itself and generates rewards of 0. These conventions are used to simplify the notation and to express close parallels between episodic and continuing tasks (see Reference 11).

2.2 | Value function and optimal policy

The state–value function, $V^\pi:\mathcal{S}\to\mathbb{R}$, assigns a value to each state, according to the given policy π. In state s_t, $V^\pi(s_t)$ is the value of the future reward an agent is expected to receive by starting at $s_t\in\mathcal{S}$ and the following policy π thereafter. In detail,

$$V^\pi(s_t) = \mathbb{E}_{h\sim p^\pi(\cdot)}[R_t(h)|s_t]. \tag{4}$$

The closely-related action-value function $Q^\pi:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$ decouples the immediate action from the policy π, assuming only that π is used for all subsequent steps:

$$Q^\pi(s_t,a_t) = \mathbb{E}_{h\sim p^\pi(\cdot)}[R_t(h)|s_t,a_t]. \tag{5}$$

The Markov property of the underlying dynamic process gives these two value functions a recurrent structure illustrated as

$$Q^\pi(s_t,a_t) = \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[r(s_t,a_t)+\gamma\pi(a_{t+1}|s_{t+1})Q^\pi(s_{t+1},a_{t+1})]. \tag{6}$$

Solving an RL problem amounts to finding a policy $\pi^\star$ that outperforms all other policies across all possible scenarios. Identifying π^* will yield an optimal Q-function, $Q^\star$, such that

$$Q^\star(s_t,a_t) = \max_\pi Q^\pi(s_t,a_t), \quad (s_t,a_t)\in\mathcal{S}\times\mathcal{A}. \tag{7}$$

Conversely, knowing the function $Q^\star$ is enough to recover an optimal policy by making a "greedy" choice of action:

$$\pi^\star(a|s_t) = \begin{cases} 1 & \text{if } a = \arg\max_{a\in\mathcal{A}}Q^\star(s_t a) \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Note that this policy is actually deterministic. While solving the Bellman equation in Equation (6) for the Q-function provides an approach to finding an optimal policy in Equation (8), and thus solving the RL problem, this solution is rarely useful in practice. This is because the solution relies on two key assumptions—(a) the dynamics of the environment is accurately known, that is, $p(s_{t+1}|s_t, a_t)$ is exactly known for all $(s_t,a_t)\in\mathcal{S}\times\mathcal{A}$; and (b) sufficient computational resources are available to calculate $Q^\pi(s_t, a_t)$ for all $(s_t,a_t)\in\mathcal{S}\times\mathcal{A}$. These assumptions are a major impediment for solving process control problems, wherein complex process behavior might not be accurately known or might change over time, and the state and action spaces may be continuous. For an RL solution to be practical, one typically needs to settle for approximate solutions. In the next section, we introduce Q-learning that approximates the optimal Q-function (and thus the optimal policy) using the agent's experiences (or samples) as opposed to process knowledge. Such class of approximate solutions to the RL problem is called the model-free RL methods.

3 | Q-LEARNING

Q-learning is one of the most important breakthroughs in RL.[46,47] The idea is to learn $Q^\star$ directly, instead of first learning Q^π and then computing $Q^\star$ in Equation (7). Q-learning constructs $Q^\star$ through successive approximations. Similar to Equation (6), using the Bellman equation, $Q^\star$ satisfies the identity

$$Q^\star(s_t,a_t)=\mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[r(s_t,a_t)+\gamma \max_{a\prime\in\mathcal{A}}Q^\star(s_{t+1},a\prime)]. \tag{9}$$

There are several ways to use Equation (9) to compute $Q^\star$. The standard Q-iteration (QI) is a model-based method that requires complete knowledge of the states, the transition function, and the reward function to evaluate the expectation in Equation (9). Alternatively, TD learning is a model-free method that uses sampling experiences (s_t, a_t, r_t, s_{t+1}) to approximate $Q^\star$.[11] This is done as follows. First, the agent explores the environment by following some stochastic behavior policy, $\beta:\mathcal{S}\rightarrow\mathcal{P}(\mathcal{A})$, and receiving an experience tuple (s_t, a_t, r_t, s_{t+1}) at each time step. The generated tuple is then used to improve the current approximation of $Q^\star$, denoted $\hat{Q}_i$, as follows:

$$\hat{Q}_{i+1}(s_t,a_t)\leftarrow\hat{Q}_i(s_t,a_t)+\alpha\delta, \tag{10}$$

where $\alpha\in(0,1]$ is the learning rate, and δ is the TD error, defined by

$$\delta=r(s_t,a_t)+\gamma\max_{a\prime\in\mathcal{A}}\hat{Q}_i(s_{t+1},a\prime)-\hat{Q}_i(s_t,a_t). \tag{11}$$

The conditional expectation in Equation (9) falls away in TD learning in Equation (10), since s_{t+1} in (s_t, a_t, r_t, s_{t+1}) is distributed according to the target density, $p(\cdot|s_t, a_t)$. Making the greedy policy choice using $\hat{Q}_{i+1}$ instead of $Q^\star$ (recall Equation (8)) produces

$$\hat{\pi}_{i+1}(s_t)=\arg\max_{a\in\mathcal{A}}\hat{Q}_{i+1}(s_t,a), \tag{12}$$

where $\hat{\pi}_{i+1}$ is a greedy policy based on $\hat{Q}_{i+1}$. In the RL literature, Equation (10) is referred to as policy evaluation and Equation (12) as policy improvement. Together, steps (10) and (12) are called Q-learning.[47] Pseudocode for implementing this approach is given in Algorithm 1. Algorithm 1 is a model-free, online and off-policy algorithm. It is model-free as it does not require an explicit model of the environment, and it is online because it only utilizes the latest experience tuple to implement the policy evaluation and improvement steps. Further, 1 is off-policy because the agent acts in the environment according to its behavior policy β, but still learns its own policy, π. Observe that the behavior policy in Algorithm 1 is ϵ-greedy, in that it generates greedy actions for the most part but has a non-zero probability, ϵ, of generating a random action. Note that off-policy is a critical component in RL as it ensures a combination of exploration and exploitation. Finally, for Algorithm 1, it can be shown that $\hat{Q}_i\rightarrow Q^\star$ with probability 1 as $i\rightarrow\infty$.[47]

Algorithm 1 Q-learning

```
Output: Action-value function $Q(s,a)$
Initialize: Arbitrarily set $Q$, e.g., to 0 for all states, set $Q$ for terminal states as 0
for each episode do
  Initialize state $s$
  for each step of episode, state $s$ is not terminal do
    $a \leftarrow$ action for $s$ derived by $Q$, e.g., $\epsilon$-greedy
    take action $a$, observe $r$ and $s'$
    $\delta \leftarrow r + \gamma\max_{a'}Q(s',a') - Q(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\delta$
    $s \leftarrow s'$
  end for
end for
```

3.1 | Q-learning with function approximation

Although Algorithm 1 enjoys strong theoretical convergence properties, it requires storing the $\hat{Q}$-values for all state–action pairs $(s,a)\in\mathcal{S}\times\mathcal{A}$. For applications like control, where both $\mathcal{S}$ and $\mathcal{A}$ are infinite sets, some further simplification is required. Several authors have proposed space discretization methods.[48] While such discretization methods may work for simple problems, in general, they are not efficient in capturing the complex dynamics of industrial processes.

The problem of generalizing Q-learning from finite to continuous spaces has been studied extensively over the last two decades. The basic idea in continuous spaces is to use an FA. An FA $Q[s, a, w]$ is a parametric function $Q(s, a, w)$, whose parameters w are chosen to make $Q(s, a, w)\approx Q(s, a)$ for all $(s\times a)\in\mathcal{S}\times\mathcal{A}$. This is achieved by minimizing the following quadratic loss function $\mathscr{L}$:

$$\mathscr{L}_t(w)=\mathbb{E}_{s_t\sim\rho^\beta(\cdot),a_t\sim\beta(\cdot|s_t)}\left[(\tilde{y}_t-Q(s_t,a_t,w))^2\right], \tag{13}$$

where $\tilde{y}_t$ is the target and ρ^β is a discounted state visitation distribution under behavior policy β. The role of ρ^β is to weight $\mathscr{L}$ based on how frequently a particular state is expected to be visited. Further, as in any supervised learning problem, the target $\tilde{y}_t$ is given by $Q^\star(s_t, a_t)$; however, since $Q^\star$ is unknown, it can be replaced with its approximation. A popular choice of an approximate target is a bootstrap target (or a TD target), given as

$$\tilde{y}_t=\mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}[r(s_t,a_t)+\gamma\max_{a\prime\in\mathcal{A}}Q(s_{t+1},a\prime,w)]. \tag{14}$$

In contrast to supervised learning, where the target is typically independent of model parameters, the target in Equation (14) depends on the FA parameters. Finally, Equation (13) can be minimized using a stochastic gradient descent (SGD) algorithm. An SGD is an iterative optimization method that adjusts w in the direction that would mostly reduce $\mathscr{L}(w)$ for it. The update step for SGD is given as follows:

$$w_{t+1} \leftarrow w_t - \frac{1}{2}\alpha_{c,t}\nabla\mathscr{L}_t(w_t), \tag{15}$$

where w_t and w_{t+1} are the old and new parameter values, respectively, and $\alpha_{c,t}$ is a positive step-size parameter. Given Equation (13), this gradient can be calculated as follows:

$$\nabla\mathscr{L}_t(w_t) = -2\mathbb{E}_{s_t\sim\rho^{\beta}(\cdot),a_t\sim\beta(\cdot|s_t)}[\tilde{y}_t - Q(s_t,a_t,w_t)]\nabla_w Q(s_t,a_t,w_t). \tag{16}$$

To derive Equation (16), it is assumed that $\tilde{y}_t$ is independent of w. Note that this is a common assumption in Q-learning with TD targets.[11] Finally, after updating w_{t+1} (and computing $Q[s_{t+1}, a', w_{t+1}]$), the optimal policy can be computed as follows:

$$\pi(s_{t+1}) = \underset{a'\in\mathcal{A}}{\arg\max}\, Q(s_{t+1},a',w_{t+1}). \tag{17}$$

The pseudocode for Q-learning with FA is given in Algorithm 2.

Algorithm 2 Q-learning with FA

```
Output: Action value function $Q(s,a,w)$
Initialize: Arbitrarily set action-value function weights $w$ (e.g., $w = 0$)
for each episode do
  Initialize state $s$
  for each step of episode, state $s$ is not terminal do
    $a \leftarrow$ action for $s$ derived by $Q$, e.g., $\epsilon$-greedy
    take action $a$, observe $r$ and $s'$
    $\tilde{y} \leftarrow r + \gamma max_{a'} Q(s',a',w)$
    $w \leftarrow w + \alpha_c(y - Q(s,a,w))\nabla_w Q(s,a,w)$
    $s \leftarrow s'$
  end for
end for
```

The effectiveness of Algorithm 2 depends on the choice of FA. Over the past decade, various FAs, both parametric and nonparametric, have been proposed, including linear basis, Gaussian processes, radial basis, and Fourier basis. For most of these choices, Q-learning with TD targets may be biased.[49] Further, unlike Algorithm 1, the asymptotic convergence of $Q(s, a, w) \rightarrow Q(s, a)$ with Algorithm 2 is not guaranteed. This is primarily due to Algorithm 2 using an off-policy (i.e., behavior distribution), bootstrapping (i.e., TD target), and FA approach—a combination known in the RL community as the "deadly triad".[11] The possibility of divergence in the presence of the deadly triad is well known. Several examples have been published: see Tsitsiklis and Van Roy,[50] Baird,[51] and Fairbank and Alonso.[52] The root cause for the instability remains unclear—taken one by one, the factors listed above are not problematic. There are still many open problems in off-policy learning. Despite the lack of theoretical convergence guarantees, all three elements of the deadly triad are also necessary for learning to be effective in practical applications. For example, FA is required for scalability and generalization, bootstrapping for computational and data efficiency, and off-policy learning for decoupling the behavior policy from the target policy. Despite the limitations of Algorithm 2, recently, Mnih et al[19,20] have successfully adapted Q-learning with FAs to learn to play **`Atari`** games from pixels (for details, see Kober et al[53]).

4 | POLICY GRADIENT

Although Algorithm 2 generalizes Q-learning to continuous spaces, the method lacks convergence guarantees except for with linear FAs, where it has been shown not to diverge. Moreover, target calculations in Equation (14) and greedy action calculations in Equation (17) require maximization of the Q-function over the action space. Such optimization steps are computationally impractical for large and unconstrained FAs and for continuous action spaces. Control applications typically have both these complicating characteristics, making Algorithm 2 nontrivial to implement.

Instead of approximating $Q^\star$ and then computing π (see Algorithms 1 and 2), an alternative approach is to directly compute the optimal policy, $\pi^\star$ without consulting the optimal Q-function. The Q-function may still be used to learn the policy, but is not required for action selection. Policy gradient methods are RL algorithms that work directly in the policy space. Two separate formulations are available: the average reward formulation and the start state formulation. In the start state formulation, the goal of an agent is to obtain a policy π_θ, parameterized by $\theta \in \mathbb{R}^{n_\theta}$, that maximizes the value of starting at state $s_0 \in \mathcal{S}$ and the following policy π_θ. For a given policy, π_θ, the performance of the agent can be evaluated as follows:

$$J(\pi_\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{h\sim p^{\pi}(\cdot)}[R_1(h)|s_0], \tag{18}$$

Observe that the agent's performance in Equation (18) is completely described by the policy parameters in θ. A policy gradient algorithm maximizes Equation (18) by computing an optimal value of θ. A stochastic gradient ascent (SGA) algorithm incrementally adjusts θ in the direction of $\nabla_\theta J(\pi_\theta)$, such that

$$\theta_{t+1} \leftarrow \theta_t + \alpha_{a,t}\nabla_\theta J(\pi_\theta)\big|_{\theta=\theta_t}, \tag{19}$$

where $\alpha_{a,t}$ is the learning rate. Note that calculating $\nabla_\theta J(\pi_\theta)$ requires access to the distribution of states under the current policy, which, as noted earlier, is unknown for most processes. The implementation of policy gradient is made effective by the policy gradient theorem that calculates a closed-form solution for $\nabla_\theta J(\pi_\theta)$ without reference to the state distribution.[54] The policy gradient theorem establishes that

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s_t\sim\rho_\gamma^{\pi}(\cdot),a_t\sim\pi_\theta(\cdot|s_t)}[Q^{\pi}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)], \tag{20}$$

where $\rho_\gamma^{\pi}(s) := \sum_{t=0}^{\infty}\gamma^t p(s_t = s|s_0,\pi_\theta)$ is the discounted state visitation distribution and $p(s_t|s_0,\pi)$ is a t-step ahead state transition density from s_0 to s_t. Equation (20) gives a closed-form solution for the gradient in Equation (19) in terms of the Q-function and the gradient of the policy being evaluated. Further, Equation (20) assumes a stochastic policy (observe the expectation over the action space). Now, since the expectation is over the policy being evaluated, that is, π_θ, Equation (20) is an on-policy gradient. Note that an off-policy gradient theorem can also be derived for a class of stochastic policies (see Degris et al[55] for details).

To implement Equation (19) with the policy gradient theorem in Equation (20), we replace the expectation in Equation (19) with its sample-based estimate and replace Q^π with the actual returns, R_t. This leads to a policy gradient algorithm, called REINFORCE.[56] Pseudocode for REINFORCE is given in Algorithm 3. In contrast to Algorithms 1 and 2, Algorithm 3 avoids solving complex optimization problems over continuous action spaces and generalizes effectively in continuous spaces. Further, unlike Q-learning that always learns a deterministic greedy policy, Algorithm 3 supports both deterministic and stochastic policies. Finally, Algorithm 3 exhibits good convergence properties, with the estimate in Equation (19) guaranteed to converge to a local optimum if the estimation of $\nabla_\theta J(\pi_\theta)$ is unbiased (see Sutton et al[54] for a detailed proof).

Algorithm 3 Policy Gradient—REINFORCE

```
Output: Optimal policy $\pi(a|s,\theta)$
Initialize: Arbitrarily set policy parameters $\theta$
for true do
    generate an episode $s_0,a_0,r_1,\ldots,s_{T-1},a_{T-1},r_T$, following $\pi(\cdot|\cdot,\theta)$
    for each step $t$ of episode $0,1,\ldots T - 1$ do
        $R_t \leftarrow$ return from step $t$
        $\theta \leftarrow \theta + \alpha_{a,\,t}\gamma^t R_t \nabla_\theta \pi(a_t|\, s_t,\, \theta)$
    end for
end for
```

Despite the advantages of Algorithm 3 over traditional Q-learning, Algorithm 3 is not amenable to online implementation as it requires access to R_t—the total reward the agent is expected to receive at the end of an episode. Furthermore, replacing the Q-function by R_t leads to a large variance in the estimation of $\nabla_\theta J(\pi_\theta)$,[54,57] which in turn leads to slower convergence.[58] An approach to address the issues mentioned above is to use a low-variance, bootstrapped estimate of the Q-function, as in the actor–critic architecture.

5 | ACTOR–CRITIC ARCHITECTURE

The actor–critic is a widely used architecture that combines the advantages of policy gradient with Q-learning.[54,55,59] Like policy gradient, actor–critic methods generalize to continuous spaces, while the issue of large variance is countered by bootstrapping, such as Q-learning with TD update. A schematic of the actor–critic architecture is shown in Figure 2. The actor–critic architecture consists of two eponymous components: an actor that finds an optimal policy, and a critic that evaluates the current policy prescribed by the actor. The actor implements the policy gradient method by adjusting policy parameters using SGA, as shown in Equation (19). The critic approximates the Q-function in Equation (20) using an FA. With a critic, Equation (20) can be approximately written as follows:

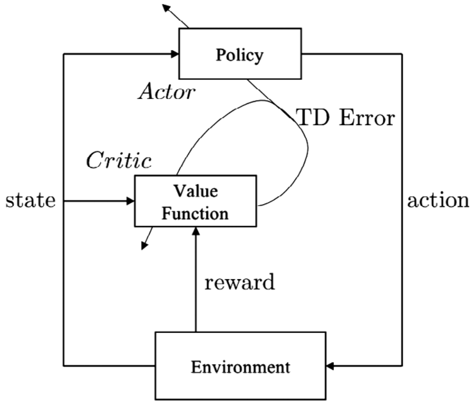

FIGURE 2 A schematic of the actor–critic architecture

$$\hat{\nabla}_\theta J(\pi_\theta) = \mathbb{E}_{s_t \sim \rho_\gamma^\pi(\cdot),\, a_t \sim \pi_\theta(\cdot|s_t)}\left[Q^\pi(s_t,a_t,w)\nabla_\theta \log \pi_\theta(a_t|s_t)\right], \tag{21}$$

where $w \in \mathbb{R}^{n_w}$ is recursively estimated by the critic using SGD in Equation (15) and $\theta \in \mathbb{R}^{n_\theta}$ is recursively estimated by the actor using SGA by substituting Equation (21) into Equation (19). Observe that while the actor–critic method combines the policy gradient with Q-learning, the policy is not directly inferred from Q-learning, as in Equation (17). Instead, the policy is updated in the policy gradient direction in Equation (19). This avoids the costly optimization in Q-learning and also ensures that changes in the Q-function only result in small changes in the policy, leading to less or no oscillatory behavior in the policy. Finally, under certain conditions, implementing Equation (19) with Equation (21) guarantees that θ converges to the local optimal policy.[54,55]

5.1 | Deterministic actor–critic method

For a stochastic policy π_θ, calculating the gradient in Equation (20) requires integration over the space of states and actions. As a result, computing the policy gradient for a stochastic policy may require many samples, especially in high-dimensional action spaces. To allow for efficient calculation of the policy gradient in Equation (20), Silver et al[45] propose a DPG framework. This assumes that the agent follows a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$. For a deterministic policy, $a_t = \mu_\theta(s_t)$ with probability 1, where $\theta \in \mathbb{R}^{n_\theta}$ is the policy parameter. The corresponding performance in Equation (18) can be written as follows:

$$J(\mu_\theta) = V^{\mu_\theta}(s_0) = \mathbb{E}_{h \sim p^\mu(\cdot)}\left[\sum_{t=1}^{\infty} \gamma^{k-1} r(s_t, \mu_\theta(s_t)) | s_0\right]. \tag{22}$$

Similar to the policy gradient theorem in Equation (20), Silver et al[45] proposed a deterministic gradient theorem to calculate the gradient of Equation (22) with respect to the policy parameter θ:

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s_t \sim \rho_\gamma^\mu(\cdot)}\left[\nabla_a Q^\mu(s_t, a)\Big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)\right], \tag{23}$$

where ρ_γ^μ is the discounted state distribution under policy μ_θ (similar to ρ_γ^π in Equation (20)). Note that unlike Equation (20), the gradient in Equation (23) only involves expectation over the states generated according to μ_θ. This makes the DPG framework computationally more efficient to implement compared to the stochastic policy gradient.

To ensure that the DPG continues to explore the state and action spaces satisfactorily, it is possible to implement DPG off-policy. For a stochastic behavior policy β and a deterministic policy μ_θ, Silver et al[45] showed that a DPG exists and can be analytically calculated as

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s_t \sim \rho_\gamma^\beta(\cdot)}\left[\nabla_a Q^\mu(s_t, a)\Big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)\right], \tag{24}$$

where ρ_γ^β is the discounted state distribution under behavior policy β. Compared to Equation (23), the off-policy DPG in Equation (24) involves expectation with respect to the states generated by a behavior policy β. Finally, the off-policy DPG can be implemented using the actor–critic architecture. The policy parameters, θ, can be recursively updated by the actor using Equation (19), where $\nabla_\theta J(\mu_\theta)$ is given in Equation (24); and the Q-function, $Q^\mu(s_t, a)$, in Equation (24) is replaced with a critic, $Q^\mu(s_t, a, w)$, whose parameter vector w is recursively estimated using Equation (15). Pseudocode for the off-policy deterministic actor–critic is given in Algorithm 4.

Algorithm 4 Deterministic Off-policy Actor–Critic Method

```
Output: Optimal policy $\mu_\theta(s)$
Initialize: Arbitrarily set policy parameters $\theta$ and $Q$-function weights $w$
for true do
  initialize $s$, the first state of the episode
  for $s$ is not terminal do
    $a \sim \beta(\cdot \mid s)$
    take action $a$, observe $s'$ and $r$
    $\tilde{y} \leftarrow r + \gamma Q^\mu(s', \mu_\theta(s'), w)$
    $w \leftarrow w + \alpha_w(\tilde{y} - Q^\mu(s, \mu_\theta(s), w))\nabla_w Q^\mu(s, a, w)$
    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a, w)|_{a=\mu_\theta(s)}$
    $s \leftarrow s'$
  end for
end for
```

6 | DRL CONTROLLER

In this section, we connect the terminologies and methods for RL discussed in Section 5, and propose a new controller, referred to as a DRL controller. The proposed DRL controller is a model-free controller based on DPG. The DRL controller is implemented using the actor–critic architecture and uses two independent deep neural networks to generalize the actor and critic to continuous state and action spaces.

6.1 | States, actions, and rewards

We consider discrete dynamical systems with input and output sequences u_t and y_t, respectively. For simplicity, we focus on the case where the outputs y_t consists of full information on the state of the system to be controlled. Removing this hypothesis is an important practical element of our ongoing research in this area.

The correspondence between the agent's action in the RL formulation and the plant input from the control perspective is direct: we identify $a_t = u_t$. The relationship between the RL state and the state of the plant is subtler. The RL state, $s_t \in \mathcal{S}$, must capture all the features of the environment on the basis of which the RL agent acts. To ensure that the agent has access to relevant process information, we define the RL state as a tuple of the current and past outputs, past actions, and current deviation from the set-point y_{sp}, such that

$$s_t := \left\langle y_t, \ldots, y_{t-d_y}, a_{t-1}, \ldots, a_{t-d_a}, \left(y_t - y_{\text{sp}}\right) \right\rangle, \tag{25}$$

where $d_y \in \mathbb{N}$ and $d_a \in \mathbb{N}$ denote the number of past output and input values, respectively. In this article, it is assumed that d_y and d_a are known a priori. Note that for $d_a = 0$, $d_y = 0$, the RL state is "memoryless", in that

$$s_t := \left\langle y_t, \left(y_t - y_{\text{sp}}\right) \right\rangle. \tag{26}$$

Further, we explore only deterministic policies expressed using a single-valued function $\mu : \mathcal{S} \to \mathcal{A}$, so that for each state $s_t \in \mathcal{S}$, the probability measure on $\mathcal{A}$ defining the next action puts weight 1 on the single point $\mu(s_t)$.

The goal for the agent in a set-point tracking problem is to find an optimal policy, μ, that reduces the tracking error. This objective is incorporated in the RL agent by means of a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, whose aggregate value the agent tries to maximize. In contrast with MPC, where the controller minimizes the tracking error over the space of control actions, here the agent maximizes the reward it receives over the space of policies. We consider two reward functions.

The first—an ℓ_1-reward function—measures the negative ℓ_1-norm of the tracking error. Mathematically, for a multi-input and multi-output (MIMO) system with n_y outputs, the ℓ_1-reward is

$$r(s_t, a_t, s_{t+1}) = -\sum_{i=1}^{n_y} \left| y_{i,t} - y_{i,\text{sp}} \right|, \tag{27}$$

where $y_{i,t} \in \mathcal{Y}$ are the ith output and $y_{i,\text{sp}} \in \mathcal{Y}$ is the set-point for the ith output. Variants of the ℓ_1-reward function are presented and discussed in Section 6.5. The second reward—a polar reward function—assigns a 0 reward if the tracking error is a monotonically decreasing

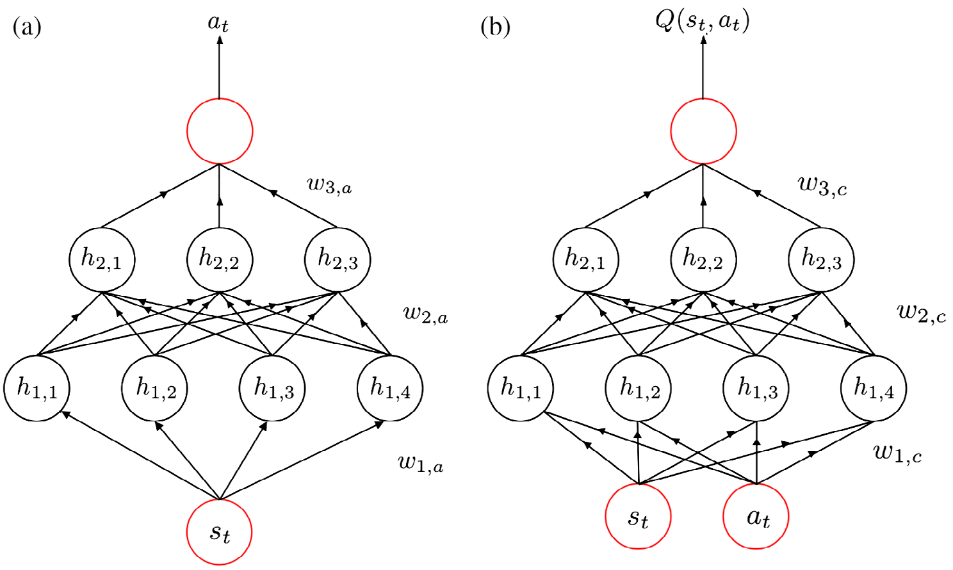

FIGURE 3 A deep neural network representation of (a) the actor and (b) the critic. The red circles represent the input and output layers and the black circles represent the hidden layers of the network

function at each sampling time for all n_y outputs or—1 otherwise. Mathematically, the polar reward function is

$$r(s_t,a_t,s_{t+1}) = \begin{cases} 0 & \text{if } |y_{i,t}-y_{i,sp}| > |y_{i,t+1}-y_{i,sp}| \quad \forall i \in \{1,\ldots,n_y\} \\ -1 & \text{otherwise} \end{cases} \tag{28}$$

Observe that a polar reward (Equation (28)) incentivizes gradual improvements in tracking performance, which leads to less aggressive control strategy and a smoother tracking compared with the ℓ_1-reward in Equation (27).

6.2 | Policy and Q-function approximations

In this section, we discuss the FAs used by the DRL controller to approximate the policy and the Q-function. We use neural networks to generalize the policy and the Q-function to continuous state and action spaces (see Appendix for the basics of neural networks). The policy, μ, is represented using a deep feed-forward neural network, parameterized by weights $W_a \in \mathbb{R}^{n_a}$, such that given $s_t \in \mathcal{S}$ and W_a, the policy network, produces an output $a_t = \mu(s_t, W_a)$. Similarly, the Q-function is also represented using a deep neural network, parameterized by weights $W_c \in \mathbb{R}^{n_c}$, such that given $(s_t,a_t) \in \mathcal{S} \times \mathcal{A}$ and W_c, the Q-network, outputs $Q^{\mu}(s_t, a_t, W_c)$.

6.3 | Learning control policies

As noted, the DRL controller uses a deterministic off-policy actor–critic architecture to learn the control policy in set-point tracking problems. The proposed method is similar to Algorithm 4 or the one proposed by Lillicrap et al,[18] but modified for set-point tracking problems. In the proposed DRL controller architecture, the actor is represented by $\mu(s_t, W_a)$ and a critic is represented by $Q(s_t, a_t, W_c)$. A schematic of the proposed DRL controller architecture is illustrated in Figure 3. The critic network predicts Q-values for each state–action pair, and the actor network proposes an action for a given state. The goal is then to learn the actor and critic neural network parameters by interacting with the process plant. Once the networks are trained, the actor network is used to compute the optimal action for any given state.

For a given actor–critic architecture, the network parameters can be readily estimated using SGD (or SGA); however, these methods are not effective in applications, where the state, action, and reward sequences are temporally correlated. This is because SGD provides a 1-sample network update, assuming that the samples are independently and identically distributed (iid). For dynamic systems, for which a tuple (s_t, a_t, r_t, s_{t+1}) may be temporally correlated to the past tuples (e.g., $[s_{t-1}, a_{t-1}, r_{t-1}, s_t]$), up to the Markov order, such iid network updates in the presence of correlations are not effective.

To make learning more effective for dynamic systems, we propose to break the inherent temporal correlations between the tuples by randomizing it. To this effect, we use a batch SGD, rather than a single sample SGD for network training. As in DQN, we use a replay memory (RM) for batch training of the networks. The RM is a finite memory that stores a large number of tuples, denoted as $\left\{\left(s^{(i)},a^{(i)},r^{(i)},s\prime^{(i)}\right)\right\}_{i=1}^{K}$, where K is the size of the RM. As a queue data structure, the latest tuples are always stored in the RM, and the old tuples are discarded to keep the cache size constant. At each time, the network is updated by uniformly sampling M ($M \le K$) tuples from the RM. The critic update in Algorithm 4 with RM results in a batch stochastic gradient, with the following update step:

$$W_c \leftarrow W_c + \frac{\alpha_c}{M}\sum_{i=1}^{M}\left(\tilde{y}^{(i)} - Q\left(s^{(i)},\mu\left(s^{(i)},W_a\right),W_c\right)\right)\nabla_{W_c}Q^{\mu}\left(s^{(i)},\mu\left(s^{(i)},W_a\right),W_c\right), \tag{29}$$

where

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}\left(s'^{(i)},\mu\left(s'^{(i)},W_a\right),W_c\right), \tag{30}$$

for all i = 1 ,..., M. The batch update, similar to Equation (29), can also be derived for the actor network using an RM.

We propose another strategy to further stabilize the actor–critic architecture. First, observe that the parameters of the critic network, W_c, being updated in Equation (29), are also used in calculating the target, $\tilde{y}$ in Equation (30). Recall that, in supervised learning, the actual target is independent of W_c; however, as discussed in Section 3.1, in the absence of actual targets, we use W_c in Equation (30) to approximate the target values. Now, if the W_c updates in Equation (29) are erratic, then the target estimates in Equation (30) are also erratic, and may cause the network to diverge, as observed in Lillicrap et al.[18] We propose to use a separate network, called target network, to estimate the target in Equation (30). Our solution is similar to the target network used in Lillicrap et al[18] and Mnih et al.[19] Using a target network, parameterized by $W_c' \in \mathbb{R}^{n_c}$, Equation (30) can be written as follows:

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}\left(s'^{(i)}, \mu\left(s'^{(i)}, W_a'\right), W_c'\right), \tag{31}$$

where

$$W_c\prime \leftarrow \tau W_c + (1-\tau) W_c\prime, \tag{32}$$

and $0 < \tau < 1$ is the target network update rate. Observe that the parameters of the target network in Equation (32) are updated by having them slowly track the parameters of the critic network, W_c. In fact, Equation (32) ensures that the target values change slowing, thereby improving the stability of learning. A similar target network can also be used for stabilizing the actor network.

Note that while we consider the action space to be continuous, in practice, it is also often bounded. This is because the controller actions typically involve changing bounded physical quantities, such as flow rates, pressure, and pH. To ensure that the actor network produces feasible controller actions, it is important to bound the network over the feasible action space. Note that if the networks are not constrained, then the critic network will continue providing gradients that encourage the actor network to take actions beyond the feasible space.

In this article, we assume that the action space, $\mathcal{A}$ is an interval action space, such that $\mathcal{A} = [a_L, a_H]$, where $a_L < a_H$ (the inequality holds element-wise for systems with multiple inputs). One approach to enforce the constraints on the network is to bound the output layer of the actor network. This is done by clipping the gradients used by the actor network in the update step (see Step 10 in Algorithm 4). For example, using the clipping gradient method proposed in Reference 60 the gradient used by the actor, $\nabla_a Q^{\mu}(s, a, w)$, can be clipped as follows

$$\nabla_a Q^{\mu}(s,a,w) \leftarrow \nabla_a Q^{\mu}(s,a,w) \times \begin{cases} (a_H - a)/(a_H - a_L), & \text{if } \nabla_a Q^{\mu}(s,a,w) \text{ increases } a \\ (a - a_L)/(a_H - a_L), & \text{otherwise} \end{cases} \tag{33}$$

Note that in Equation (33), the gradients are downscaled as the controller action approaches the boundaries of its range, and are inverted if the controller action exceeds the range. With Equation (33) in place, even if the critic continually recommends increasing the controller action, it will converge to the its upper bound, a_H. Similarly, if the critic decides to decrease the controller action, it will decrease immediately. Using Equation (33) ensures that the controller actions are within the feasible space. Now, putting all the improvement strategies discussed in this section, a schematic of the proposed DRL controller architecture for set-point tracking is shown in Figure 4, and the pseudocode is provided in Algorithm 5.

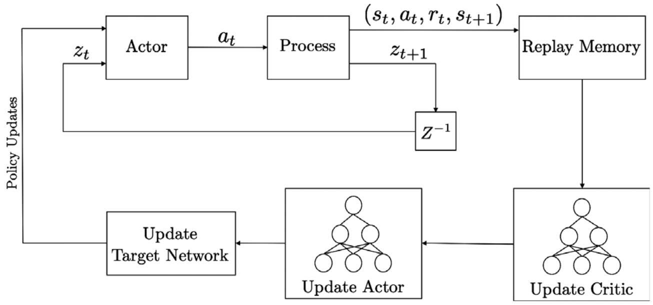

FIGURE 4 A schematic of the proposed DRL controller architecture for set-point tracking problems

Algorithm 5 Deep Reinforcement Learning Controller

```
Output: Optimal policy $\mu(s, W_a)$
Initialize: $W_a, W_c$ to random initial weights
Initialize: $W_a' \leftarrow W_a$ and $W_c' \leftarrow W_c$
Initialize: Replay memory with random policies
for each episode do
  Initialize an output history $\langle y_0, \ldots, y_{-n+1} \rangle$ from an action history $\langle a_{-1}, \ldots, a_{-n} \rangle$
  Set $y_{sp} \leftarrow$ set-point from the user
  for each step $t$ of episode 0,1,...$T$ - 1 do
    Set $s \leftarrow \left\langle y_t, \ldots, y_{t-d_y}, a_{t-1}, \ldots, a_{t-d_a}, (y_t - y_{sp}) \right\rangle$
    Set $a_t \leftarrow \mu(s, W_a) + \mathcal{N}$
    Take action $a_t$, observe $y_{t+1}$ and $r$
    Set $s\prime \leftarrow \left\langle y_{t+1}, \ldots, y_{t+1-d_y}, a_t, \ldots, a_{t+1-d_a}, (y_{t+1} - y_{sp}) \right\rangle$
    Store tuple ($s$, $a_t$, $s'$, $r$) in RM
    Uniformly sample $M$ tuples from RM
    for $i$ = 1 to $M$ do
      Set $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}\left(s\prime^{(i)}, \mu\left(s\prime^{(i)}, W\prime_a\right), W\prime_c\right)$
    end for
    Set
$W_c \leftarrow W_c + \frac{\alpha_c}{M}\sum_{i=1}^{M}\left(\tilde{y}^{(i)} - Q^{\mu}\left(s^{(i)}, a^{(i)}, W_c\right)\right)\nabla_{W_c} Q^{\mu}\left(s^{(i)}, a^{(i)}, W_c\right)$
    for $i$ = 1 to $M$ do
      Calculate $\nabla_a Q^{\mu}\left(s^{(i)}, a, W_c\right)\big|_{a=a^{(i)}}$
      Clip $\nabla_a Q^{\mu}\left(s^{(i)}, a, W_c\right)\big|_{a=a^{(i)}}$ using (33)
    end for
    Set $W_a \leftarrow W_a + \frac{\alpha_a}{M}\sum_{i=1}^{M}\nabla_{W_a}\mu\left(s^{(i)}, W_a\right)\nabla_a Q^{\mu}\left(s^{(i)}, a, W_c\right)\Big|_{a=a^{(i)}}$
    Set $W_a' \leftarrow \tau W_a + (1 - \tau) W_a'$
    Set $W_c' \leftarrow \tau W_c + (1 - \tau) W_c'$
  end for
end for
```

As outlined in Algorithm 5, first we randomly initialize the parameters of the actor and critic networks (Step 2). We then create a copy of the network parameters and initialize the parameters of the target networks (Step 3). The final initialization step is to supply the RM with a large enough collection of tuples (s, a, s', r) on which to begin training the RL controller (Step 4). Each episode is preceded by two steps. First, we ensure the state definition s_t in Equation (25) is defined for the first d_y steps by initializing a sequence of actions $\langle a_{-1}, \ \dots, a_{-n}\rangle$ along with the corresponding sequence of outputs $\langle y_0, \ \dots, y_{-n+1}\rangle$ where $n \in \mathbb{N}$ is sufficiently large. A set-point is then fixed for the ensuing episode (Steps 6 and 7).

Next, for a given user-defined set-point, the actor $\mu(s, W_a)$ is queried, and a control action is implemented on the process (Steps 8–10). Implementing $\mu(s, W_a)$ on the process generates a new output y_{t+1}. The controller then receives a reward r, which is the last piece of information needed to store the updated tuple (s, a, r, s') in the RM (Steps 9–12). M uniformly sampled tuples are generated from the RM to update the actor and critic networks using batch gradient methods (Steps 14–23). Finally, for a fixed τ, we update the target actor and target critic networks in Steps 24–25. The above steps are then repeated until the end of the episode.

To ensure that the proposed method adequately explores the state and action spaces, the behavior policy for the agent is constructed by adding noise sampled from a noise process, $\mathcal{N}$, to our actor policy, μ (see Step 10 in Algorithm 5). Generally, $\mathcal{N}$ is to be chosen to suit the process. We use a zero-mean Ornstein–Uhlenbeck (OU) process [61] to generate temporally correlated exploration samples; however, the user can define their own noise process. We also allow for random initialization of system outputs at the start of an episode to ensure that the policy is not stuck in a local optimum. Finally, it is important to highlight that Algorithm 5 is a fully automatic algorithm that learns the control policy in real time by continuously interacting with the process.

6.4 | Network structure and implementation

The actor and critic neural networks each had two hidden layers with 400 and 300 units, respectively. We initialized the actor and critic networks using uniform Xavier initialization.[62] Each unit was modeled using a rectified nonlinearity activation function. Example 3 (Section 7.3) differs slightly: the output layers were initialized from a uniform distribution over [−0.003, 0.003], and we elected to use tanh activation for the second hidden layer. For all examples, the network hidden layers were batch normalized using the method in Ioffe and Szegedy[63] to ensure that the training is effective in processes where variables have different physical units and scales. Finally, we implemented the Adam optimizer in Kingma and Ba[64] to train the networks and regularized the weights and biases of the second hidden layer with an L_2 weight decay of 0.0001.

Algorithm 5 was scripted in `Python` and implemented on an `iMac` (3.2 GHz, Intel Core i5, 8 GB RAM). The deep networks for the actor and critic were built using `Tensorflow`.[65] For Example 3, we trained the networks on `Amazon g2.2xlarge EC2 instances`; the matrix multiplications were performed on a graphics processing unit (GPU) available in the `Amazon g2.2xlarge EC2 instances`. The hyperparameters for Algorithm 5 are process-specific and need to be selected carefully. We list in Table 1 the nominal values for hyperparameters used in all of our numerical examples (see Section 7); additional specifications are listed at the end of each example. An optimal selection of hyperparameters is crucial, but is beyond the scope of the current work.

TABLE 1 The hyperparameters used in Algorithm 5

Hyperparameter	Symbol	Nominal value
Actor learning rate	α_a	10^{-4}
Critic learning rate	α_c	10^{-4}
Target network update rate	τ	10^{-3}
OU process parameters	$\mathcal{N}$	$\theta = 0.15, \sigma = 0.30$

6.5 | DRL controller tuning

In this section, we provide some general guidelines and recommendations for implementing the DRL controller (see Algorithm 5) for set-point tracking problems.

a. **Episodes:** To begin an episode, we randomly sample an action from the action space and let the system settle under that action before starting the time steps in the main algorithm. For the examples considered in Section 7, each episode is terminated after 200 time steps or when $|y_{i,\,t} - y_{i,\,sp}| \ \leq \varepsilon$ for all $1 \leq i \leq n_y$ for 5 consecutive time steps, where ε is a user-defined tolerance. For processes with large time constants, it is recommended to run the episodes longer to ensure that the transient and steady-state dynamics are effectively captured in the input–output data.

b. **Rewards:** We consider the reward hypotheses (Equations (27) and (28)) in our examples. A possible variant of Equation (27) is given as follows:

$$r(s_t, a_t, s_{t+1}) = \begin{cases} c & \text{if } |y_{i,t} - y_{i,sp}| \leq \varepsilon \ \ \forall i \in \{1, 2, \dots, n_y\} \\ -\sum_{i=1}^{n_y} |y_{i,t} - y_{i,sp}| & \text{otherwise} \end{cases}, \tag{34}$$

where $y_{i,t} \in \mathcal{Y}$ are the ith output, $y_{i,sp} \in \mathcal{Y}$ is the set-point for the ith output, $c \in \mathbb{R}_+$ is a constant reward, and $\epsilon \in \mathbb{R}_+$ is a user-defined tolerance. According to Equation (34), the agent receives the maximum reward, c, only if all the outputs are within the tolerance limit set by the user. The potential advantage of Equation (34) over Equation (27) is that it can lead to faster tracking. On the other hand, the control strategy learned by the RL agent is generally more aggressive. Ultimately, we prefer the ℓ_1-reward function due the gradual increasing behavior of the function as the RL agent learns. Note that under this reward hypothesis, the tolerance ε in Equation (34) should be the same as the early termination tolerance described previously. These hypotheses are well suited for set-point tracking as they use tracking error to generate rewards; however, in other problems, it is imperative to explore other reward hypothesis. Some examples, include—(i) negative of 2-norm of the control error instead of 1-norm in Equation (27); (ii) weighted rewards for different outputs in a MIMO system; and (iii) economic reward function.

c. **Gradient clipping:** The gradient clipping in Equation (33) ensures that the DRL controller outputs are feasible and within the operating range. Also, defining tight lower and upper limits on the inputs has a significant effect on the learning rate. In general, setting tight limits on the inputs lead to faster convergence of the DRL controller.
d. **RL state:** The definition of an RL state is critical as it affects the performance of the DRL controller. As shown in Equation (25), we use a tuple of current and past outputs, past actions, and current deviation from the set-point as the RL state. While Equation (25) is well suited for set-point tracking problems, it is instructive to highlight that the choice of an RL state is not unique, as there may be other RL states for which the DRL controller could have a similar or better performance. In our experience, including additional relevant information in the RL states improves the overall performance and convergence rate of the DRL controller. The optimal selection of an RL state is not within the scope of the article; however, it certainty is an important area of research that warrants additional investigation.
e. **Replay memory:** We initialize the RM with M tuples (s, a, s', r) generated by simulating the system response under random actions, where M is the batch size used for training the networks. During initial learning, having a large RM is essential as it gives DRL controller access to the data generated from the old policy. Once the actor and critic networks are trained, and the system attains steady state, the RM need not be updated as the input–output data no longer contributes new information to the RM.
f. **Learning:** The DRL controller learns the policy in real time and continues to refine it as new experiences are collected. To reduce the computational burden of updating the actor and critic networks at each sampling time, it is recommended to “turn-off” learning once the set-point tracking is complete. For example, if the difference between the output and the set-point is within certain predefined threshold, the networks need not be updated. We simply terminate the episode once the DRL controller has tracked the set-point for five consecutive time steps.
g. **Exploration:** Once the actor and critic networks are trained, it is recommended that the agent stops exploring the state and action spaces. This is because adding exploration noise to a trained network adds unnecessary noise to the control actions as well as the system outputs.

7 | SIMULATION RESULTS

The efficacy of the DRL controller outlined in Algorithm 5 is demonstrated on four simulation examples. The first three examples include the following: a single-input single-output (SISO) paper-making machine; a MIMO high-purity distillation column; and a nonlinear MIMO heating, ventilation, and air conditioning (HVAC) system. The fourth example evaluates the robustness of Algorithm 5 to process changes.

7.1 | Example 1: Paper machine

In the pulp and paper industry, a paper machine is used to form the paper sheet and then remove water from the sheet by various means, such as vacuuming, pressing, or evaporating. The paper machine is divided into two main sections: wet-end and dry-end. The sheet is formed in the wet-end on a continuous synthetic fabric, and the dry-end of the machine removes the water in the sheet through evaporation. The paper passes over rotating steel cylinders, which are heated by superheated pressurized steam. Effective control of the moisture content in the sheets is an active area of research.

In this section, we design a DRL controller to control the moisture content in the sheet, denoted by y_t (in percentage), to a desired set-point, y_{sp}. There are several variables that affect the drying time of the sheet, such as machine speed, steam pressure, drying fabric, and so forth. For simplicity, we only consider the effect of steam flowrate, denoted by a_t (in m^3/hr) on y_t. For simulation purposes, we assume that a_t and y_t are related according to the following discrete-time transfer function:

$$G(z) = \frac{0.05z^{-1}}{1-0.6z^{-1}}. \tag{35}$$

Note that Equation (35) is strictly used for generating episodes and is not used in the design of the DRL controller. In fact, the user has complete freedom to substitute Equation (35) with any linear or nonlinear process model, in the case of simulations, or with the actual data from the paper machine, in the case of industrial implementation. Next, we implement Algorithm 5 with the hyperparameters listed in Tables 1 and 2 and additional specifications discussed at the end of the example (Table 3).

The generated data are then used for real-time training of the actor and critic networks in the DRL controller. Figure 5a captures the increasing trend of the cumulative reward per episode by showing the moving average across the number of set-points used in training. From Figure 5a, it is clear that as the DRL controller interacts with the process, it learns a progressively better policy, leading to higher rewards per episode. In less than 100 episodes, the DRL controller is able to learn a policy suitable for basic set-point tracking; it was able to generalize effectively in the remaining 400 episodes, as illustrated in Figures 5b,c and 6.

Figure 5b,c showcase the DRL controller, putting aside the physical interpretation of our system (Equation (35)). We highlight the smooth tracking performance of the DRL controller on a sinusoidal reference signal in Figure 5b, as this exemplifies the responsiveness and robustness of the DRL controller to tracking problems far different from the fixed collection of set-points in the interval [0,10] seen during training.

To illustrate this further, observe in Figure 6a that the DRL controller is able to track randomly generated set-points with little overshoot, both inside and outside the training output space $\mathcal{Y}$. Finally, noting that our action space for training is [0, 100], Figure 6b shows that the DRL controller is operating outside the action space to track the largest set-point.

7.1.1 | Further implementation details

We define the states according to Equation (26) and select the ℓ_1-reward function in Equation (27). Concretely,

$$r(s_t,a_t,s_{t+1}) = -|y_t - y_{sp}|, \tag{36}$$

where y_{sp} is uniformly sampled from {0, 0.5, 1, …, 9.5, 10} at the beginning of each episode. We added zero-mean Gaussian noise with variance of 0.01 to the outputs during training. We defined the early termination tolerance for the episodes to be $\varepsilon = .01$.

TABLE 2 Specifications for Example 1

Hyperparameter	Symbol	Nominal value
Episodes		500
Minibatch size	M	128
Replay memory size	K	5×10^4
Reward discount factor	γ	0.99
Action space	$\mathcal{A}$	[0, 100]
Output space	$\mathcal{Y}$	[0, 10]

TABLE 3 Specifications for Example 2

Hyperparameter	Symbol	Nominal value
Episodes		5,000
Minibatch size	M	128
Replay memory size	K	5×10^5
Reward discount factor	γ	0.95
Action space	$\mathcal{A}$	[0, 50]
Output space	$\mathcal{Y}$	[0, 5]

7.2 | Example 2: High-purity distillation column

In this section, we consider a two-product distillation column as shown in Figure 7. The objective of the distillation column is to split the feed, which is a mixture of a light and heavy components, into a distillate product, $y_{1,t}$, and a bottom product, $y_{2,t}$. The distillate contains most of the light component and the bottom product contains most of the heavy component. Figure 7 has multiple manipulated variables; however, for simplicity, we only consider two inputs: boilup rate, $u_{1,t}$, and the reflux rate, $u_{2,t}$. Note that the boilup and reflux rates have immediate effects on the product compositions. The distillation column shown in Figure 7 has a complex nonlinear dynamics; however, for simplicity, we linearize the model around the steady-state value. The transfer function for the distillation column is given as follows[66]:

$$G(s) = \begin{bmatrix} \frac{0.878}{\tau s+1} & -\frac{0.864}{\tau s+1} \\ \frac{1.0819}{\tau s+1} & -\frac{1.0958}{\tau s+1} \end{bmatrix}, \text{ where } \tau > 0. \tag{37}$$

Further, to include measurement uncertainties, we add a zero-mean Gaussian noise to Equation (37) with a variance of 0.01.

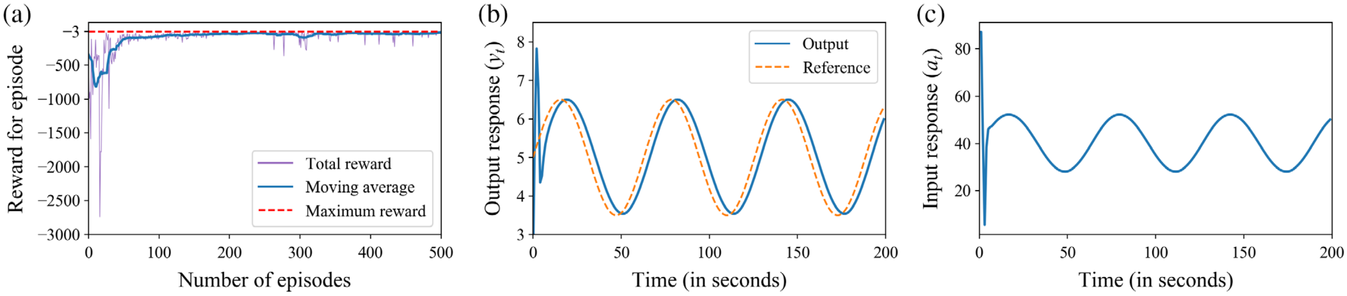

FIGURE 5 Simulation results for Example 1—(a) moving average with window size 21 (the number of distinct set-points used in training) for the total reward per episode; (b) tracking performance of the DRL controller on a sinusoidal reference signal; and (c) the input signal generated by the DRL controller

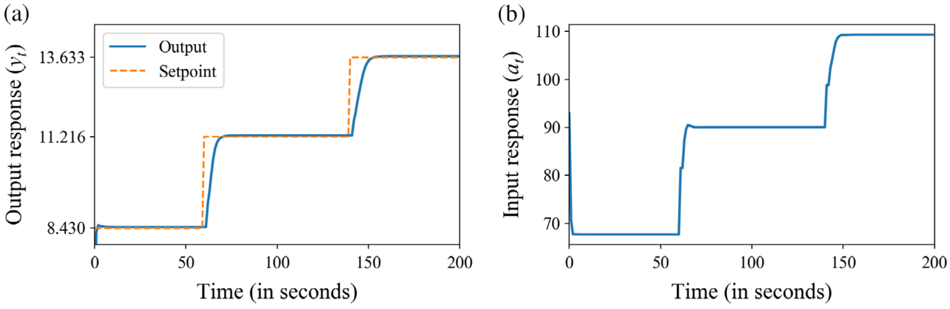

FIGURE 6 Simulation results for Example 1—(a) tracking performance of the DRL controller on randomly generated set-points outside of the training interval [0,10]; (b) the input signal generated by the DRL controller. DRL, deep reinforcement learning

Skogestad et al[66] showed that a distillation column with a model representation in Equation (37) is ill-conditioned. In other words, the plant gain strongly depends on the input direction, such that inputs in directions corresponding to high plant gains are strongly amplified by the plant, while inputs in directions corresponding to low plant gains are not. The issues around controlling ill-conditioned processes are well known to the community.[67,68]

Next, we implement Algorithm 5 with the reward hypothesis in Equation (27) and specifications listed in Table 3. Figure 8a,b shows the performance of the DRL controller in tracking the distillate and the bottom product to desired set-points for a variety of random initial starting points. Observe that starting from some initial conditions, the DRL controller successfully tracks the set-points within t = 75 minutes. Figures 8c,d show the boilup and reflux rates selected by the DRL controller, respectively. This example clearly highlights the efficacy of the DRL controller in successfully tracking a challenging, ill-conditioned MIMO system. Finally, it is instructive to highlight that the model in Equation (37) is strictly for generating episodes, and is not used by the DRL controller. The user has freedom to replace Equation (37) either with a complex nonlinear model or with the actual plant data.

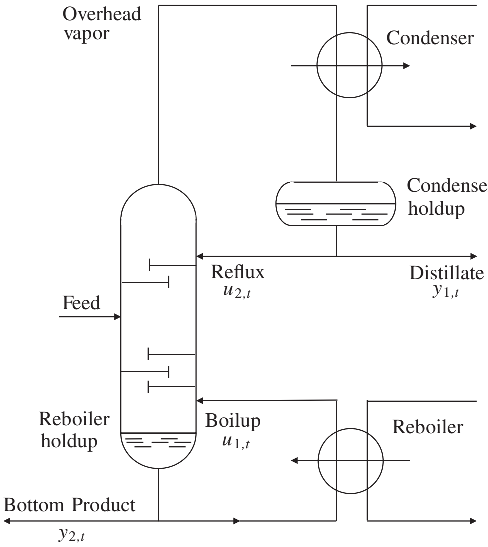

FIGURE 7 A schematic of a two-product distillation column in Example 2. Adapted from Reference 66

7.2.1 | Further implementation details

First, note that our system (Equation (37)) needs to be discretized when implementing Algorithm 5. We used the ℓ_1-reward hypothesis given in Equation (27) and defined the RL state as in Equation (26). To speed up learning, it is important to determine which pairs of set-points make sense in practice. To address this, we refer to Figure 9. We uniformly sample constant input signals from [0, 50] × [0, 50] and let the MIMO system (Equation (37)) settle for 50 time steps. Figure 9a shows these settled outputs. Clearly, we have very little flexibility when initializing feasible set-points in Algorithm 5. Therefore, we only select set-point pairs $(y_{1,sp}, y_{2,sp})$ where $y_{1,sp}, y_{2,sp} \in \{0, 0.5, 1, \ldots, 4.5, 5\}$ and $|y_{1,sp} - y_{2,sp}| \le 0.5$ Finally, Figure 9a shows the outputs in a restricted subset of the plane, [−5, 10] × [−5, 10]; the outputs can far exceed this even when sampling action pairs within the action space. A quick way to initialize an episode such that the outputs begin around the desired output space, [0, 5] × [0, 5], is to initialize action pairs according to linear regression such as the one shown in Figure 9b; we also added zero-mean Gaussian noise with variance 1 to this line. We reiterate that these steps are implemented purely to speed up learning, as the DRL controller will saturate and accumulate a large amount of negative reward when tasked with set-points outside the scope of Figure 9a. Further, this step does not solely rely on the process model (Equation (37)), as similar physical insights could be inferred from real plant data or by studying the reward signals associated with each

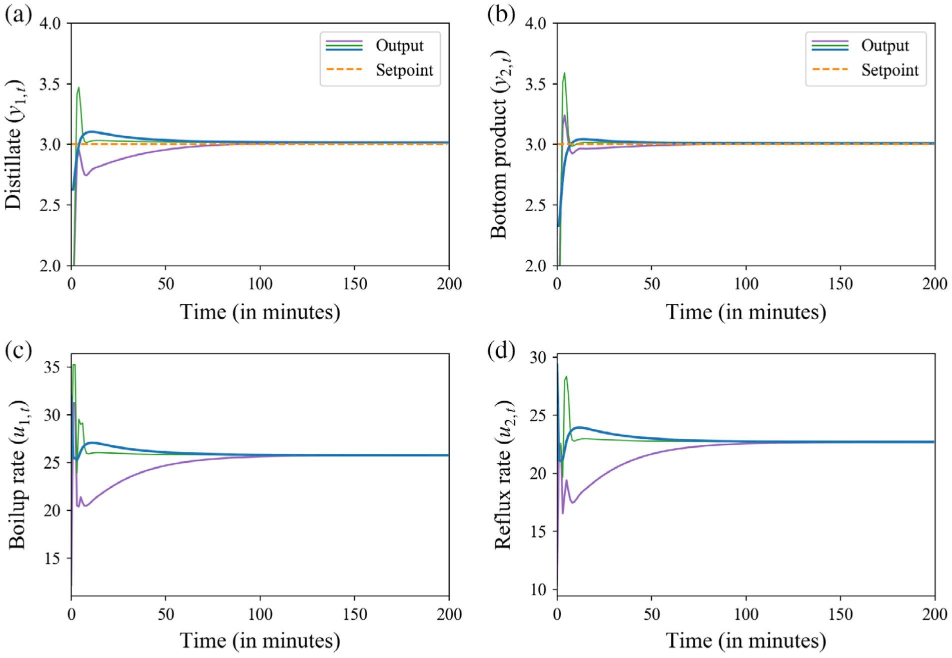

FIGURE 8 Simulation results for Example 2—(a) and (b) set-point tracking for distillate and bottom product, respectively, where the different colors correspond to different starting points; (c) and (d) boilup and reflux rates selected by the DRL controller, respectively

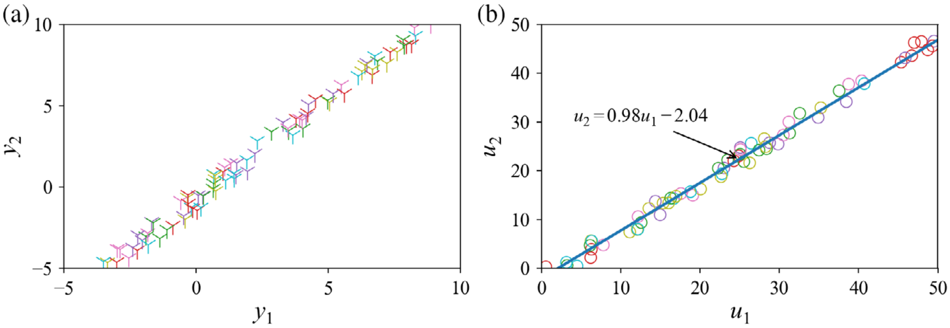

FIGURE 9 Set-point selection for Example 2—(a) the distribution of settled outputs after simulating the MIMO system (37) with uniformly sampled actions in [0,50] × [0,50]; (b) the corresponding action pairs such that the settled outputs are both within [0,5], along with a linear regression of these samples. MIMO, multi-input and multi-output

possible set-point pair as Algorithm 5 progresses. Figure 11 at the end of Example 3 demonstrates a possible approach for determining feasible set-points and effective hyperparameters.

7.3 | Example 3: HVAC system

Despite the advances in research on HVAC control algorithms, most field equipment is controlled using classical methods, such as hysteresis/on/off and PID controllers. Despite their popularity, these classical methods do not perform optimally. The high thermal inertia of buildings induces large time delays in the building dynamics, which cannot be handled efficiently by the simple on/off controllers. Furthermore, due to the high nonlinearity in building dynamics coupled with uncertainties such as weather, energy pricing, and so forth, these PID controllers require extensive re-tuning or auto-tuning capabilities,[69] which increases the difficulty and complexity of the control problem.[70]

Due to these challenging aspects of HVAC control, various advanced control methods have been investigated, ranging from nonlinear MPC[71] to optimal control.[72] Between these methods, the quality of control performance relies heavily on accurate process identification and modeling; however, large variations exist with building design, zone layout, long-term dynamics, and a wide range of operating conditions. In addition, large disturbance effects from external weather, occupancy schedule changes, and varying energy prices make process identification a very challenging problem.[70]

In this section, we demonstrate the efficacy of the proposed *model-free* DRL controller to control the HVAC system. First, to generate episodes, we assume that the HVAC system can be modeled as follows:[73]

$$\begin{aligned}T_{wo,t} &= T_{wo,t-1} + \theta_1 f_{w,t-1}(T_{wi,t-1} - T_{wo,t-1})\\ &+ (\theta_2 + \theta_3 f_{w,t-1} + \theta_4 f_{a,t-1})\left[T_{ai,t-1} - \overline{T}_{w,t-1}\right],\end{aligned} \tag{38a}$$

$$\begin{aligned}T_{ao,t} &= T_{ao,t-1} + \theta_5 f_{a,t-1}(T_{ai,t-1} - T_{ao,t-1})\\ &+ (\theta_6 + \theta_7 f_{w,t-1} + \theta_8 f_{a,t-1})(\overline{T}_{w,t-1} - T_{ai,t-1}) + \theta_9(T_{ai,t} - T_{ai,t-1}),\end{aligned} \tag{38b}$$

$$\overline{T}_{w,t} = 0.5\left[T_{wi,t} + T_{wo,t}\right], \tag{38c}$$

where $T_{wo} \in \mathbb{R}_+$ and $T_{ao} \in \mathbb{R}_+$ are the outlet water and discharge air temperatures (in °C), respectively; $T_{wi} \in \mathbb{R}_+$ and $T_{ai} \in \mathbb{R}_+$ are the inlet water and air temperatures (in °C), respectively; $f_w \in \mathbb{R}_+$ and $f_a \in \mathbb{R}_+$ are the water and air mass flow rates (in kg/s), respectively; and $\theta_i \in \mathbb{R}$ for i = 1, …, 9 are various physical parameters, with nominal values given in.[73] The water flow rate, f_w, is assumed to be related to the controller action as follows:

$$f_{w,t} = \theta_{10} + \theta_{11} a_t + \theta_{11} a_t^2 + \theta_{12} a_t^3, \tag{39}$$

where a_t is the control signal in terms of 12-bits and θ_j, for j = 10, 11, 12 are valve model parameters given in.[73]

In Equations (38a)–(38c), T_{wi}, T_{ai}, and f_a are disturbance variables that account for the set-point changes and other external disturbances. Furthermore, T_{wi}, $T_{ai,}$ and f_a are assumed to follow constrained random-walk models, such that

$$0.6 \le f_{a,t} \le 0.9;\;\; 73 \le T_{wi,t} \le 81;\;\; 4 \le T_{ai,t} \le 10, \tag{40}$$

for all $t \in \mathbb{N}$. For simplicity, we assume that the controlled variable is T_{ao} and the manipulated variable is f_w. The objective is to design a DRL controller to achieve desired discharge air temperature, $y_t \equiv T_{ao,t}$ by manipulating a_t.

Next, we implement Algorithm 5 to train the DRL controller for the HVAC system. Using the ℓ_1-reward hypothesis in Equation (27), we observed a persistent offset in T_{ao} to set-point changes. Despite increasing the episode length, the DRL controller was still unable to eliminate the offset with Equation (27). Next, we implement Algorithm 5 with the polar reward hypothesis in Equation (28). With Equation (28), the response to control actions was found to be less noisy and *offset-free*. In fact, Algorithm 5 performed much better with Equation (28) in terms of noise and offset reduction compared with Equation (27). Figure 10a shows the response of the discharge air temperature to a time-varying set-point change. Observe that the DRL controller is able to track the set-point changes in less than 50 s with *small* offset. Moreover, it is clear that the response to the change

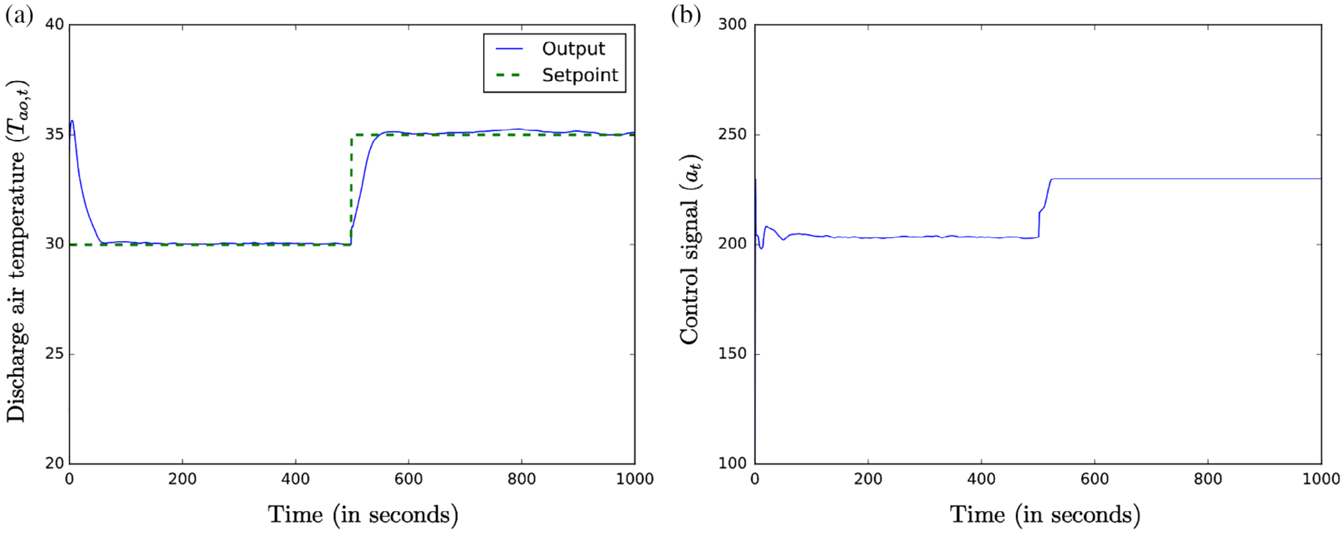

FIGURE 10 Simulation results for Example 3—(a) the response of the discharge air temperature to the set-point change and (b) the control signal output calculated by the DRL controller

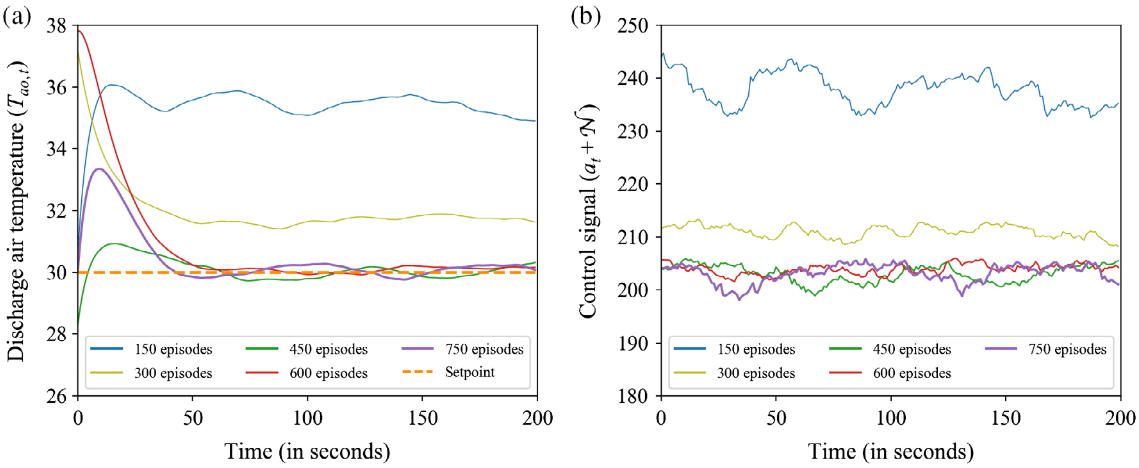

FIGURE 11 Snapshots during training of the input and output signals—(a) progression of set-point tracking performance of the DRL controller; (b) the respective controller actions taken by the DRL controller

TABLE 4 Specifications for Example 3

Hyperparameter	Symbol	Nominal value
Episodes		100,000
Mini-batch size	M	64
Replay memory size	K	10^6
Reward discount factor	γ	0.99
Action space	$\mathcal{A}$	[150, 800]
Output space	$\mathcal{Y}$	[30, 35]

in control action with Equation (28) is smooth. Finally, the control signal generated by the DRL controller is shown in Figure 10b. This example demonstrates the efficacy of Algorithm 5 in learning the control policy of complex nonlinear processes simply by interacting with the process in real time.

7.3.1 | Further implementation details

We contrast the exceedingly long training time listed in Table 4 to generate Figure 10 by noting from Figure 11 that the DRL controller was able to learn to track a fixed set-point, y_{sp} = 30, in fewer than 1,000 episodes. In general, training the DRL controller with a single set-point is a reasonable starting point for narrowing the parameter search when implementing Algorithm 5.

7.4 | Example 4: Robustness to process changes

In this section, we evaluate the robustness of DRL controller to adapt to abrupt process changes. We revisit the example from the pulp and paper industry in Section 7.1.

We run Algorithm 5 continuously with the same specifications as in Example 1 for a fixed set-point y_{sp} = 2. As shown in Figure 12a,b, in

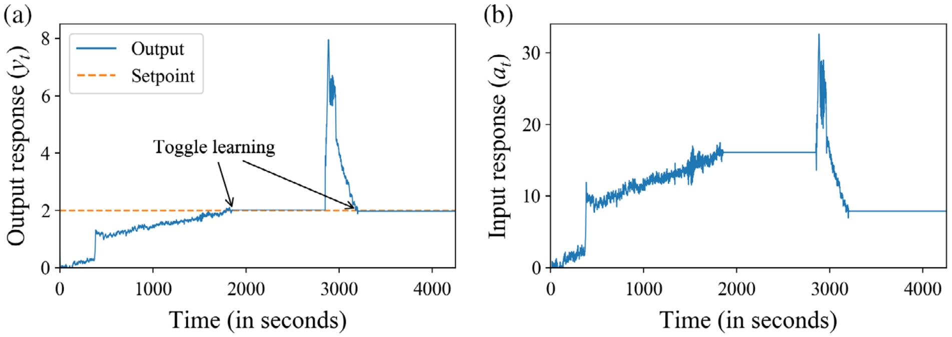

FIGURE 12 Simulation results for Example 4—(a) set-point tracking performance of the DRL controller under abrupt process changes; (b) the controller action taken by the DRL controller

the approximate interval 0 ≤ t ≤ 1900, the DRL agent is learning the control policy. Then at around t = 1900, we turn off the exploration noise and stop training the actor and critic networks because the moving average of the errors $|y_t$ - 2| across the past four time steps was sufficiently small (less than 0.0001 in our case). Next, at around t = 2,900, a sudden process change is introduced by doubling the process gain in Equation (35). Observe that as soon as the process change is introduced, the policy learned by the DRL controller is no longer valid. Consequently, the system starts to deviate from the set-point. However, since the DRL controller learns in real time, as soon as the process change is introduced, the controller starts re-learning a new policy. Observe that for the same set-point y_{sp} = 2, the DRL controller took about 400 s to learn the new policy after the model change was introduced (see Figure 12a,b for t ≥ 2,900). This demonstrates how the DRL controller first learns the process dynamics and then learns the control policy—both in real time, without access to a priori process information.

8 | COMPARISON WITH MPC

MPC is well established and widely deployed. The proposed DRL controller has a number of noteworthy differences.

a. **Process model:** An accurate process model is a key ingredient in any MPC. MPC relies on model-based predictions and model-based optimization to compute the control action at each sampling time.[74] In contrast, the DRL controller does not require any a priori access to a process model: it develops the control policy in real time by interacting with the process.
b. **Constraints:** Industrial control systems are subject to operational and environmental constraints that need to be satisfied at all sampling times. MPC handles these constraints by explicitly incorporating them into the optimization problem solved at each step.[75,76] In the DRL controller, constraints can be embedded directly in the reward function or enforced through gradient clipping—see Equation (33). The latter approach is softer: constraint violation may occur during training, but it is discouraged by triggering an enormous penalty in the reward signal. Robust mechanisms for safe operation of a fully trained system, and indeed, for safe operation during online training, are high priorities for further investigation.
c. **Prediction horizon:** MPC refers to a user-specified *prediction horizon* when optimizing the future response of a process. The duration of the MPC's planning horizon determines the influence of future states; it may be either finite or infinite depending on the application. The corresponding adjustment in the DRL controller comes through the discount factor $\gamma \in [0, 1]$ used to determine the present value of future rewards.
d. **State estimation:** The performance of MPC relies on an accurate estimate of the hidden process states over the length of the horizon. The state estimation problem is typically formulated as a filtering problem,[77,78] with Kalman filtering providing the standard tool for linear processes, while nonlinear systems remain an active area of research.[79,80] In this article, we consider the DRL controller for cases where the full system state is available for policy improvement. Extending the design to include filtering and/or observer design will be the subject of future work.
e. **Adaptation:** Practical control systems change over time. Maintaining acceptable performance requires responding to both changes and unknown disturbances. Traditional MPC-based systems include mechanisms for detecting model-plant mismatch and responding as appropriate—typically by initiating interventions in the process to permit reidentification of the process model. This process, which calls for simultaneous state and parameter estimation, can be both challenging and expensive.[81-83] Some recent variants of MPC, such as robust MPC and stochastic MPC, take model uncertainties into account by embedding them directly into the optimization problem.[76] The DRL controller, on the other hand, updates the parameters in the actor and critic networks at each sampling time using the latest experiences. This gives the DRL controller a self-tuning aspect, so that process operations remain nearly optimal with respect to the selected reward criterion.

9 | LIMITATIONS

Over the decades since MPC was first proposed, its theoretical foundations have become well established. Strong convergence and stability proofs are available for several MPC formulations, covering both linear and nonlinear systems.[84] At this comparatively early stage, the DRL controller comes with no such optimality guarantees. Indeed, the assumptions of discrete-time dynamics and full state observation are both obvious challenges demanding further investigation and development. (For example, while the temporal difference error for a discrete-time system can be computed in a model-free environment, the TD error formulation for a continuous-time system requires complete knowledge of its dynamics.[85]) Other issues of interest include the role of data requirements, computational complexity, nonconvexity, over- and under-fitting, performance improvement, hyperparameter selection, and initialization.

10 | CONCLUSIONS

We have developed an adaptive DRL controller for set-point tracking problems in discrete-time nonlinear processes. The DRL controller is a data-based controller based on a combination of RL and deep learning. As a model-free controller, this DRL controller learns the control policy in real time, simply by interacting with the process. The efficacy of the DRL controller has been demonstrated in set-point tracking problems on a SISO, a MIMO, and a nonlinear system with external disturbances. The DRL controller shows significant promise as an alternative to traditional model-based industrial controllers, even though some practical and theoretical challenges remain to be overcome. As both computing power and data volumes continue to increase, DRL controllers have the potential to become an important tool in process control.

ACKNOWLEDGMENTS

We would like to thank MITACS for financial support through the Globalink Graduate Fellowship and Natural Sciences and Engineering Research Council of Canada (NSERC). The first author would like to thank Kevin and Zilun for helpful discussions.

ENDNOTE

[1] To simplify notation, we drop the random variable in the conditional density and write $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|S_t = s_t, A_t = a_t)$.

ORCID

Aditya Tulsyan https://orcid.org/0000-0002-8915-2187
R. Bhushan Gopaluni https://orcid.org/0000-0002-4321-0468

REFERENCES

1. Tulsyan A, Garvin C, Undey C. Machine-learning for biopharmaceutical batch process monitoring with limited data. *IFAC-PapersOnLine*. 2018;51(18):126-131.
2. Tulsyan A, Garvin C, Ündey C. Advances in industrial biopharmaceutical batch process monitoring: machine-learning methods for small data problems. *Biotechnol Bioeng*. 2018;115(8):1915-1924.
3. Tulsyan A, Garvin C, Undey C. Industrial batch process monitoring with limited data. *J Process Control*. 2019;77:114-133.
4. Kano M, Ogawa M. The state of the art in advanced chemical process control in Japan. *IFAC Proc Vol*. 2009;42(11):10-25.
5. Tulsyan A, Khare SR, Huang B, Gopaluni RB, Forbes JF. Bayesian identification of non-linear state-space models: part I-input design. *IFAC Proc Vol*. 2013;46(32):774-779.
6. Seborg DE, Edgar TF, Shah SL. Adaptive control strategies for process control: a survey. *AIChE J*. 1986;32(6):881-913.
7. Krstic M, Kanellakopoulos I, Kokotovic PV. *Nonlinear and Adaptive Control Design*. New York, NY: John Wiley & Sons; 1995.
8. Mailleret L, Bernard O, Steyer JP. Nonlinear adaptive control for bioreactors with unknown kinetics. *Automatica*. 2004;40(8):1379-1385.
9. Guan C, Pan S. Adaptive sliding mode control of electro-hydraulic system with nonlinear unknown parameters. *Control Eng Pract*. 2008;16(11):1275-1284.
10. Spielberg SPK, Gopaluni RB, Loewen PD. Deep reinforcement learning approaches for process control. Paper presented at: In 2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP); Taipei, Taiwan: IEEE; 2017: 201-206. Accessed May 28, 2017.
11. Sutton RS, Barto AG. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press; 1998.
12. Bertsekas DP, Tsitsiklis JN. Neuro-dynamic programming: an overview. Paper presented at: Proceedings of the 34th IEEE Conference on Decision and Control. 1995; Piscataway, NJ; p. 560–564.
13. Bertsekas DP. *Dynamic Programming and Optimal Control*. MA: Athena Scientific; 2005.
14. Powell WB. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York, NY: John Wiley & Sons; 2007.
15. Sugiyama M. *Statistical Reinforcement Learning: Modern Machine Learning Approaches*. Florida: CRC Press; 2015.
16. Bellman R. *Dynamic Programming*. New York, NY: Dover Publications; 2003.
17. Lehnert L, Precup D. Policy Gradient Methods for Off-policy Control. arXiv Preprint, arXiv:151204105; 2015.
18. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous Control with Deep Reinforcement Learning. arXiv Preprint, arXiv:150902971; 2015.
19. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. arXiv Preprint, arXiv:13125602; 2013.
20. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature*. 2015;518:529-533.
21. Pednault E, Abe N, Zadrozny B. Sequential cost-sensitive decision making with reinforcement learning. Paper presented at: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2002; Edmonton, Canada; p. 259–268.
22. Tesauro G. Temporal difference learning and TD-gammon. *Commun ACM*. 1995;38(3):58-68.
23. Silver D, Huang A, Maddison CJ, et al. Mastering the game of go with deep neural networks and tree search. *Nature*. 2016;529:484-489.
24. Badgwell TA, Lee JH, Liu KH. Reinforcement learning–overview of recent progress and implications for process control. *Computer Aided Chemical Engineering*. Vol 44. Amsterdam, Netherlands: Elsevier; 2018:71-85.
25. Lee JH, Lee JM. Approximate dynamic programming based approach to process control and scheduling. *Comput Chem Eng*. 2006;30(10–12):1603-1618.
26. Lewis FL, Vrabie D, Vamvoudakis KG. Reinforcement learning and feedback control: using natural decision methods to design optimal adaptive controllers. *IEEE Control Syst*. 2012;32(6):76-105.
27. Morinelly JE, Ydstie BE. Dual mpc with reinforcement learning. *IFAC PapersOnLine*. 2016;49(7):266-271.
28. Lee JM, Kaisare NS, Lee JH. Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. *J Process Control*. 2006;16(2):135-156.
29. Lee JM, Lee JH. Neuro-dynamic programming method for MPC1. *IFAC Proc Vol*. 2001;34(25):143-148.

30. Kaisare NS, Lee JM, Lee JH. Simulation based strategy for nonlinear optimal control: application to a microbial cell reactor. *Int J Robust Nonlin Control*. 2003;13(3–4):347-363.
31. Lee JM, Lee JH. Simulation-based learning of cost-to-go for control of nonlinear processes. *Korean J Chem Eng*. 2004;21(2): 338-344.
32. Lee JM, Lee JH. Approximate dynamic programming-based approaches for input–output data-driven control of nonlinear processes. *Automatica*. 2005;41(7):1281-1288.
33. Al-Tamimi A, Lewis FL, Abu-Khalaf M. Discrete-time nonlinear HJB solution using approximate dynamic programming: convergence proof. *IEEE Trans Syst Man Cybern B*. 2008;38(4):943-949.
34. Si J, Wang YT. Online learning control by association and reinforcement. *IEEE Trans Neural Netw*. 2001;12(2):264-276.
35. Heydari A, Balakrishnan SN. Finite-horizon control-constrained nonlinear optimal control using single network adaptive critics. *IEEE Trans Neural Netw Learn Syst*. 2013;24(1):145-157.
36. Wang D, Liu D, Wei Q, Zhao D, Jin N. Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming. *Automatica*. 2012;48(8):1825-1832.
37. Vrabie D, Lewis F. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. *Neural Netw*. 2009;22(3):237-246.
38. Vamvoudakis KG, Lewis FL. Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica*. 2010;46(5):878-888.
39. Liu D, Wang D, Li H. Decentralized stabilization for a class of continuous-time nonlinear interconnected systems using online learning optimal control approach. *IEEE Trans Neural Netw Learn Syst*. 2014;25(2):418-428.
40. Mu C, Wang D, He H. Novel iterative neural dynamic programming for data-based approximate optimal control design. *Automatica*. 2017;81:240-252.
41. Luo B, Wu HN, Huang T, Liu D. Data-based approximate policy iteration for affine nonlinear continuous-time optimal control design. *Automatica*. 2014;50(12):3281-3290.
42. Wang D, Liu D, Zhang Q, Zhao D. Data-based adaptive critic designs for nonlinear robust optimal control with uncertain dynamics. *IEEE Trans Syst Man Cybern: Syst*. 2016;46(11):1544-1555.
43. Tang W, Daoutidis P. Distributed adaptive dynamic programming for data-driven optimal control. *Syst Control Lett*. 2018;120: 36-43.
44. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; Lake Tahoe, Nevada; 2012. p. 1097–1105.
45. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M. Deterministic policy gradient algorithms. Paper presented at: Proceedings of the 31st International Conference on Machine Learning; 2014; Beijing, China.
46. Watkins CJCH. Learning from Delayed Rewards. [PhD thesis]. Cambridge: King's College; 1989.
47. Watkins CJ, Dayan P. Q-learning. *Machine Learning*. 1992;8(3–4): 279-292.
48. Sutton RS. Generalization in reinforcement learning: successful examples using sparse coarse coding. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; 1996; Denver. p. 1038–1044.
49. Lazaric A, Ghavamzadeh M, Munos R. Finite-sample analysis of LSTD. Paper presented at: Proceedings of the 27th International Conference on Machine Learning; 2010; Haifa, Israel. p. 615–622.
50. Tsitsiklis JN, Van Roy B. Analysis of temporal-diffference learning with function approximation. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; 1997; Denver. p. 1075–1081.
51. Baird L. Residual algorithms: reinforcement learning with function approximation. *Machine Learning Proceedings 1995*. Amsterdam, Netherlands: Elsevier; 1995:30-37.
52. Fairbank M, Alonso E. The Divergence of Reinforcement Learning Algorithms with Value-Iteration and Function Approximation. arXiv preprint arXiv:11074606; 2011.
53. Kober J, Bagnell JA, Peters J. Reinforcement learning in robotics: a survey. *Int J Rob Res*. 2013;32(11):1238-1274.
54. Sutton RS, McAllester DA, Singh SP, Mansour Y. Policy gradient methods for reinforcement learning with function approximation. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; Vancouver, Canada; 2000. p. 1057–1063.
55. Degris T, White M, Sutton RS. Off-policy Actor-Critic. arXiv Preprint, arXiv:12054839; 2012.
56. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement Learning*. Berlin, Germany: Springer; 1992:5-32.
57. Riedmiller M, Peters J, Schaal S. Evaluation of policy gradient methods and variants on the cart-pole benchmark. Paper presented at: Proceedings of the International Symposium on Approximate Dynamic Programming and Reinforcement Learning; 2007; Honolulu, Hawaii. p. 254–261.
58. Konda VR, Tsitsiklis JN. Actor-critic algorithms. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; 2000; Denver. p. 1008–1014.
59. Bhatnagar S, Ghavamzadeh M, Lee M, Sutton RS. Incremental natural actor-critic algorithms. Paper presented at: Proceedings of the Advances in Neural Information Processing Systems; 2008; Vancouver, Canada. p. 105–112.
60. Hausknecht M, Stone P. Deep Reinforcement Learning in Parameterized Action Space. arXiv Preprint, arXiv:151104143; 2015.
61. Uhlenbeck GE, Ornstein LS. On the theory of the Brownian motion. *Phys Rev*. 1930;36(5):823-841.
62. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. Paper presented at: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics; Sardinia, Italy; 2010; p. 249–256.
63. Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv Preprint, arXiv: 150203167; 2015.
64. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. arXiv Preprint, arXiv:14126980; 2014.
65. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv Preprint, arXiv:160304467; 2016.
66. Skogestad S, Morari M, Doyle JC. Robust control of ill-conditioned plants: high-purity distillation. *IEEE Trans Autom Control*. 1988;33(12): 1092-1105.
67. Waller JB, Böling JM. Multi-variable nonlinear MPC of an ill-conditioned distillation column. *J Process Control*. 2005;15(1):23-29.
68. Mollov S, Babuska R. Analysis of interactions and multivariate decoupling fuzzy control for a binary distillation column. *Int J Fuzzy Syst*. 2004;6(2):53-62.
69. Wang YG, Shi ZG, Cai WJ. PID autotuner and its application in HVAC systems. Paper presented at: Proceedings of the American Control Conference, 2001; Arlington; p. 2192–2196.
70. Wang Y, Velswamy K, Huang B. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. *Processes*. 2017;5(3):1-18.
71. Moradi H, Saffar-Avval M, Bakhtiari-Nejad F. Nonlinear multivariable control and performance analysis of an air-handling unit. *Energy Build*. 2011;43(4):805-813.
72. Greensfelder EM, Henze GP, Felsmann C. An investigation of optimal control of passive building thermal storage with real time pricing. *J Build Perform Simul*. 2011;4(2):91-104.

73. Underwood DM, Crawford RR. Dynamic nonlinear Modeling of a hot-water-to-air heat exchanger for control applications. *ASHRAE*. 1991; 97(1):149-155.
74. Pon Kumar SS, Tulsyan A, Gopaluni B, Loewen P. A Deep learning architecture for predictive control. Paper presented at: Proceedings of the 10th IFAC Conference on the Advanced Control of Chemical Processes; 2018; Shenyang, China.
75. Wallace M, Pon Kumar SS, Mhaskar P. Offset-free model predictive control with explicit performance specification. *Ind Eng Chem Res*. 2016;55(4):995-1003.
76. Lee JH, Wong W. Approximate dynamic programming approach for process control. *J Process Control*. 2010;20(9):1038-1048.
77. Tulsyan A, Tsai Y, Gopaluni RB, Braatz RD. State-of-charge estimation in lithium-ion batteries: a particle filter approach. *J Power Sources*. 2016;331:208-223.
78. Tulsyan A, Huang B, Gopaluni RB, Forbes JF. A particle filter approach to approximate posterior Cramér-Rao lower bound: the case of hidden states. *IEEE Trans Aerosp Electron Syst*. 2013;49(4):2478-2495.
79. Tulsyan A, Gopaluni RB, Khare SR. Particle filtering without tears: a primer for beginners. *Comput Chem Eng*. 2016;95(12):130-145.
80. Tulsyan A, Huang B, Gopaluni RB, Forbes JF. Performance assessment, diagnosis, and optimal selection of non-linear state filters. *J Process Control*. 2014;24(2):460-478.
81. Tulsyan A, Khare S, Huang B, Gopaluni B, Forbes F. A switching strategy for adaptive state estimation. *Signal Process*. 2018;143: 371-380.
82. Tulsyan A, Huang B, Gopaluni RB, Forbes JF. On simultaneous on-line state and parameter estimation in non-linear state-space models. *J Process Control*. 2013;23(4):516-526.
83. Tulsyan A, Huang B, Gopaluni RB, Forbes JF. Bayesian identification of non-linear state-space models: part II-error analysis. *IFAC Proc Vol*. 2013;46(32):631-636.
84. Mayne DQ. Model predictive control: recent developments and future promise. *Automatica*. 2014;50(12):2967-2986.
85. Bhasin S. *Reinforcement Learning and Optimal Control Methods for Uncertain Nonlinear Systems*. Gainesville, Florida: University of Florida; 2011.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Spielberg S, Tulsyan A, Lawrence NP, Loewen PD, Bhushan Gopaluni R. Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE J*. 2019;e16689. https://doi.org/10.1002/aic.16689