Where Reinforcement Learning Meets Process Control: Review and Guidelines

Authors:

Ruan de Rezende Faria, Bruno Didier Olivier Capron, Argimiro Resende Secchi, Maurício B. de Souza Jr

Date Submitted: 2023-02-21

Keywords: Markov decision process, imitation learning, transfer learning, process optimization

Abstract:

This paper presents a literature review of reinforcement learning (RL) and its applications to process control and optimization. These applications were evaluated from a new perspective on simulation-based offline training and process demonstrations, policy deployment with transfer learning (TL) and the challenges of integrating it by proposing a feasible approach to online process control. The study elucidates how learning from demonstrations can be accomplished through imitation learning (IL) and reinforcement learning, and presents a hyperparameter-optimization framework to obtain a feasible algorithm and deep neural network (DNN). The study details a batch process control experiment using the deep-deterministic-policy-gradient (DDPG) algorithm modified with adversarial imitation learning.

Record Type: Published Article

Submitted To: LAPSE (Living Archive for Process Systems Engineering)

Citation (overall record, always the latest version):	LAPSE:2023.0840
Citation (this specific file, latest version):	LAPSE:2023.0840-1
Citation (this specific file, this version):	LAPSE:2023.0840-1v1

DOI of Published Version: https://doi.org/10.3390/pr10112311

processes

MDPI

Review

Where Reinforcement Learning Meets Process Control: Review and Guidelines

Ruan de Rezende Faria [1,*], Bruno Didier Olivier Capron [1], Argimiro Resende Secchi [2] and Maurício B. de Souza, Jr. [1,2]

[1] Escola de Química, EPQB, Universidade Federal do Rio de Janeiro, Rio de Janeiro 21941-909, Brazil
[2] Programa de Engenharia Química, PEQ/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro 21941-972, Brazil
* Correspondence: rrfaria@eq.ufrj.br

Abstract: This paper presents a literature review of reinforcement learning (RL) and its applications to process control and optimization. These applications were evaluated from a new perspective on simulation-based offline training and process demonstrations, policy deployment with transfer learning (TL) and the challenges of integrating it by proposing a feasible approach to online process control. The study elucidates how learning from demonstrations can be accomplished through imitation learning (IL) and reinforcement learning, and presents a hyperparameter-optimization framework to obtain a feasible algorithm and deep neural network (DNN). The study details a batch process control experiment using the deep-deterministic-policy-gradient (DDPG) algorithm modified with adversarial imitation learning.

Keywords: Markov decision process; imitation learning; transfer learning; process optimization

Citation: Faria, R.d.R.; Capron, B.D.O.; Secchi, A.R.; de Souza, M.B., Jr. Where Reinforcement Learning Meets Process Control: Review and Guidelines. *Processes* **2022**, *10*, 2311. https://doi.org/10.3390/pr10112311

Academic Editors: Jean-Pierre Corriou and Francisco Vázquez

Received: 12 September 2022
Accepted: 1 November 2022
Published: 6 November 2022

1. Introduction

Reinforcement learning is a machine-learning (ML) technique characterized by an agent capable of self-learning in an environment guided only by numerical rewards [1]. Historically, its evolution was mainly related to artificial intelligence (AI), but it cannot be ignored that RL is directly derived from optimal control theory [2,3]. Nevertheless, it was a long time before the process-control community realized the potential of this technique to address control, as reported in Hoskins and Himmelblau [4]. There was a reduction in research incentives and state-of-the-art development at that time, as the results showed a technique with inferior performance to the proportional–integral–derivative controller (PID) that was also algorithmically complex, data-driven, and a black-box model.

This situation began to change in 2012 due to the consolidation of deep-learning (DL) theory, in which deep neural networks (DNN) began to be used as feature extractors [5,6]. This allowed RL in problems with high-dimensional state space (i.e., deep reinforcement learning (DRL)) such as, for example, cyber-physical systems (e.g., robot control; see Wulfmeier et al. [7], Peng et al. [8]), fixed, strict and complex environments (e.g., AlphaGO; see Silver et al. [9]; and AI in gaming; see Mnih et al. [10]), and large-scale environments (e.g., Vinyals et al. [11]).

For these reasons, as reported in [12,13], the number of publications on RL applied to process control began to grow again. The majority of the reported works are related to batch operation (e.g., [14–17]), although there are also applications for cases of continuous operation (e.g., [18–20]).

This paper presents a literature review on RL and addresses the online implementation and maintenance of RL techniques to control and optimize batch and continuous processes. At this point, model-based predictive control (MPC) theory and real-time optimization (RTO) are well-established technologies for application in chemical processes considering a decision-making scale of minutes and hours, respectively [21]. However, they depend

on complex models with periodic recalibration. Decisions are based on open-control-loop simulations, with the process and economic constraints encompassing an entire unit. Under these conditions, such integration may be unfeasible when the approach does not consider the model mismatch, presents a long waiting time to reach the steady state and demands high computational costs associated with the computation of the control actions [22–24].

Likewise, developing an RL agent (control policy) for application and maintenance in an online process faces challenges similar to those discussed above. However, RL can result in adaptive and low-computational-cost control (i.e., after training). Theoretically, it seems to be an ideal technology for application in process control. However, it depends on the evolution of theory to incorporate process constraints and to ensure a level of control stability. It also needs extensive offline learning and requires adjustment of the deep neural network and algorithms' hyperparameters [1,25].

To take into account process constraints, three state-of-the-art developments can be mentioned: (1) modifying the Markov decision process (MDP) framework to incorporate process constraints implicitly (e.g., Pan et al. [26]); (2) combining imitation learning and reinforcement learning to learn from demonstrations (e.g., Mowbray et al. [27] used inverse reinforcement learning); (3) combining MPC and RL to obtain a control policy with the mutual benefits of the synergistic integration of both methodologies (e.g., Shah and Gopal [28], Alhazmi et al. [29], Kim et al. [30]).

Some review papers have addressed RL techniques applied to process control. Buşoniu et al. [13] established a detailed review of the methodology. Badgwell et al. [31] reviewed recent progress and implications for process control. Nian et al. [12] covered applications to the process industry. Görges [32] analyzed the relationships between model-based nonlinear predictive control (NMPC) and RL through tutorials. The present article reviews RL from a new perspective on simulation-based offline training and process demonstrations, policy deployment with transfer learning and the challenges of integrating it, by proposing a feasible approach to online process control. A control experiment elucidates how learning from demonstrations can be accomplished through IL and RL. A hyperparameter-optimization framework is proposed to obtain a feasible algorithm and DNN.

This review starts in Section 2 with RL theory, DRL and the state-of-the-art algorithms for DRL. Section 3 details simulation-based offline training and process demonstrations, and policy deployment with transfer learning. Section 4 presents the challenges of implementing reinforcement learning for process control. Section 5 presents an approach based on IL and RL for a batch-process-control experiment. The review is concluded in Section 6.

2. Reinforcement Learning

Four distinct but complementary phases mark the historical evolution of the RL methodology; in chronological order, they are [1,33]:

1. Definition of the term from animal psychology;
2. Analysis for optimal control theory and machine learning;
3. Evolution of training procedures and pattern recognition;
4. Development of DNN, powerful hardware, data availability and more stable algorithms.

References that are important for the evolution of RL theory or have influenced the current literature are listed in Table 1. Difficulties in defining the learning elements, as well as creating and implementing algorithms, were the main challenges during Phase 2. Howard [34] was the first to propose a viable RL algorithm based on dynamic programming (DP). In Phase 3, there was a return of interest in the field of AI in general [1], where developments for RL remain current, specifically with Q-learning and REINFORCE algorithms and stochastic RL theory. Phase 4 is when RL algorithms incorporated DNN, in addition to methodologies to store information in memory (i.e., buffer replay), in order to stabilize training and improve convergence.

Table 1. Several historical references and application contexts.

Author	Main Topic	Approach
	(Phase 1)	
[35]	Animal psychology	Definition of RL
	(Phase 2)	
[2,3]	Optimal control	MDP, DP and Bellman's equation
[36]	AI	Discussed RL models
[34]	AI	DP algorithm (policy iteration)
[37]	AI	Pointed out directions for the evolution of RL
	(Phase 3)	
[38]	AI	Neuro-like network
[39]	AI	Temporal difference (TD)
[40]	AI	Q-learning algorithm
[41]	AI	Stochastic RL algorithm
[42]	AI	REINFORCE algorithm
	(Phase 4)	
[10]	AI	RL in games (Atari)
[9]	AI	AlphaGo

2.1. Basics of RL

ML technologies are divided into three core classes: supervised learning, unsupervised learning and reinforcement learning [25,43]. In supervised learning, knowledge about the case study results from labeling its input and output features. An example of this is classification and regression tasks where only cause (X) and effect (Y) are known; their relationship is approximated ($Y = f(X)$) using some data-driven technique (e.g., neural networks) that has adequate generalization and accuracy when applied to the case study. On the other hand, unsupervised learning obtains information about the case study only through knowledge of the cause (X), and it is not necessary to include knowledge in the form of a "teacher" to label its effect (Y). A typical application is clustering [44].

RL differs significantly from those technologies discussed above. According to Sutton and Barto [1], this is because RL does not require external interference in the form of a "teacher" to explore its environment, and is also not limited to learning how input data is distributed ($p(X)$). Instead, it learns the best way to keep up with a given task through repeated interactions with its environment. This makes the task complex, as more elements have to be studied to characterize the problem. However, it can make machine learning automated.

Defining an RL problem depends on understanding its essential elements: agent, environment and reward. A widely studied example is the bandit problem. This is a problem in which a fixed limited set of resources must be allocated between competing choices (i.e., actions (a_t)) in a way that maximizes their expected gain. In other words, the agent's objective is to maximize the rewards sum along the trajectory $h = (a_0, a_1, \cdots, a_T)$ [45].

Although the definition of the RL problem seems to be simple, selecting which actions are the best without any a priori knowledge about the environment is a challenging task. Sutton and Barto [1] detailed that it is necessary to estimate the value of the actions (A_t) so that it is possible to predict the most valuable actions in the long term. However, through repeated executions of the bandit problem (episode), just selecting the k actions with the highest value (exploitation) will likely lead to a sub-optimal expected sum of rewards (Equation (1)), since RL does not explore enough combinations of actions throughout the episodes, and neither approaches the expectation of Equation (1). The exploration–exploitation trade-off dilemma is a classic problem in the RL literature. An alternative would be to select some actions randomly at a specific rate, thus preventing the selection of greedy actions and furthering exploration of the environment.

$$q^*(a) = E[R(h)|A_t = a_t] \tag{1}$$

2.2. Mathematical Background

2.2.1. Definition

Mathematically, the RL problem is formulated as an MDP (Figure 1): a stochastic sequential decision-making problem through the interaction of an agent with the environment [3]. That is, the agent, faced with the current representation of the state of the environment $s_t \in S$, executes strategies (*policies*) and implements the best action available at that moment $a_t \in A$, which will result in the transition of the environment to a new state $s_{t+1} \in S'$, evaluated employing a reinforcement sign (Equation (2)).

$$r_t = r(s_t, a_t, s_{t+1}) \in R_t \tag{2}$$

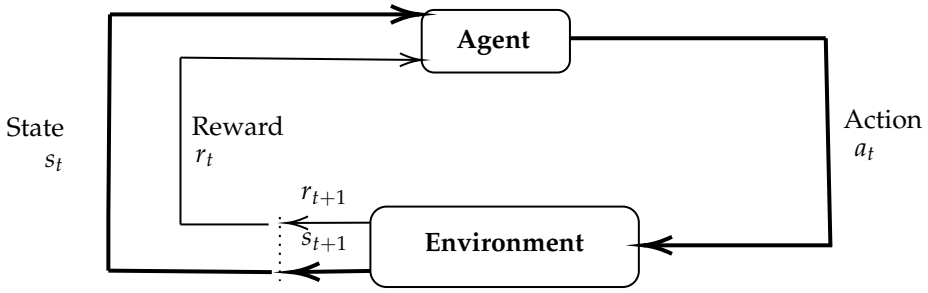

Figure 1. Simplified outline of the RL process.

For the case of continuous variables, the stochastic elements of this process include the initial condition of the state s, in the form of Equation (3). In this equation, the probability density function returns probability values for all possible states belonging to the set S.

$$\begin{aligned} p(s) \geq 0, \forall s \in S_t \\ \int_{s \in S_t} p(s) ds = 1 \end{aligned} \tag{3}$$

The probability transition from state s to state s' when action a is taken (state transition) defines the conditional probability density function $p(s'|s,a)$ (Equation (4)).

$$\begin{aligned} p(s'|s,a) \geq 0, \forall s, s' \in S_{t+1}, \forall a \in A_t \\ \int_{s' \in S_{t+1}} p(s'|s,a) ds' = 1, \forall s \in S_t, \forall a \in A_t \end{aligned} \tag{4}$$

The decision made by the agent is determined by a policy (π), which aims to map states to actions. In other words, it is a rule for deciding what to do given the current state of the environment, and is robust enough to specify what to do in any situation and for the entire state space. When using a deterministic policy, the action to be taken in each state is unique, as shown in Equation (5).

$$\pi(s) \in A_t, \forall s \in S_t \tag{5}$$

On the other hand, when the action to be taken in each state is stochastic, such a policy is a function of the conditional probability density of taking action a in-state s, as shown in Equation (6) [1,33,40].

$$\begin{aligned} \pi(a|s) \geq 0, \forall s \in S_t, \forall a \in A_t \\ \int_{a \in A_t} \pi(a|s) da = 1, \forall s \in S_t \end{aligned} \tag{6}$$

2.2.2. Optimization Objective

The sampling of the trajectory (h) of the MDP is illustrated in Figure 2. This sampling is an extension of Markov chain processes, in which the current state of the system (s_t) must contain all the information needed to decide which action (a_t) to take, resulting in the subsequent transition to the new state s_{t+1}, making it unnecessary to know about actions, states and rewards selected in the past. Hence, the agent's objective is to maximize the reward sum along the trajectory h, known as return [40]. Generally, the choice is between three alternatives known as total reward, average reward and discounted reward, the latter being the most common choice according to Sutton and Barto [1]. The corresponding return associated to a trajectory h is defined by Equation (7).

$$R(h) = \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \tag{7}$$

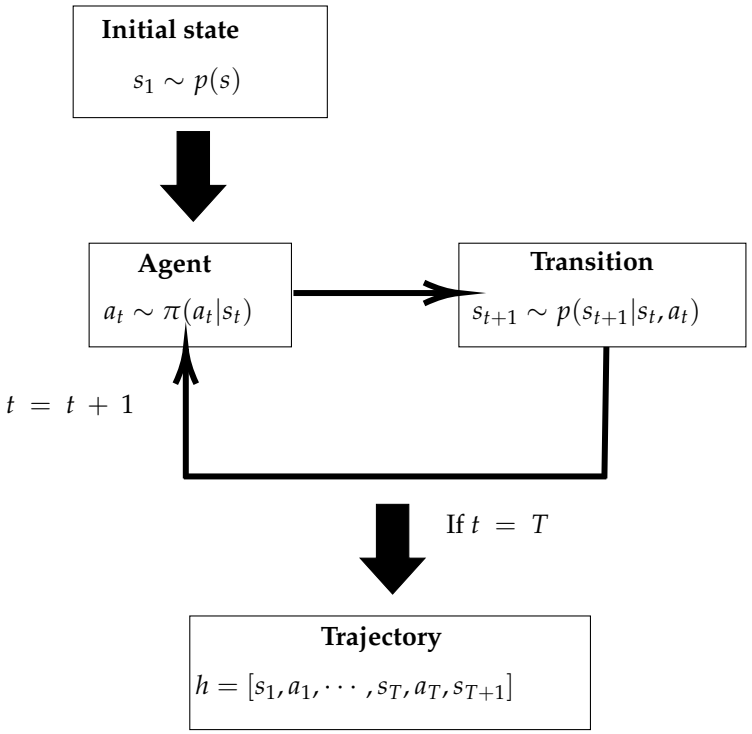

Figure 2. Trajectory sample generation.

In this equation, γ is a parameter known as the discount factor (ranging between 0 and 1), which determines the present value of future rewards. When $\gamma = 0$, the agent acts "greedily", maximizing the reward obtained in the next step, whereas if $\gamma = 1$, the agent will emphasize all future rewards equally, corresponding to the total reward alternative, or to the average reward when this return value is divided by T. Additionally, values of γ much less than 1 can make future reward values eventually negligible [31,40].

According to the logical sequence of Figure 2, the objective of RL-based methodologies is to learn an optimal policy that, regardless of the chosen initial state s_1, will produce as much of the return value as possible starting from that state (Equation (8)).

$$\pi^* = \underset{\pi}{argmax} E_{p^\pi(h)}[R(h)] \tag{8}$$

In this equation, $E_{p^\pi(h)}$ denotes the expectation about the trajectory h extracted from $p^\pi(h)$, and $p^\pi(h)$ denotes the probability density of observing the trajectory h under policy π (Equation (9)).

$$p^\pi(h) = p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{9}$$

The procedure for computing the optimal policy π^* is not an obvious task due to the problem of sequentially defining actions that can result in delayed rewards. Because of this difficulty in determining which actions are right or wrong, it can be challenging to decide which changes have to be made to improve the suboptimal policy. Therefore, it is necessary to have efficient ways of discovering changes to the employed policy so that it is improved [40,42,46].

2.2.3. Algorithms

Approaches employing value function are a traditional way to learn optimal policy. The overall objective of value functions is to approximate the return value for all possible trajectories (h) to improve the employed policy (π). When only the state value is taken for the computation of the value function, it results in Equation (10), while if the action value is also included, Equation (11) is obtained.

$$V^\pi(s) = E_{p^\pi(h)}[R(h)|s_1 = s] \tag{10}$$

$$Q^\pi(s, a) = E_{p^\pi(h)}[R(h)|s_1 = s, a_1 = a] \tag{11}$$

Howard [34] proposed the first viable algorithm to approximate the RL problem. However, it considered $E_{p^\pi(h)}[R(h)]$ as known and was specific to low-dimensional and completely specified problems. First, it calculates the optimal value function (*policy evaluation*) and then improves the optimal policy (*policy improvement*). When the "value" of the state (s) is used for this purpose, the steps of the algorithm are in the form of Equations (12) and (13), respectively, with δ representing the Dirac delta function.

$$V^\pi(s) = E_{p^\pi(h)}[r(s, a, s') + \gamma V^\pi(s')] \tag{12}$$

$$\begin{aligned} \pi^*(a|s) &= \delta(a - a^\pi(s)) \\ a^\pi(s) &= \underset{a \in A_t}{argmax} E_{p^\pi(h)}[r(s, a, s') + \gamma V^\pi(s')] \end{aligned} \tag{13}$$

The other option is to use the value function of the state–action pair, such that the optimal policy depends on Equations (14) and (15), respectively.

$$Q^\pi(s, a) = r(s, a) + \gamma E_{\pi(a'|s')p(s'|a,s)}[Q^\pi(s', a')] \tag{14}$$

$$\pi^*(a|s) = \delta(a - \underset{a' \in A_t}{argmax}\, Q^\pi(s, a')) \tag{15}$$

Considering a model-free RL approach (i.e., when a model of $E_{p^\pi(h)}[R(h)]$ is not available), it is not easy to learn from sampled data, especially when the state and action space is continuous [1]. One alternative is to approximate the value function of the state–action pair in the form of Equation (16) (i.e., approximate RL) to improve generalization and deal with the dimensionality problem. In this equation, $\phi(s, a)$ is the base function vector and θ is the parameter to be adjusted.

$$Q^\pi(s, a, \theta) = \sum_{t=1}^{T} \phi_i(s, a)\theta_i = \phi^{\mathrm{T}}(s, a)\theta \tag{16}$$

The problem now is to estimate θ to minimize Equation (17), using Monte Carlo (MC) or temporal difference estimates for $Q^{\pi}(s,a)$ [31].

$$\theta^{*} = \underset{\theta}{argmin}[Q^{\pi}(s,a,\theta) - Q^{\pi}(s,a)] \tag{17}$$

Approximating $Q^{\pi}(s,a)$ by temporal difference is a good choice because it is a combination of Monte Carlo ideas and dynamic programming (Equation (18)). As in Monte Carlo methods, temporal difference methods can learn directly from current experiences without an environment model. As in dynamic programming, temporal difference methods update estimates based in part on other learned estimates without waiting to complete their search for the entire state and action space (i.e., using *bootstrapping*) [1].

$$Q^{\pi}(s,a) = r(s,a) + \gamma Q^{\pi}(s',a',\theta) \tag{18}$$

A policy gradient is an alternative to deal with some limitations of value-based methods, mainly when it is crucial to consider the physical system's stability criteria. An example of this is control tasks in which the agent must compute continuous actions [1]. Unlike what is seen in value-based approaches, the optimal auto-parameterized policy (θ^{*}) should maximize the expected return $J(\theta)$ (Equations (19)–(21)). However, directly maximizing $J(\theta)$ can be difficult due to the high nonlinearities for θ; thus, it is necessary to improve and adapt solutions based on methods of the gradient, so that at least a feasible local optimum is found.

$$\theta^{*} = \underset{\theta}{argmax} J(\theta) \tag{19}$$

$$J(\theta) = E_{p^{\pi}(h|\theta)}[R(h)] = \int p(h|\theta)R(h)dh \tag{20}$$

$$p(h|\theta) = p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t,a_t)\pi(a_t|s_t,\theta) \tag{21}$$

2.3. Deep Reinforcement Learning

Two achievements were essential for DL and RL theory development. First, Hinton et al. [6] demonstrated that it is possible to learn a model capable of classifying image classes without any pre-processing, mainly in the form of feature extractors to deal with the invariance dilemma [44]. Afterwards, Mnih et al. [10] developed a DRL algorithm for the Atari 2600 game platform, which in some cases surpassed the scores of human players and could address new situations distinct from those previously used for training. Specifically speaking, some general advances are listed below:

- Using elements of the mathematical theory of communication in terms of encoding information based on entropy [47];
- Updating DNNs to operate on different dynamic forms of the data distribution in space and time [25];
- Updating gradient descent (GD) and backpropagation algorithms;
- Since 2012, the DRL methodology has moved towards more stable, adaptable and robust optimization algorithms, thus avoiding obtaining time-correlated samples with a high variance which, in effect, could result in sub-optimal and divergent policies [1,48];
- Advances in parallel and distributed computing;
- Open-source software (e.g., Python code).

2.3.1. Deep Q-Learning

Approximating $E_{p^{\pi}(h)}[R(h)]$ with Q-learning depends on the number of combinations of valid actions and states required for MDP sampling. The use of tabular methods is

indicated by the limit on the number of updates of all states and actions that can be efficiently performed individually. On the other hand, with parametric approximators, the update of the value function for each state–action pair individually also influences the estimate of the value function for the other state–action pairs, thus giving it a capacity for generalization that makes the learning process more powerful, but potentially more challenging to manage and understand [1,25,46].

Within this perspective, in DNNs, updating the value function of each state–action pair is conducted following Equations (22) and (23). The TD technique is used for the value function evaluation (i.e., $TD(0)$) and defines the loss function (L_t), with backpropagation and stochastic descending-gradient algorithms to update the neural-network parameters (θ). As a result, Sutton and Barto [1] did not expect to find a value function with zero error for all states. But an approximation that balances the errors in different states.

$$L_t = \left\{ [r_t + \gamma \, max_{a'} Q(s_{t+1}, a', \theta_t)] - Q(s_t, a_t, \theta_t) \right\}^2 \tag{22}$$

$$\theta_{t+1} = \theta_t + \alpha \left\{ [r_t + \gamma \, max_{a'} Q(s_{t+1}, a', \theta_t)] - Q(s_t, a_t, \theta_t) \right\} \frac{\delta Q(s_t, a_t, \theta_t)}{\delta \theta_t} \tag{23}$$

Figure 3 presents a simplified scheme describing the procedure for training the critic network (i.e., until each episode of size T ends). It is an iterative procedure in which the sampled initial state is restarted after the completion of each episode of size T, depending on how the environment was constructed and the rewards were formulated (e.g., setting a threshold a priori for accumulated rewards, or declaring state transitions considered impossible or unfeasible) [25]. In addition, in the terminal state, the target value $\hat{y}_t = r_t$.

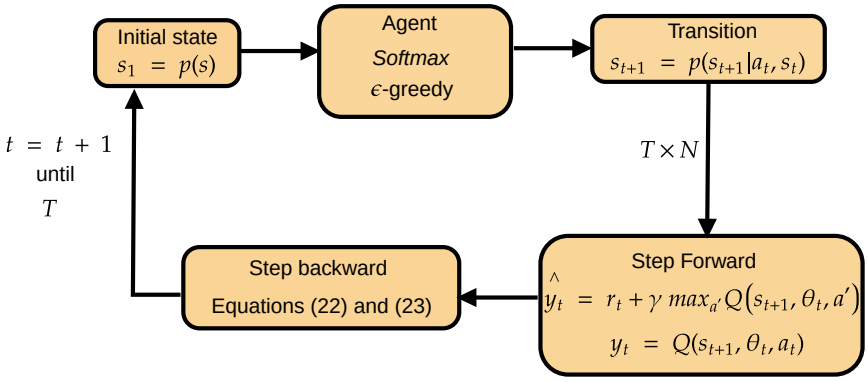

Figure 3. Simplified diagram describing the training procedure of a critic (*Q-learning*) parameterized by a DNN.

In its original form, the procedure described above may result in inadequate learning because the target $\hat{y}_t$ is never exactly approximated in the forward step. It can result in a critic with divergent policy for overestimating $\hat{y}_t$, leaving some amount of residual TD-error ($\delta_t = \hat{y}_t - y_t$). In Equation (24), the dispersion of the critic estimates depends on its complexity, variance of future rewards and the TD-error, accentuated by the value of $\gamma \sim 1$ [48–50].

$$\begin{aligned} Q(s_t, a_t, \theta_t) &= r_t + \gamma E[Q(s_{t+1}, a_{t+1}, \theta_{t+1})] - \delta_t \\ &= r_t + \gamma E[[r_{t+1} + Q(s_{t+2}, a_{t+2}, \theta_{t+2})] - \delta_{t+1}] - \delta_t \\ &= E_{p^\pi(h)} \left[\sum_{i=t}^{T} \gamma^{i-t} (r_i - \delta_i) \right] \end{aligned} \tag{24}$$

The alternatives proposed in the literature to alleviate the mentioned problems are:

- Controlling the exploratory component of the critic model used by the agent (e.g., *softmax* or ϵ-*greedy* [46])) will contribute to adequately exploring enough transitions of state, avoiding obtaining sub-optimal policies;
- Using experience replay to reduce the effect of temporal correlations between transitions uniformly sampled at random from the buffer, which allows estimating θ with important dynamic information [10];
- Updating from the target value $\hat{y}_t$ with delayed (or filtered) copies of the original DNN (i.e., $\theta_{t+1} = \kappa\theta_t + (1-\kappa)\theta_{t+1}$) [48].

2.3.2. Deep Policy Gradient

During the development of RL theory, new complementary methodologies emerged to overcome the main limitations of methods based on value function, which are a consequence of using a deterministic model to select the actions. Thus, it was difficult to obtain a stable model since small changes made in the estimation of the value function led to significant changes in the action performed by the agent. Therefore, an under-optimized and inadequate critic for exploiting states and actions was obtained [51–53].

Historically, policy-gradient methods were the first option developed to deal with such disadvantages. They are stable compared to value-based methods, as they use an auto-parameterized policy known as an actor and implement the actions without employing a critic, allowing them to approach stochastic dynamics and learn in a more stable way from the interaction between agent and environment.

Mathematically, they inherit concepts from Equations (17)–(19), in which the challenge is to carry out the optimization of $J(\theta)$ simultaneously with the exploration of more rewarding actions and states for $R(h)$. For that purpose, the most-used technique is the ascending-gradient-optimization algorithm (Equation (25)), which depends on a new approximation for Equation (19), aiming to compute the gradient with relation to the chosen parameterized model (Equation (26)) [33]. According to Sutton and Barto [1] and Grondman et al. [54], the solution of this last equation results from the policy-gradient theorem, which explains that the unknown effect of changes in $\pi(a_t|s_t,\theta)$ over the state distribution $p(h|\theta)$ does not involve its derivative (Equation (27)), so its expectation can be approximated by the empirical average in the form of Equation (28), where $h_n = [s_{1,n}, a_{1,n}, \cdots, s_{T,n}, s_{T+1,n}]$.

$$\theta_{t+1} = \theta_t + \alpha\nabla_\theta J(\theta_t) \tag{25}$$

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta p(h|\theta) R(h) dh \\ &= \int p(h|\theta) \nabla_\theta \log p(h|\theta) R(h) dh \\ &= \int p(h|\theta) \sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|s_t,\theta) R(h) dh\end{aligned} \tag{26}$$

$$\nabla_\theta J(\theta) = E_{p^\pi(h|\theta)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|s_t,\theta) R(h)\right] \tag{27}$$

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi(a_{t,n}|s_{t,n},\theta) R(h_n) \tag{28}$$

The actor categorizes actions instead of their value $\pi(a_t|s_t,\theta)$, and there is no way to compute ∇J directly using gradient-based optimization techniques. Because of this, the Monte Carlo simulation approximates the value of Equation (28) (i.e., random sampling N episodes of size T). As a result, the actor is updated by Equation (25), where $R(h_n)$ is directly proportional to $\nabla_\theta J$, and $\log \pi(a_{t,n}|s_{t,n},\theta)$ is inversely proportional to the policy adopted (i.e., it follows from the identity $\nabla \ln x = \frac{\nabla x}{x}$), as actions sampled with frequency are chosen even if they do not produce the highest expected return [1,13,25,33,54].

For example, training with the REINFORCE [42] algorithm requires the execution of a very large number of cycles, as represented in the diagram shown in Figure 4 (i.e., when $\dim(T) \sim \inf$), which is computationally unfeasible for some online applications [1,25]. Therefore, it uses a baseline to reduce variance, similar to the temporal-difference-learning methodology. For example, Williams [42] used a constant to represent it (e.g., mean reward). However, it is more appropriate to employ a state-specific option, with the value taken immediately before the actor samples the action.

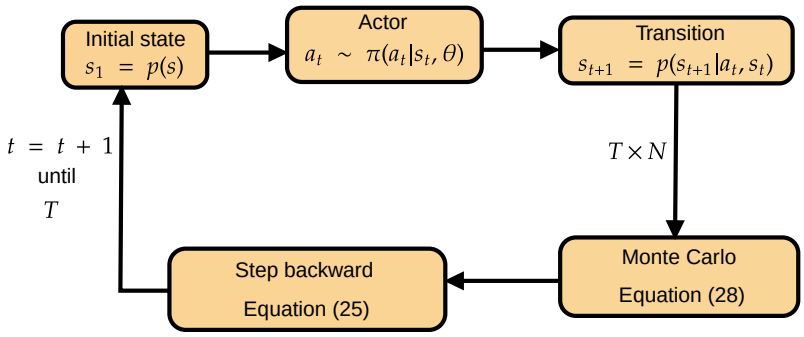

Figure 4. Simplified diagram describing the REINFORCE algorithm.

2.3.3. Deep Actor–Critic

The actor–critic methodology evolved due to the individual limitations of the methods based on the policy gradient and value function (Figure 5). Historically, it first addressed adaptive elements consisting of an associative search element and an adaptive critic element [38]. However, they were not yet structured in an actor combined with a critic. This only happened after formulating the policy-gradient theorem [51], which guaranteed convergence for both methodologies. Currently, its methodology stands out as a complete approach to RL (e.g., see algorithms developed in the works of Silver et al. [48], Fujimoto et al. [50], Ramicic and Bonarini [55] and Haarnoja et al. [56]), as they add the advantages of methods based on the value function for accelerating training and reducing variance while maintaining stability and convergence properties resulting from policy-gradient methods during training [1,54].

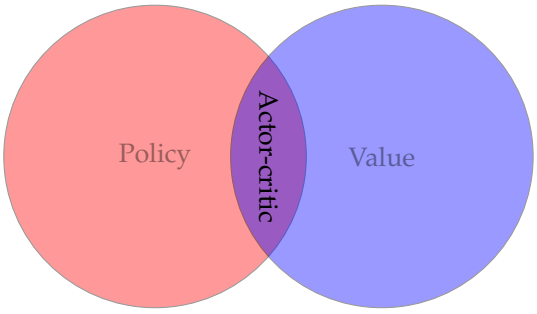

Figure 5. Venn diagram of the methods that constitute the actor–critic methodology.

Mathematically, the main difference concerning policy-gradient methodology is the replacement of the complete return (e.g., obtained with Equation (28)) by the return in one time step. According to Benhamou [57], the algorithms must remain unbiased concerning the approximation of Equation (28) and follow the principle of the control variant, where $R(h)$ is replaced by TD methods such as TD(0), Sarsa(0) and Q-learning; or by methods based on advantage, according to Equation (29).

$$A(s,a) = E_{\pi_\theta}\left[r_{t+1} + \gamma V(s_{t+1}|s_t = s, a_t = a)\right] - V(s_t) \quad (29)$$

This paradigm for DNN is shown in Figure 6, with the actor selecting the action a_t. After this step, the critic weighs the descending gradient in the algorithm for updating $\theta_a(t+1)$ (e.g., Q-AC, TD-AC, A-AC). In the backward step, Equation (30) characterizes the updating of the actor's parameters. Additionally, the critic updates the values of such actions (Equation (31)). At the end, both parameterized models must achieve good generalization so that the actor does not becomes stuck around sub-optimal solutions and the critic minimizes the residual TD-error, as seen in the form of Equation (24) [1,13,25].

$$\theta_a(t+1) = \theta_a(t) + \alpha_a Q(s_t, a_t, \theta_c) \nabla \theta_a \log(\pi(a_t|s_t, \theta_a)) \tag{30}$$

$$\begin{aligned} \delta_t &= r_t + \gamma Q(s_{t+1}, a_{t+1}, \theta_c) - Q(s_t, a_t, \theta_c) \\ \theta_c(t+1) &= \theta_c(t) + \alpha_c \delta_t \nabla \theta_c Q(s_t, a_t, \theta_c) \end{aligned} \tag{31}$$

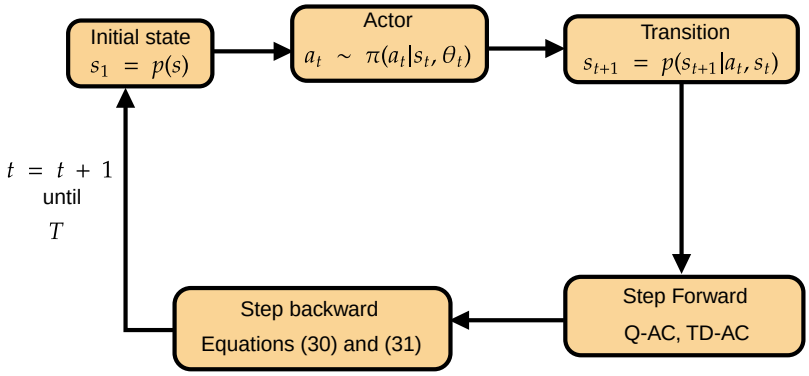

Figure 6. Simplified diagram describing the training procedure of a critic and actor parameterized by a DNN.

2.3.4. State-of-the-Art Algorithms

Dating from 2012, the created algorithms are mainly categorized as off-policy or on-policy. The former defines a learning process in which the policy update and prediction are disassociated. Such a paradigm is shown in Figure 7, where π_{k+1} is updated from samples of policy rollout data up to π_k (i.e., $\pi_1, \cdots, \pi_k$), which are contained in a large buffer. On the other hand, the latter defines a process where prediction and update are associated so that the rollout data to update the policy π_{k+1} is exclusively taken from the previous policy (i.e., π_k), meaning that it does not depend on a buffer [1,49]. Some key examples are detailed below:

- Deterministic policy gradient (DPG) (i.e., actor–critic and off-policy). According to Silver et al. [48], this is an adaptation to the policy-gradient and Q-learning algorithms, since the stochastic component intrinsic to the policy-gradient algorithms ($\pi(s_t, a_t)$) has a parameterized function in the form of $\pi(s_t, \mu_\theta(s_t))$, while depending on the computation of gradients to approximate the optimal values of a^* and $Q(s, a)^*$, which is guaranteed according to the deterministic-policy-gradient theorem, as shown in Equation (32):

$$\nabla J(\mu_\theta) = E_{p^\pi(h|\mu_\theta)}\left[\nabla_\theta \mu_\theta(s)\, \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]; \quad (32)$$

- Deep deterministic policy gradient (DDPG) (i.e., actor–critic and off-policy) [58]. This is an updated version of the DPG algorithm regarding the use of DNN, replay buffer, target networks and batch normalization, in addition to the possibility of handling the exploration problem independent of the learning algorithm used;
- Proximal policy optimization (PPO) [59]. Contrary to the algorithms above (i.e., off-policy), PPO is an algorithm that learns while interacting with the environment over different episodes (i.e., on-policy). Methodologically, this property comes from another similar algorithm considered more complex (trust region policy optimization (TRPO)), addressing the Kullback–Leibler (KL) divergence effect and surrogate objective functions;
- Soft actor–critic (SAC) (i.e., actor–critic and off-policy) [56]. This algorithm is composed of an actor and a critic, and includes a smooth value function, which is responsible for stabilizing the training of the actor and the critic. In addition, it also has similar properties to the DDPG algorithm; however, it adds an entropy value to compose the buffer.

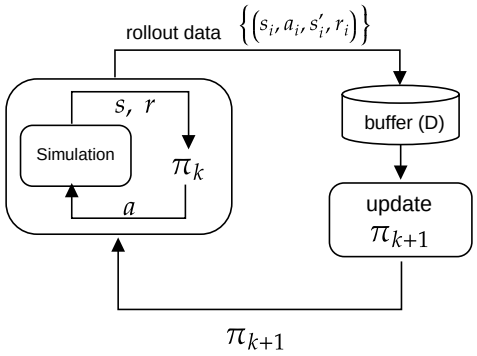

Figure 7. Off-policy reinforcement learning.

3. Reinforcement Learning for Process Control

For the definition of the MDP, it is crucial to understand the dynamics of the process rather than just focusing on the RL agent itself. This article uses batch and continuous operating regimes as case studies. The main difference between them lies in considering a single operating point in continuous operation. From a process-control point of view, for both operating regimes, it is difficult to establish a robust methodology to account for the effects of the process uncertainty, with model-based (MPC) or data-based approaches [12,13,32]. At this point, the analyzed RL paradigm comprises methodologies that rely on extensive offline training, both simulation-based and data-driven, and how to implement the control policy with transfer learning. It is illustrated in Figure 8, where Modules (1) and (2) characterize the offline training phase. In contrast, Module (3) characterizes the deployment phase of the policy obtained in the preceding phase ($\pi_S + \pi_E$). Explicitly speaking, Module (1) defines a type of training known as off-policy, demanding a reliable model from the process. The obtained policy $\pi_s = \pi_{k+1}$ is updated through samples stored in a large buffer, which is appended from samples (i.e., rollout data) taken from other policies $\pi_0, \pi_1, \cdots, \pi_k$. On the other hand, Module (2) learning depends on a fixed policy sample (π_β), which can be either known or unknown, to fill the buffer. Thus, the final policy π_E will equal the base policy π_β, characterizing a learning process similar to supervised learning. However, information from the process can be helpful to delimit the exploration and exploitation of agents trained from it.

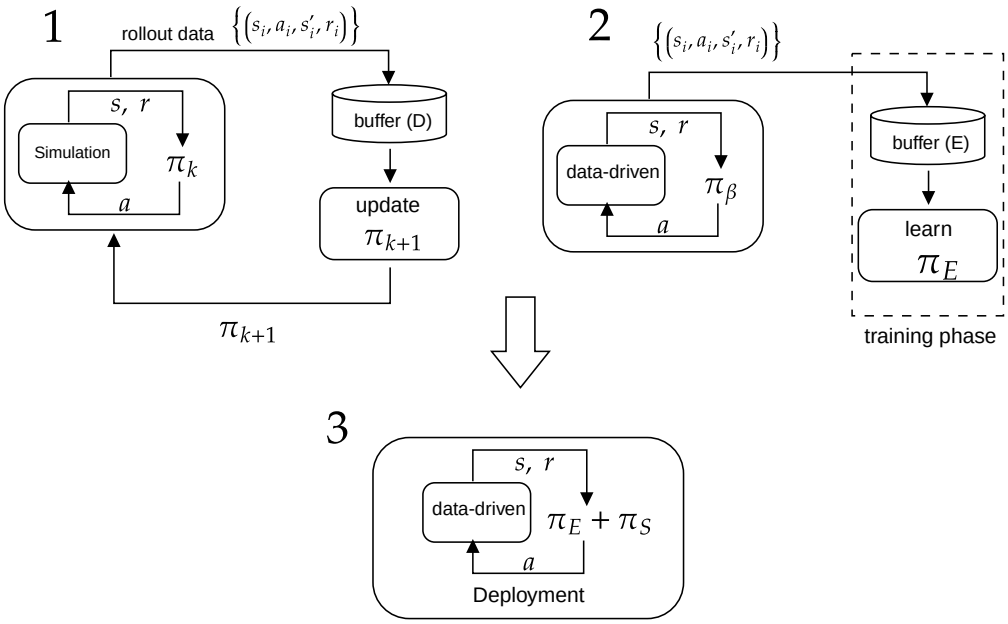

Figure 8. Diagram describing the modules for offline agent training (Modules (1) e (2)) and process-line deployment (Module (3)).

3.1. Defining Elements of RL

Whether the operating regime is continuous or batch, the RL elements compose a non-stationary MDP with an agent acting under uncertainty. According to RL theory, the state considered in the design of the MDP must be Markov, and implicitly contains all the relevant information of the past, which may not be the case for a batch or continuous process, as the prediction of the future will also depend on a few past states. For example, Yoo et al. [16] used a phase-segmentation strategy for a polymerization process, in which the MDP consists of RL elements for each phase of the process. Furthermore, they used historical information by adopting the DDPG algorithm with an MC estimator instead of TD(0). Nian et al. [12] defined the RL elements for an experiment to set-point tracking control, and they considered it crucial to modify the reward function to avoid its saturation due to the size of the time horizon (i.e., $T = 2000$). They also increased the dimension of the state–action pair to contain historical information (i.e., with a delay of one or two steps in time) and stabilized the training with the DDPG algorithm. Petsagkourakis et al. [14] addressed a process with batch-to-batch dynamics. To that end, they included historical information from the state–action pair to compose the input vector, which is fed to the REINFORCE algorithm (i.e., $s_k, s_{k-1}, a_k, a_{k-1}, t$). It is parameterized by a vanilla recurrent neural network (VRNN). It uses an MC estimator, storing historical information by the recurrence of VRNN hidden states and approximating the return considering the entire process trajectory.

The examples cited above assume that the current state of the system is observable, which is desirable to approximate the MDP without relying on historical information to predict the state transition. An extension of this is called partially observable MDP (POMDP) [60]. It is still not a critical problem for process control with RL, as there is an offline training step in which an attempt is made to reduce the interference of hidden states, adding to the acquired data experience, to obtain the control policy to be implemented in the actual process.

Under all the conditions mentioned above, there may still be incompatibilities between offline and online MDP elements, especially given the stochastic nature of the online process. An alternative is to combine RL techniques with IL and TL. The former can augment the MDP with data-driven information, thereby implicitly including factual process information about control objectives and constraints. The latter helps compose the

online MDP by transferring information from the offline MDP and then starting to learn from a condition that is at least sub-optimal.

3.2. Batch Process

Batch processes are intrinsically dynamic, as the operation point constantly changes between the start-up and the end of the batch. This operation regime is specific to units in which the processed load volume is relatively small compared to units operating in a continuous regime, in addition to being multi-objective regarding the processing steps in order to reach the desired end product. Therefore, understanding this dynamic is not simple, as there may be several challenges to adopting an adequate control structure in this context, especially considering technical problems related to [61–63]:

- Modeling: processes in batches normally exhibit nonlinear dynamics;
- Measurements: these are typically only available at the end of the batch;
- Uncertainty: present in practically all batch processes, whether resulting from reactant quality, modeling errors, process disturbances and measurement errors;
- Constraints: there is usually the added effect of terminal constraints and the existing security and operational constraints.

Optimizing a control structure can be challenging, as it depends on managing all the issues discussed above. In this context, Bonvin [61] and Arpornwichanop et al. [63] commented that the most appropriate option for dynamic optimization of such processes is based on uncertainty, measurement and model-free optimization.

Therefore, the RL methodology can be a viable option for the dynamic optimization of these processes. Specifically speaking, the control of robots by RL agents is conceptually similar to the control of batch processes. Hence, the number of applications aimed at batch process control is increasing due to the structure of the optimization problem being easily adaptable.

For example, Petsagkourakis et al. [14,15] analyzed the optimization of batch bio-processes using the REINFORCE algorithm, adapting it to handle continuous domains by using an agent with control outputs represented by a multivariate normal distribution. At the end, the control structure only includes Modules (1) and (3), with a batch-to-batch configuration, in which the policy initially obtained in (1) (i.e., in the simulated environment π_S), is updated in (3) according to each new batch made ($\pi_S^1, \cdots, \pi_S^N$). For this, the policy-transfer technique is used so that the actor approximated by DL in π_S^1 as the batch progresses results in a new agent with updated weights π_S^2 until the batches are completed, π_S^N.

Mowbray et al. [64] also analyzed batch bio-processes. They included process constraints to define the MDP (Module (1)) and proposed the entropy-regularized proximal-policy-optimization algorithm instead of the REINFORCE algorithm to stabilize training in the deployment phase (Module(3)).

Oh et al. [65] proposed an integrated algorithm of a double-deep Q-network and model predictive control. The proposed method learns the action–value function in an off-policy fashion (i.e., Module (1)) and solves the model-based optimal control problem where the terminal cost is assigned by the action–value function (Module (3)).

Yoo et al. [16] analyzed the control problem of a batch process using an RL agent with the DDPG algorithm and Monte Carlo sampling instead of TD (i.e., polymerization process). Likewise, the controller design depends only on Modules 1 and 3. The policy is obtained in the simulated environment given different process conditions, and is segmented into phases to facilitate the definition of learning elements and interaction between agent and environment. Then, the trained policy was applied online and proven efficient under new process conditions.

Although applications for batch processes with RL can be used to replace standard approaches (e.g., with MPC methodologies), the most significant difficulty for its implementation is in the proper design of the offline training phase, followed by the deployment of the obtained policy. Additionally, it is currently common to have measurements of the dynamics of the process and control actions, which enables the inclusion of Module (2)

applied in the offline training of the RL agent. This allows obtaining an RL agent with information about the actual process, including process constraints and control objectives.

3.3. Continuous Process

On the other hand, for continuous process control, reaching a steady state is essential to meet the plant optimum and evaluate the process trajectory rather than just considering the final optimization objective. Contrary to the batch process, measurements and good models are prerequisites for an effective approach to the problem, given that the operation is practically uninterrupted. Currently, the problem may be that the benefits achieved by following an optimal steady-state trajectory may not be economically the most viable path, as can be evidenced in the chemical industry's hierarchical structure for process optimization and control at the descending decision-making scale level: RTO, supervisory control, regulatory control and process. Namely, the RTO layer depends on a rigorous process model with hour-scale control decisions. When the new optimal values for other layers are not transmitted, they may operate in a sub-optimal stationary condition, causing economic losses [12,13,21,66].

Based on this, Ellis et al. [66] pointed out the advantages and disadvantages of an alternative methodology called economic model-based predictive control (EMPC), which implicitly accounts for the effect of economic cost based on a less rigorous model of the process, but with minute-by-minute control decisions, in such a way that it dispenses with the optimization layer of higher-level processes (e.g., RTO). However, they found that the path for its consolidation, similar to MPC, is still open, due to the difficulty of meeting the economic objective of optimization while addressing restrictions, uncertainties, measurement noise and process disturbances. These challenges are also problems for RL-based methodologies, as it is intrinsically related to the field of AI, in which the objective is to achieve the greatest possible return regardless of the trajectory followed.

Table 2 summarizes the main state-of-the-art developments regarding RL applied to continuous process control, where the implementation and training details of the agent (i.e., control algorithm, learning method, value function evaluation method) applied in the process are shown for each reference cited.

Table 2. Several references related to the control of continuous processes.

Author	Control Algorithm	Learning	Estimator	Process
[18]	Deep actor–critic	Off-policy (1)	TD(0)	CSTR
[67]	Deep Q-learning	Off-policy (1)	TD(0)	Conical tank systems
[68]	Deep Q-learning	Off-policy (1)	TD(0)	Liquid–liquid separation
[30]	MPC plus DRL	Online	MC	Nonlinear control-affine system
[28]	MPC plus DRL	Online	—	Nonlinear process
[69]	MADDPG	Off-policy (1)	TD(0) and TD(λ)	Waste treatment
[70]	A2C	Off-policy (1)	TD(0)	Hydrocracking
[20]	A3C	Off-policy (1)	TD(0)	Hybrid tank system

In general, all control algorithms are parameterized by DNNs for the actor and critic, as they have already proven to be efficient in handling large and complex data [10]. Furthermore, only Ramanathan et al. [67] and Hwangbo and Sin [68] have employed value-based methods, while the other authors used algorithms derived from actor–critic methods. The explanation for this is the benefit of integrating an actor to decide the control actions weighted by a critic, accelerating the learning process while reducing the variance of actions selected by the actor resulting from TD methods.

Except for Shah and Gopal [28] and Kim et al. [30], other authors cited employed off-policy learning, with the index (1) referring to training carried out in a simulated environment. The practicality of the offline training step depends on a simulated environment that is trustworthy, allowing the testing of various process conditions. Furthermore, TD methods are preferable to MC methods precisely because they combine MC and DP ideas to obtain process estimates and can be improved with eligibility traces (λ), which is a way of weighting between TD(0) "targets" and Monte-Carlo "returns".

Another important consideration is the agents used by Dogru et al. [20], Chen et al. [69] and Oh et al. [70], who developed control structures with agents learning asynchronously. The A2C and A3C algorithms are variations of the actor–critic algorithm with agents learning asynchronously, with two or three agents in parallel [71]. In multi-agent DDPG (i.e., MADDPG with two agents), the training is decentralized regarding the control actions taken by each actor, while it is centralized by only one critic to evaluate the actions taken by each actor. Thus, such approaches allow agents with competitive and cooperative control objectives to improve offline learning and facilitate deployment in the entire process, as seen in Chen et al. [69].

Powell et al. [18] innovated by proposing the first algorithm for RTO. They employed an agent with a deep actor–critic algorithm and replaced the standard descending gradient optimization algorithm with the particle swarm method, justifying it as a global optimization method.

3.4. Policy Deployment with Transfer Learning

The transfer-learning methodology is about the process of reusing and transmitting information obtained by a specific agent (e.g., animals, robots or human beings), which is trained in a particular environment (i.e., source task), in another unexplored environment (target task). According to DL theory, TL (i.e., through a DNN) relaxes the assumption that training data should be independent and identically distributed with test data, which motivates using it against the problem of insufficient training data. As a result, what would be the benefits of using this approach for RL (e.g., see Tan et al. [72] for more details)? Briefly, in principle, it could result in an improvement in the sampling efficiency of RL methodologies applied to an online problem, in which there are usually few samples available to train the agent and obtain a suitable policy [72,73].

The definition of transfer-learning methodology depends on the similarity between the analyzed tasks, in other words, the similarity between the offline MDP (i.e., source task, Γ_S) and the online MDP (i.e., target task, Γ_T). In the literature, this is called inductive transfer learning when the tasks and rewards are different (i.e., $\Gamma_S \neq \Gamma_T$ and $r_S \neq r_T$), despite having the same state and action space ($D_S = D_T$). It is called transductive transfer learning when $\Gamma_S = \Gamma_T$, $r_S = r_T$ and $D_S \neq D_T$ [74]. For both scenarios, TL must improve initial performance, learning performance, asymptotic performance and convergence [73–75].

TL methods aim to improve the learning and sampling rate in an online MDP, as it is impractical to learn while exploring the real environment in process-control applications. In Table 3, several references on TL for RL are summarized, giving a view of the techniques and leading control-oriented applications that may be promising for process control. Zhu et al. [76] analyzed the different ways the TL process can take place in an RL context. In general, they categorized the approaches by distinguishing them according to what is learned, namely: through demonstrations (learning from demonstrations (LD)); assigning external knowledge to reconstruct target domain reward distributions to guide agent learning (reward shaping (RS)); using external knowledge of pre-trained policies from one or multiple source domains (policy transfer (PT)); using inter-member transfer learning (inter-task mapping (IM)).

Table 3. Several references on TL for RL to robot control-oriented.

Author	TL Methodology	Environment
Wulfmeier et al. [7]	RS and LD	Robot control-oriented
Peng et al. [8]	PT	Robot control-oriented
Yan et al. [77]	LD with IL	Robot control-oriented
Christiano et al. [78]	LD with IRL	Robot control-oriented
Joshi and Chowdhary [75]	RS and PT	OpenAIGym
Kostrikov et al. [79]	LD with AIL	Robot control-oriented

The most employed TL methodology for the RL-based control of batch and continuous processes is the policy-transfer technique, which is similar to the approach used in Peng et al. [8] (e.g., see [16–18,80]). It depends on a reliable process approximation for extensive offline learning before application to the real process.

For example, Petsagkourakis et al. [14] implemented this TL methodology to develop an RL-based controller for a biochemical batch process. First, they obtained the optimal policy for the simulated environment. Then, they applied it online by allowing the output layer of the DNN to learn from consecutive batches while freezing the remaining layers. Mowbray et al. [27] developed a TL framework employing an inverse RL technique, which allowed them to analyze historical process data for synchronous identification of a reward function and then obtain the control policy in a step before its application in the online process. According to the work's objective, this was proven necessary, as the proposed RL agent started the learning process following the policy already obtained in the previous step.

The other forms of TL in RL (e.g., RS, LD, IM) have still not been significantly explored because they combine all forms of learning and obtaining a stable policy therefore becomes challenging. An example of these difficulties is the inclusion of the process constraints. However, this is the way forward in chemical processes in the future, as it will result in a more efficient and faster control agent to adapt to changes in the process dynamics [12,32].

3.5. Conclusions about RL for Process Control

In general, offline learning methods take advantage of both simulated and process data. Pan et al. [26] and Mowbray et al. [27] developed methodologies enabling Modules (1), (2) and (3) shown in Figure 8, which is the way forward for the design of RL agents suitable for control in online processes. Although not contemplated in this review article, there are many adaptations of these algorithms whose function is to improve the sampling efficiency of the MDP. For example, the actor approximated by DL does not have saturated neurons (i.e., inverting gradient [81]); rather, the actor selects samples with information prioritized by the critic (i.e., prioritized buffer replay [82]).

4. Challenges for the Implementation of RL to Process Control

In this section, the challenges to implementing a general framework for optimal process control are considered:

1. The MDP design based on the process dynamics;
2. The offline training step;
3. Policy transfer to the process line;
4. Keeping the policy stable against new process changes.

4.1. Overview

First, implementing a new methodology for optimal process control depends on designing an MDP based on the process dynamics. Through the sections already described in this review, the available knowledge about the process dynamics will define the stages of offline training and TL. In other words, the objective is to extract useful information from already established policies in the process and simulation data.

The focus has been on batch and continuous processes, in which the inclusion of the transfer-learning process is complex. At this point, state-of-the-art development is still being consolidated, as the inclusion of process constraints remains to be addressed. Some exceptions are the works of Pan et al. [26] and Mowbray et al. [27,64].

As long as the process states can be observed, the configuration of the offline training step is directly related to the training of the learning agent itself, namely:

1. The choice of the algorithm;
2. The exploration–exploitation trade-off dilemma;
3. Hyperparameter optimization.

The choice of algorithm depends on the complexity of the process. There is a preference for off-policy algorithms for batch and continuous processes, as they allow the storage of information from other policies. However, the exploration–exploitation trade-off dilemma analysis indicates the most appropriate policy to approximate the MDP. Being stochastic, it deals better with the uncertainties inherent in the process. Finally, hyperparameter optimization is essential to obtain a feasible algorithm and DNN.

The transferring of knowledge obtained in the previous phases is also necessary to implement the agent for optimal process control. In Section 3.4, common approaches to robot control were detailed. However, for chemical processes, a conservative approach must consider process constraints. Thus, learning from demonstrations, transferring actor and critic policies, adapting rewards (i.e., reward shaping) and observing the correspondence between offline and online environments are essential to fulfilling this purpose.

Lastly, process-line maintenance is another critical issue to consider. At this point, the control dynamics and complexity of the process indicate how often measurements are available. This allows establishing when to retrain the obtained policy. Generally, the preference is to directly apply an alternative in the online environment without performing a backup. However, the current state of the art is still embryonic. For example, Wang and Ye [83] proposed consciousness-driven reinforcement learning, which showed superior results to standard algorithms (i.e., DDPG, PPO) for simplified examples only (i.e., OpenAIGym). Hence, the current state of the art in chemical processes is exploring RL techniques that include information from demonstration to discover unknown a priori information and limit the search space of the chosen algorithm.

4.2. Proposed Learning Structure

The proposed learning structure is derived from the one shown in Figure 8. Figure 9 characterizes the modules to design the proposed learning structure. Again, Module (1) characterizes off-policy training. The simulated environment must approximate the actual process that allows for evaluating the conditions to which the control agent may be subject. The policy is continually updated from samples stored in a large buffer in this configuration. Thus, π_{k+1} is updated with transitions obtained through the implementation of policies up to π_k, which provides information about various dynamic conditions of the process and facilitates the gradient approximation for policies based on a DNN. Feise and Schaer [84] described the importance of simulation for chemical engineering, highlighting its importance in the context of Industry 4.0 and big data. In addition, they comment that it is crucial to introduce new methodologies that integrate AI with process control to add value to new businesses in the chemical industry.

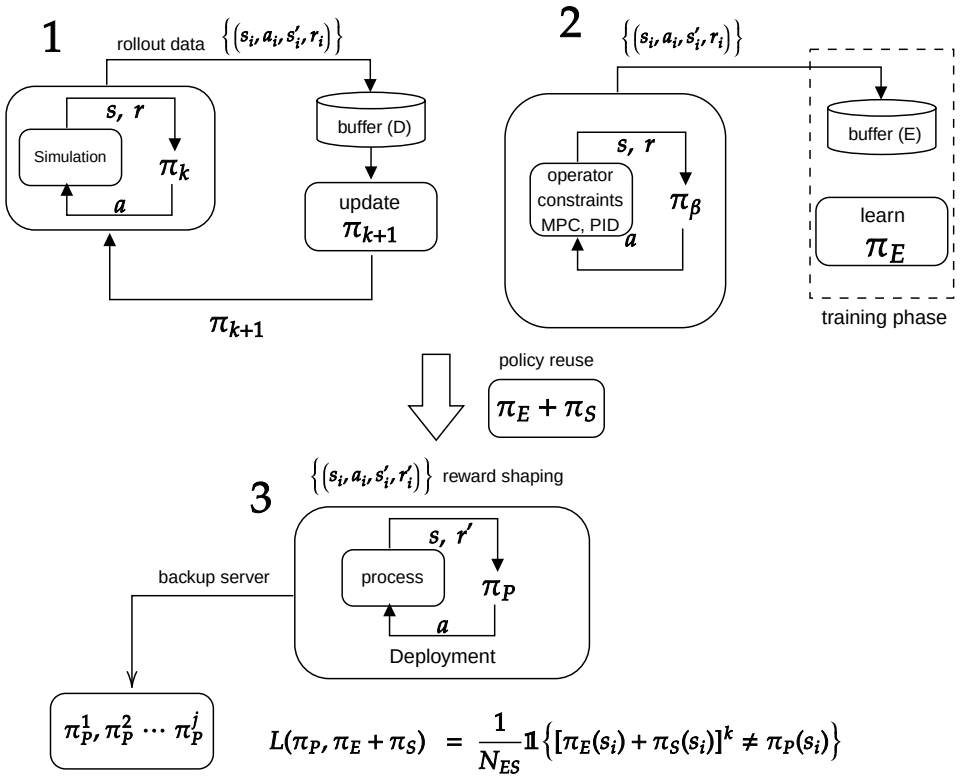

Figure 9. Diagram containing the modules for offline agent training (Modules (1) and (2)) and process-line deployment (Module (3)) adapted for chemical processes.

In Module (2), all available knowledge from demonstrations, such as sub-optimal policy information (e.g., MPC), from plant operators and from process constraints can also be included to help off-policy training. It is essential to extract useful information from the buffer E, containing implicit knowledge about control objectives and process constraints. However, the policy π_β will at most be equal to that of the expert, π_E.

Despite this, simply reusing policies previously obtained directly over the online process can result in sub-optimal policies, as the process undergoes unknown changes to its dynamics (e.g., resulting from the degradation of a catalyst). For this reason, an alternative is to choose a function for the reward regularized by the knowledge about the offline process (i.e., reward shaping) so that $\pi_P = [\pi_E + \pi_S]^{k+1}$ does not result in erroneous rewards, as well as taking advantage of the new samples of r' from the actual process.

Furthermore, for training to be effective in the online process, an ideal loss function $L(\pi_P, \pi_E + \pi_S)$ should minimize the discrepancy between π_P and $\pi_E + \pi_S$ while adding new process information. This dilemma is a problem for RL in general. Therefore, new works are investing in its solution (e.g., see [64]). Finally, a backup server could store policies updated across episodes, which then, depending on the dynamics of the process (i.e., batch or continuous), could reused again in an offline training phase, since the current policy would continue to act in the process.

5. MDP Design and Agent Training through Imitation Learning

This section highlights two points of Section 4, elucidating how learning from demonstrations can be accomplished through imitation learning and proposing a hyperparameter-optimization framework to obtain a feasible algorithm and DNN. For that purpose, a complete approach to imitation learning (IL) and reinforcement learning is detailed in a batch-process-control experiment using the deep-deterministic-policy-gradient algorithm modified with adversarial imitation learning.

5.1. Imitation Learning Techniques

Imitation learning for AI means learning from demonstrations [85]. The basic technique directly uses these demonstrations without adding experience to the learning process (i.e., direct learning). The policy trained on these principles will be equal to that used to generate demonstration samples, similar to supervised learning techniques (i.e., behavior cloning). The problem with this is generalization to unseen scenarios, which is often observed in robot control (e.g., see [8,54]). This results from an approach that does not consider different learning scenarios, thus generating sub-optimal and divergent policies.

There is currently a preference for so-called indirect-imitation-learning techniques (e.g., inverse-reinforcement learning and adversarial-imitation learning), intending to learn from experience when refining the policy obtained a priori through RL techniques and TL, among others. These techniques allows the combination of information available from the actual process and simulation, dealing with the difficulties of establishing an efficient offline training procedure, such as [86]:

- Exploration that is outside the scope of the algorithm (Module (2));
- Learning a policy that does something different from the pattern of behavior observed in the dataset E (Module (2));
- Scaling up to complex high-dimensional function approximators, such as DNN, high-dimensional state or observation spaces and temporally extended tasks.

DDPG Algorithm with Adversarial Imitation Learning

The IL approach is based on the algorithm proposed by Kostrikov et al. [79], which uses an off-policy discriminator and an off-policy actor–critic reinforcement learning algorithm. The discriminator is derived from the generative-adversary-network (GANs) theory, whose objective is to train two architectures of "adversary" neural networks, known as the generative and the discriminate [87]. Based on this, we propose here a DDPG algorithm modified by a discriminator. In Figure 10, the generator represents the DDPG algorithm learning in the simulated environment and the expert learns from the samples obtained from demonstrations. Hence, the discriminator (Ds) learns from samples of D and E until the moment it can no longer distinguish them. For this, a new reward weighs the information of the expert and the generator (r_s), as shown in Equation (33).

$$r_t = \beta r_s - (1-\beta)\log(1 - Ds(s_t, a_t)) \tag{33}$$

The generator employs a zero-mean Ornstein–Uhlenbeck process to generate temporally correlated exploration samples (i.e., η_t). In Equation (34), this is used in the actor-network μ considering time horizon T and N episodes. Subsequently, the new rollout data is stored in the buffer D (s_t, a_t, s_{t+1}, r_t). After filling the buffer of size D, random samples of size K are selected for the actor. Then, the error-TD is approximated using Equations (35) and (36) to update the critic network, with the loss function shown in Equation (37). Equation (38) defines the actor update following the deterministic policy theorem. In addition, the target networks for the critic and actor have delayed (or filtered) copies of the original network, i.e., $\theta_c(t+1) = \kappa\theta_c(t) + (1-\kappa)\theta_c(t+1)$ and $\theta_a(t+1) = \kappa\theta_a(t) + (1-\kappa)\theta_a(t)$. At the same time, the discriminator learns to discriminate between expert and generator samples by minimizing the loss function shown in Equation (39), where the expert and generator samples (i.e., $y_d = Ds(E)$ and $x_d = Ds(D)$) must be equal to one and zero (i.e., binary cross-entropy loss).

$$a_t = \mu(a_t, \theta_a) + \eta_t \tag{34}$$

$$y_t = r_t + \gamma Q(s_t, \mu(s_{t+1}, \theta'_a), \theta_c) \tag{35}$$

$$\delta_t = y_t - Q(s_t, a_t, \theta_c) \tag{36}$$

$$L_c = \frac{1}{K}\sum \delta_t^2 \tag{37}$$

$$\nabla J(\mu_\theta) = \frac{1}{K}\sum \left[\nabla_\theta \mu(s_t, \mu(s_{t+1}, \theta'_a)\, \nabla_a Q(s_t, a_t, \theta_a)|_{a=\mu(s_t,\theta_a)}\right] \tag{38}$$

$$L_d = \frac{1}{2K}\sum y_d \log(x_d) + (1 - y_d)\log(1 - x_d) \tag{39}$$

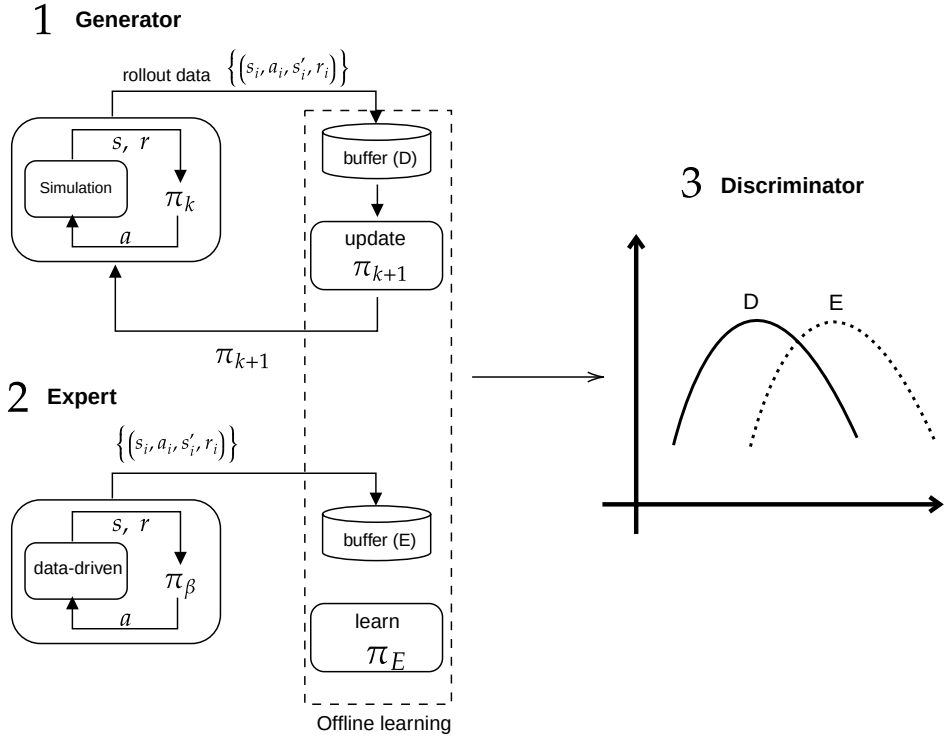

Figure 10. Diagram describing the modules for offline training of a generator and expert with a discriminator.

5.2. Hyperparameter Optimization

In general, hyperparameter optimization is recommended for complex data-driven algorithms. At the beginning, it was directed to problems with parameters characterized exclusively numerically and of low dimension. This approach became unfeasible when applied to algorithms depending on the adjustment of a large number of parameters, which may be, in addition to numeric, ordinal, and categorical, possibly dependent on each other (e.g., the CPLEX optimizer is based on a mixed-integer programming algorithm and has 80 hyperparameters to be automatically configured) [88,89]. To overcome this, Hutter et al. [88] proposed the creation of an automatic algorithm configuration structure, known as ParamILS (Hutter [89] considers this algorithm the first viable alternative to configure hyperparameters efficiently, automatically and robustly), applying it to adjust the CPLEX algorithm and return the best set of hyperparameters to meet the objective sought by the user. At that point, the value of the cost function must be efficient concerning the computational time required to complete the task, and robust so that it does not deviate significantly from the best possible solution.

Since then, this theory has been adapted for DL to optimize its complex structures (e.g., DNN), which are deep and whose training also depends on other algorithms (i.e., backpropagation and descending stochastic gradient). At this point, Coates and Ng [90] and Coates et al. [91] were the first to realize how important it can be to focus on the DL structure itself rather than just improving the learning algorithms. They tested several configurations of their main parameters, concluding that previously unfeasible alternatives can also reach results comparable to other more tested methodologies.

5.2.1. Hyperparameter Optimization Software

A good hyperparameter-optimization software framework must improve the steps seen in Figure 11. First, it is necessary to define the limit for the domain containing the hyperparameters and the starting values to carry out the other steps. This is done by categorizing all selected hyperparameters as integer, real (i.e., *float*), ordinal, or fixed value. Then, the software in charge of this will run some trials, resulting in different optimization scenarios, which are evaluated on different test samples (e.g., offline training) and returned as a solution of the employed cost function. In the end, this makes it possible to update the information about the values of the hyperparameters, directing them to produce better future optimization scenarios. A promising example is Optuna [92] due to the following features:

- Simplicity and practicality for writing the source code, an improvement over Nevergrad [92,93], Ray tune [94] and Hyperopt [95]);
- Open-source software (i.e., Python).

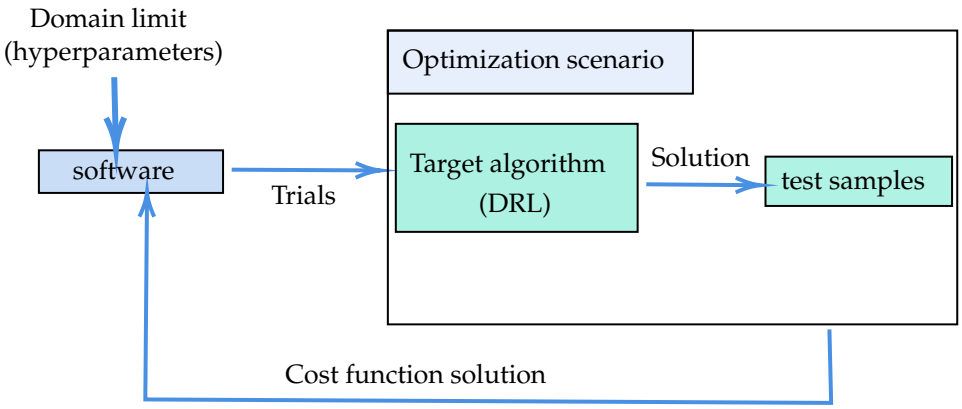

Figure 11. Simplified hyperparameter-optimization scheme.

Besides, Optuna has two optimization approaches:

- Sequential optimization relies on multiple runs of such a cycle (trials) to obtain credible information about how the parameters influence the value of the cost function. The Bayesian-optimization algorithms use an approximate model for the unknown cost function a priori, being Gaussian processes that maximize expectancy of the cost function concerning the current best solution (e.g., the expected improvement algorithms, such as the tree-structured Parzen estimator (TPE)) [95,96];
- Parallel optimization algorithms are complementary alternatives to sequential algorithms, as they update the information of hyperparameter values based on distributed executions, in contrast to the single-cycle approach used in sequential optimization.

The best of both approaches are being exploited to guarantee robust results (i.e., resulting from the plausibility of Bayes models to explore hyperparameters) and are computationally efficient (i.e., a result of distributed optimization). Successful examples of this are algorithms known as ASHA (asynchronous successive halving algorithm) [97], HYPERBAND [98] and those in which training is based on populations [99].

5.3. An Offline RL Control Experiment

For the elaboration of the offline RL control experiment, the control of batch process using DDPG algorithm with adversarial imitation learning will be investigated. The characteristics of the algorithm are detailed below:

- The algorithm is specific to MDP where the sampled state–action pairs are continuous;
- The simplicity of the algorithm, which makes writing the source code easier, the proposed updates to the algorithm (e.g., prioritized buffer replay, inverting gradient) and distributed optimization;
- There are applications for optimal process control (e.g., for DDPG, see Ma et al. [17] and Spielberg et al. [80]);
- The algorithm combines RL and adversarial imitation learning to learn from demonstrations.

5.3.1. Case Study: Batch Process

The case study is a simplified version of a batch process (Petsagkourakis et al. [14]). Following to Figure 10, the expert uses sub-optimal demonstrations from Equations (40) and (41) to fill the buffer E, representing the actual process. At the same time, the generator depends on a simulation based on a simplified version of the actual process to fill the buffer D (Equations (42) and (43)).

$$\frac{ds_1}{dt} = -(a_1 + 0.5a_1^2)s_1 + 0.5\frac{a_2 s_2}{s_1 + s_2} \tag{40}$$

$$\frac{ds_2}{dt} = a_1 s_1 - 0.7 a_2 s_1 \tag{41}$$

$$\frac{ds_1}{dt} = -(a_1 + 0.5a_1^2)s_1 + a_2 \tag{42}$$

$$\frac{ds_2}{dt} = a_1 s_1 - a_2 s_1 \tag{43}$$

The optimization objective is to maximize $E[s_2(T)]$ given the limits of the control actions (i.e., $a_1 \in (0, 5)$ and $a_2 \in (0, 5)$) for batches of ten time intervals, with the reward for each time interval taken from the simulation and discriminated with the expert reward, as shown in Equation (44). In this equation, a standard DDPG is derived from $\beta = 1$ and an indirect-imitation-learning approach from $\beta = 0$. In addition, the initial condition is fixed at the $s_1 = 1$ and $s_2 = 0$ ($s_0 = (1, 0)$).

$$r = \beta s_2(t) - (1 - \beta)\log(1 - Ds(s, a)) \tag{44}$$

5.3.2. Tree-Structured Parzen Estimator

As described in Section 5.2.1, the Optuna software was chosen to perform hyperparameter optimization applied to the control experiment. Thus, it remains to determine the optimization algorithm. Specifically, the sequential optimization algorithm (TPE) is suitable to serve this purpose [92,95,100]:

1. It is computationally efficient compared to standard approaches;
2. Hyperparameters can belong to any set: integer, ordinal, real (i.e., float) and categorical.
3. It optimizes the expected improvement with a Gaussian process and Parzen estimator;
4. The default algorithm implemented in Optuna offers an alternative to overcome its main disadvantage, that is, it also models the interactions between hyperparameters, which improves the efficiency and robustness of the algorithm.

5.3.3. Validation of the Control Experiment

For validation, the policies obtained through the different trials were tested on the actual process, and the objective function defined by the amount of s_2 obtained at the end of the batch was used for comparison (Figure 12).

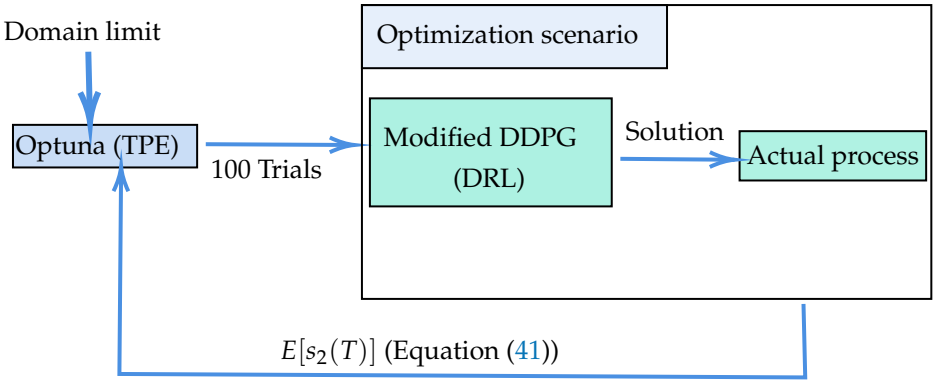

Figure 12. Optimization scheme with Optuna to validate the online control experiment with domain limit according to Table 4.

Table 4 summarizes the essential hyperparameters for optimization with the TPE algorithm and the search space for the domain containing the hyperparameters. β is the main hyperparameter to validate the proposed algorithm against the standard DDPG algorithm [58] and sub-optimal demonstrations from the actual process (E), which were obtained with REINFORCE algorithm [14]. In addition, the hyperparameters N, D, the activation function and the optimizer (i.e., Adam [101]) do not influence the optimization of $E[s_2(T)]$. Then, a limit value considered sufficient for the control experiment was defined.

Table 4. Hyperparameters for modified DDPG algorithm.

Hyperparameters	Search Space
RL	
γ	(0.80, 0.99)
β	(0, 1)
Batch size (K)	(10, 150)
Buffer (D)	500
Expert buffer (E)	500
Episodes (N)	2000
κ	(0.005, 0.01)
Actor Network	
Activation function	ReLU, Tanh
Layers (La_a)	(1, 5)
Neurons (Na_i)	(4, 250)
Critic Network	
Activation function	ReLU, Tanh
Layers (La_c)	(1, 5)
Neurons (Nc_i)	(4, 250)
Discriminator	
Activation function	Linear, Sigmoid
Layers (La_{ds})	(1, 5)
Neurons (Nds_i)	(4, 250)
NN training algorithm	
Optimizer	Adam
Actor learning rate (α_a)	$(1 \times e^{-5}, 1 \times e^{-2})$
Critic learning rate (α_c)	$(1 \times e^{-5}, 1 \times e^{-2})$
Discriminator learning rate (α_{ds})	$(1 \times e^{-5}, 1 \times e^{-2})$

When running the online control experiment according to the algorithmic implementation details described above, Figure 13 shows that the TPE algorithm took 18 trials to obtain policies with an objective function value comparable to that obtained by Petsagkourakis et al. [15] (i.e., 0.583) against 0.575 e 0.55 with the DDPG algorithm and

MPC, respectively. The TPE algorithm obtained some erroneous policies until trial 33, where the best value was obtained (i.e., 0.595). At the end, the TPE algorithm showed dispersion of the objective function values precisely because it was not able to find a solution that exceeded the best value, as detailed in Bergstra et al. [95].

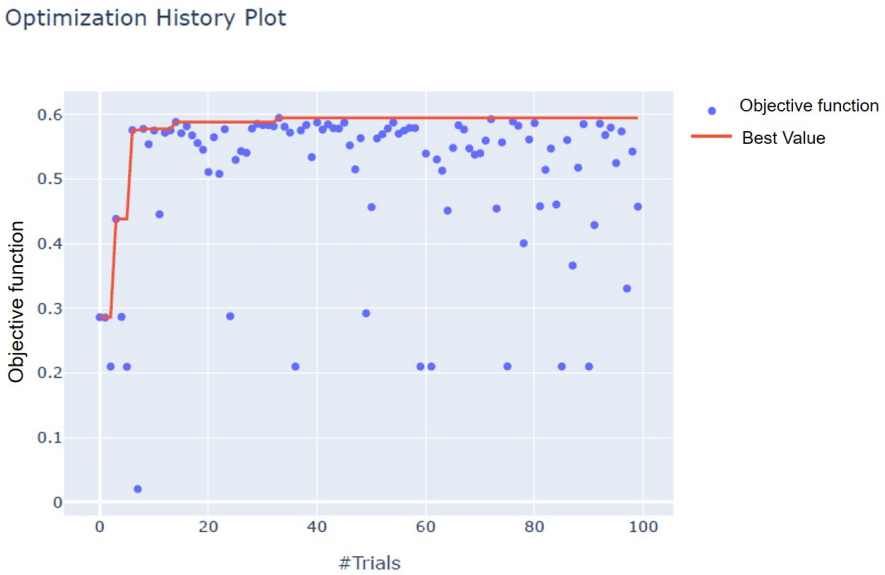

Figure 13. Optimization history plot for modified DDPG considering 100 trials with the TPE algorithm.

Figure 14 shows the hyperparameters that had more influence on the objective function. The highlight here is the hyperparameter β, which weighs the importance of the discriminator for the objective value; according to [79] it is crucial to properly know the dynamics of the environment to specify this hyperparameter correctly. This can be seen in Figure 15, where values of β important to the objective function are dispersed around its optimal value (i.e., $\beta = 0.8$), indicating that including demonstrations in the process of offline learning results in appropriate online control policies.

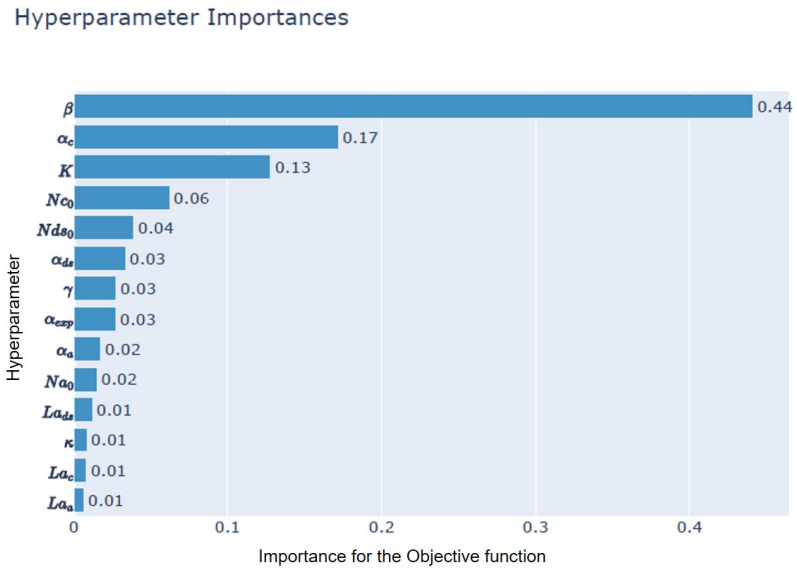

Figure 14. Hyperparameter importance plot for modified DDPG considering 100 trials with the TPE algorithm.

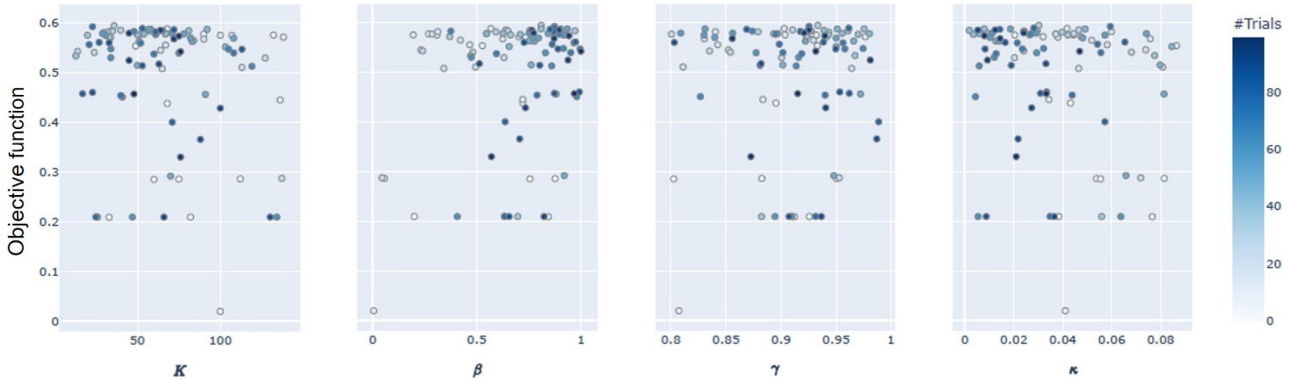

Figure 15. Hyperparameter importance plot considering 100 trials of the TPE algorithm for data distribution and sampling.

Regarding the structures of the actor, critic and discriminator networks regarding the number of layers and neurons and the learning rate, the critic network has the most significant contribution to the objective function (e.g., see α_c and Nc_0). More details can be found in Figure 16, where a shallow architecture (i.e., $La_c = 2$) is sufficient to approximate the value function, which depends on a significant number of neurons only in the first layer for local approximation of the input features (i.e., state–action pair), as explained in Das et al. [102]. Furthermore, training with the gradient-descent algorithm is less complex compared to the actor and discriminator. The former depends on a deeper architecture in general (i.e., $L_a = 4$) and training with a conservative GD optimization algorithm due to its greater complexity and nonlinearity to approximate the deterministic gradient (Figure 17). The latter also depends on a deep architecture and a conservative GD optimization (Figure 18), which is due to the complexity of the objective function resulting from generative and discriminative networks (i.e., by employing binary cross-entropy loss) [87].

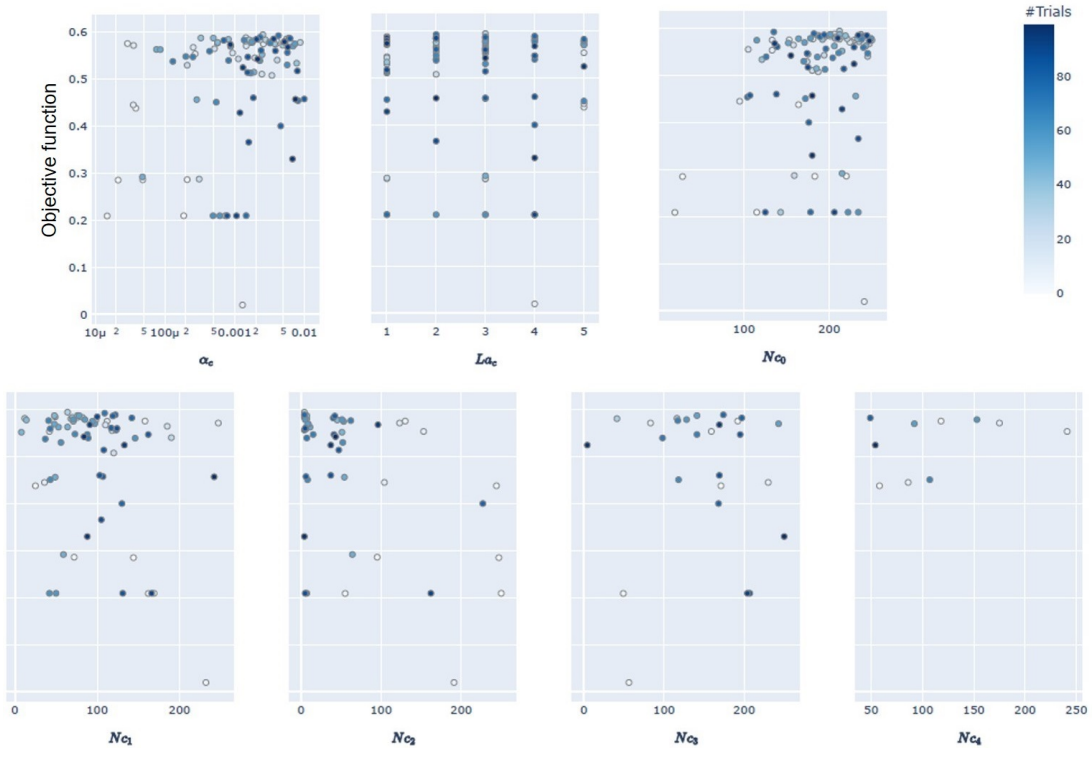

Figure 16. Hyperparameter importance slice plot for modified DDPG considering 100 trials of the TPE algorithm for the critic network.

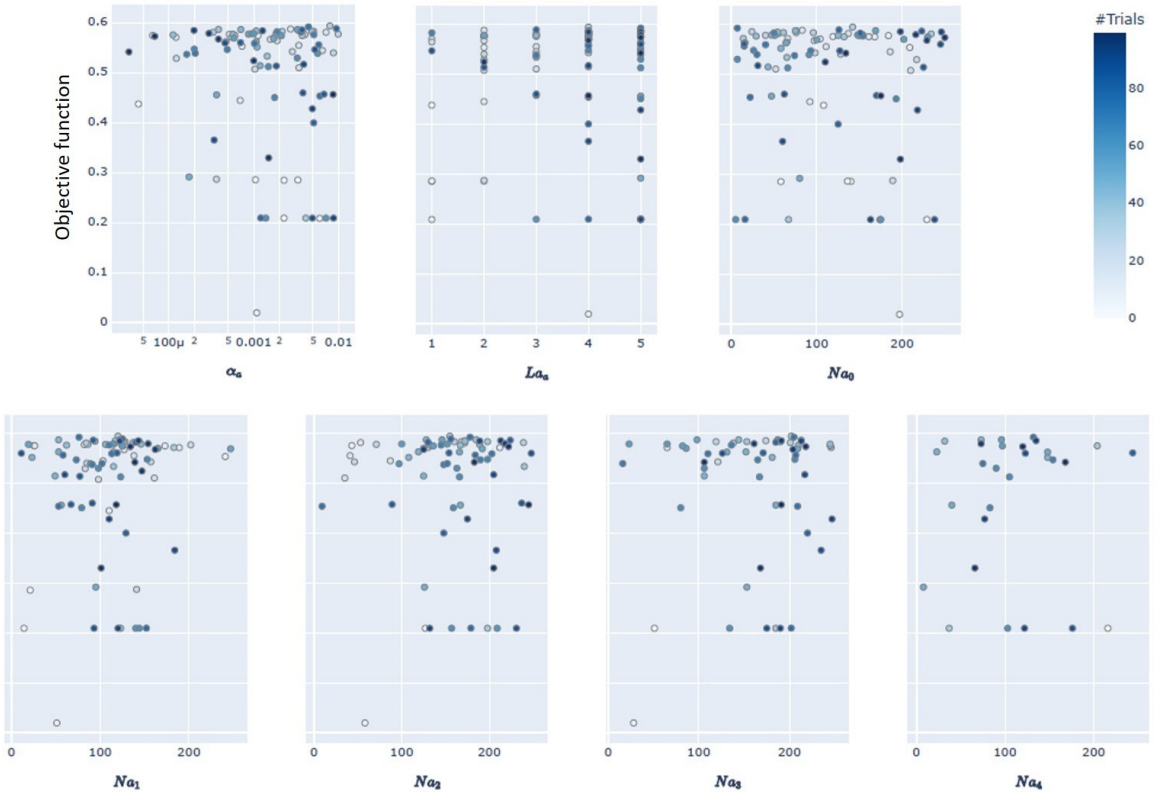

Figure 17. Hyperparameter importance slice plot for modified DDPG considering 100 trials of the TPE algorithm for the actor network.

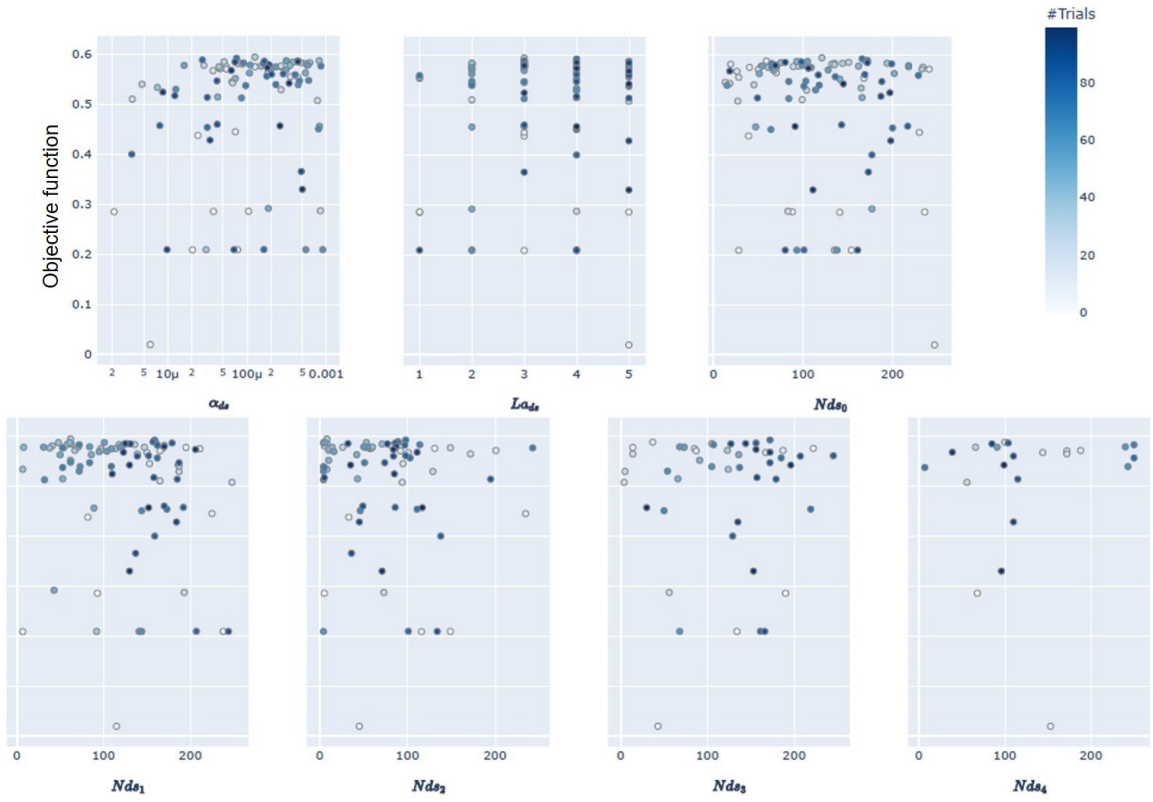

Figure 18. Hyperparameter importance slice plot for modified DDPG considering 100 trials of the TPE algorithm for the discriminative network.

6. Conclusions

These final considerations result from an in-depth state-of-the-art review of artificial intelligence and optimal process control. The objective was to develop a complete guide for hyperparameter optimization, imitation learning and transfer learning, since there are no articles covering such subjects together. Thus, the conclusions made answer several claims about the integration of reinforcement learning and process control:

- State-of-the-art technologies are still embryonic;
- Batch and continuous processes require different learning structures;
- Developing state-of-the-art offline training technologies is essential;
- Transfer learning has a broad meaning in RL, since it can encompass learning from demonstration, reward shaping, policy transfer and inter-task mapping;
- The proposed modified DDPG algorithm with an off-policy discriminator confirmed the hypothesis that information from process demonstrations improves the performance of the standard DDPG algorithm, as detailed in Section 5.

Finally, the control experiment carried out in Section 5 provides some guidelines for extending this approach to more complex systems (batch and continuous). These results can support the study of hyperparameter optimization for other case studies. Furthermore, the main challenge is developing new control structures appropriate to process constraints. To achieve this, it remains necessary to improve Modules 1, 2 and 3, shown in Figure 9.

Author Contributions: R.d.R.F. participated in all steps of the research method: conceptualization, methodology, writing—original draft preparation. For review and editing, all authors participated. Conceptualization and supervision, B.D.O.C., A.R.S. and M.B.d.S.J. All authors have read and agreed to the published version of the manuscript.

Funding: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001. Maurício B. de Souza Jr. is grateful for financial support from CNPq (Grant No. 311153/2021-6) and Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) (Grant No. E-26/201.148/2022).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
2. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957; Volume 95.
3. Bellman, R. A Markovian decision process. *J. Math. Mech.* **1957**, *6*, 679–684. [CrossRef]
4. Hoskins, J.; Himmelblau, D. Process control via artificial neural networks and reinforcement learning. *Comput. Chem. Eng.* **1992**, *16*, 241–251. [CrossRef]
5. Hinton, G.; Srivastava, N.; Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited On* **2012**, *14*, 2.
6. Hinton, G.E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R.R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv* **2012**, arXiv:1207.0580.
7. Wulfmeier, M.; Posner, I.; Abbeel, P. Mutual alignment transfer learning. In Proceedings of the Conference on Robot Learning (PMLR), Mountain View, CA, USA, 13–15 November 2017; pp. 281–290.
8. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 3803–3810.
9. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of go without human knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef]
10. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; 38 Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
11. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef]
12. Nian, R.; Liu, J.; Huang, B. A review on reinforcement learning: Introduction and applications in industrial process control. *Comput. Chem. Eng.* **2020**, *139*, 106886. [CrossRef]

13. Buşoniu, L.; de Bruin, T.; Tolić, D.; Kober, J.; Palunko, I. Reinforcement learning for control: Performance, stability, and deep approximators. *Annu. Rev. Control* **2018**, *46*, 8–28. [CrossRef]
14. Petsagkourakis, P.; Sandoval, I.O.; Bradford, E.; Zhang, D.; del Rio-Chanona, E.A. Reinforcement learning for batch bioprocess optimization. *Comput. Chem. Eng.* **2020**, *133*, 106649. [CrossRef]
15. Petsagkourakis, P.; Sandoval, I.O.; Bradford, E.; Zhang, D.; del Rio-Chanona, E.A. Reinforcement learning for batch-to-batch bioprocess optimisation. In *Computer Aided Chemical Engineering*; Elsevier: Amsterdam, The Netherlands, 2019; Volume 46, pp. 919–924.
16. Yoo, H.; Kim, B.; Kim, J.W.; Lee, J.H. Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Comput. Chem. Eng.* **2021**, *144*, 107133. [CrossRef]
17. Ma, Y.; Zhu, W.; Benton, M.G.; Romagnoli, J. Continuous control of a polymerization system with deep reinforcement learning. *J. Process Control* **2019**, *75*, 40–47. [CrossRef]
18. Powell, K.M.; Machalek, D.; Quah, T. Real-time optimization using reinforcement learning. *Comput. Chem. Eng.* **2020**, *143*, 107077. [CrossRef]
19. Nikita, S.; Tiwari, A.; Sonawat, D.; Kodamana, H.; Rathore, A.S. Reinforcement learning based optimization of process chromatography for continuous processing of biopharmaceuticals. *Chem. Eng. Sci.* **2021**, *230*, 116171. [CrossRef]
20. Dogru, O.; Wieczorek, N.; Velswamy, K.; Ibrahim, F.; Huang, B. Online reinforcement learning for a continuous space system with experimental validation. *J. Process Control* **2021**, *104*, 86–100. [CrossRef]
21. Ławryńczuk, M.; Marusak, P.M.; Tatjewski, P. Cooperation of model predictive control with steady-state economic optimisation. *Control Cybern.* **2008**, *37*, 133–158.
22. Skogestad, S. Control structure design for complete chemical plants. *Comput. Chem. Eng.* **2004**, *28*, 219–234. [CrossRef]
23. Backx, T.; Bosgra, O.; Marquardt, W. Integration of model predictive control and optimization of processes: Enabling technology for market driven process operation. *IFAC Proc. Vol.* **2000**, *33*, 249–260. [CrossRef]
24. Adetola, V.; Guay, M. Integration of real-time optimization and model predictive control. *J. Process Control* **2010**, *20*, 125–133. [CrossRef]
25. Aggarwal, C.C. *Neural Networks and Deep Learning*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10, pp. 978–983.
26. Pan, E.; Petsagkourakis, P.; Mowbray, M.; Zhang, D.; del Rio-Chanona, E.A. Constrained model-free reinforcement learning for process optimization. *Comput. Chem. Eng.* **2021**, *154*, 107462. [CrossRef]
27. Mowbray, M.; Smith, R.; Del Rio-Chanona, E.A.; Zhang, D. Using process data to generate an optimal control policy via apprenticeship and reinforcement learning. *AIChE J.* **2021**, *67*, e17306. [CrossRef]
28. Shah, H.; Gopal, M. Model-free predictive control of nonlinear processes based on reinforcement learning. *IFAC-PapersOnLine* **2016**, *49*, 89–94. [CrossRef]
29. Alhazmi, K.; Albalawi, F.; Sarathy, S.M. A reinforcement learning-based economic model predictive control framework for autonomous operation of chemical reactors. *Chem. Eng. J.* **2022**, *428*, 130993. [CrossRef]
30. Kim, J.W.; Park, B.J.; Yoo, H.; Oh, T.H.; Lee, J.H.; Lee, J.M. A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system. *J. Process Control* **2020**, *87*, 166–178. [CrossRef]
31. Badgwell, T.A.; Lee, J.H.; Liu, K.H. Reinforcement learning–overview of recent progress and implications for process control. In *Computer Aided Chemical Engineering*; Elsevier: Amsterdam, The Netherlands, 2018; Volume 44, pp. 71–85.
32. Görges, D. Relations between model predictive control and reinforcement learning. *IFAC-PapersOnLine* **2017**, *50*, 4920–4928. [CrossRef]
33. Sugiyama, M. *Statistical Reinforcement Learning: Modern Machine Learning Approaches*; CRC Press: Boca Raton, FL, USA, 2015.
34. Howard, R.A. *Dynamic Programming and Markov Processes*; MITPL: Cambridge, MA, USA, 1960.
35. Thorndike, E.L. Animal intelligence: An experimental study of the associative processes in animals. *Psychol. Rev. Monogr. Suppl.* **1898**, *2*, 1.
36. Minsky, M. Neural Nets and the Brain-Model Problem. Doctoral Dissertation, Princeton University, Princeton, NJ, USA, 1954; *Unpublished*.
37. Minsky, M. Steps toward artificial intelligence. *Proc. IRE* **1961**, *49*, 8–30. [CrossRef]
38. Barto, A.G.; Sutton, R.S.; Anderson, C.W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* **1983**, *SMC-13*, 834–846. [CrossRef]
39. Sutton, R.S. Learning to predict by the methods of temporal differences. *Mach. Learn.* **1988**, *3*, 9–44. [CrossRef]
40. Watkins, C.J.C.H. *Learning from Delayed Rewards*; University of Cambridge: Cambridge, UK, 1989.
41. Gullapalli, V. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Netw.* **1990**, *3*, 671–692. [CrossRef]
42. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
43. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006.
44. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
45. Berry, D.A.; Fristedt, B. *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*; Chapman and Hall: London, UK, 1985; Volume 5, pp. 71–87.
46. Sutton, R.S.; Barto, A.G. *Introduction to Reinforcement Learning*; MIT Press Cambridge: Cambridge, MA, USA, 1998; Volume 135.

47. Shannon, C.E. A mathematical theory of communication. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **2001**, *5*, 3–55. [CrossRef]
48. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the International Conference on Machine Learning (PMLR), Bejing, China, 22–24 June 2014.
49. Thrun, S.; Schwartz, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*; Lawrence Erlbaum: Hillsdale, NJ, USA, 1993.
50. Fujimoto, S.; Van Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. *arXiv* **2018**, arXiv:1802.09477.
51. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 29 November–4 December 2000; pp. 1057–1063.
52. Gordon, G.J. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*; Elsevier: Amsterdam, The Netherlands, 1995; pp. 261–268.
53. Tsitsiklis, J.N.; Van Roy, B. Feature-based methods for large scale dynamic programming. *Mach. Learn.* **1996**, *22*, 59–94. [CrossRef]
54. Grondman, I.; Busoniu, L.; Lopes, G.A.; Babuska, R. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **2012**, *42*, 1291–1307. [CrossRef]
55. Ramicic, M.; Bonarini, A. Augmented Replay Memory in Reinforcement Learning With Continuous Control. *arXiv* **2019**, arXiv:1912.12719.
56. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International Conference on Machine Learning (PMLR), Stockholm, Sweden, 10–15 July 2018; pp. 1861–1870.
57. Benhamou, E. Variance Reduction in Actor Critic Methods (ACM). *arXiv* **2019**, arXiv:1907.09765.
58. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
59. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
60. Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. Planning and acting in partially observable stochastic domains. *Artif. Intell.* **1998**, *101*, 99–134. [CrossRef]
61. Bonvin, D. Optimal operation of batch reactors—A personal view. *J. Process Control* **1998**, *8*, 355–368. [CrossRef]
62. Bonvin, D.; Srinivasan, B.; Ruppen, D. *Dynamic Optimization in the Batch Chemical Industry*; Technical Report; NTNU: Trondheim, Norway, 2001.
63. Arpornwichanop, A.; Kittisupakorn, P.; Mujtaba, I. On-line dynamic optimization and control strategy for improving the performance of batch reactors. *Chem. Eng. Process. Process. Intensif.* **2005**, *44*, 101–114. [CrossRef]
64. Mowbray, M.; Petsagkourakis, P.; Chanona, E.A.d.R.; Smith, R.; Zhang, D. Safe Chance Constrained Reinforcement Learning for Batch Process Control. *arXiv* **2021**, arXiv:2104.11706.
65. Oh, T.H.; Park, H.M.; Kim, J.W.; Lee, J.M. Integration of reinforcement learning and model predictive control to optimize semi-batch bioreactor. *AIChE J.* **2022**, *68*, e17658. [CrossRef]
66. Ellis, M.; Durand, H.; Christofides, P.D. A tutorial review of economic model predictive control methods. *J. Process Control* **2014**, *24*, 1156–1178. [CrossRef]
67. Ramanathan, P.; Mangla, K.K.; Satpathy, S. Smart controller for conical tank system using reinforcement learning algorithm. *Measurement* **2018**, *116*, 422–428. [CrossRef]
68. Hwangbo, S.; Sin, G. Design of control framework based on deep reinforcement learning and Monte-Carlo sampling in downstream separation. *Comput. Chem. Eng.* **2020**, *140*, 106910. [CrossRef]
69. Chen, K.; Wang, H.; Valverde-Pérez, B.; Zhai, S.; Vezzaro, L.; Wang, A. Optimal control towards sustainable wastewater treatment plants based on multi-agent reinforcement learning. *Chemosphere* **2021**, *279*, 130498. [CrossRef]
70. Oh, D.H.; Adams, D.; Vo, N.D.; Gbadago, D.Q.; Lee, C.H.; Oh, M. Actor-critic reinforcement learning to estimate the optimal operating conditions of the hydrocracking process. *Comput. Chem. Eng.* **2021**, *149*, 107280. [CrossRef]
71. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
72. Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; Liu, C. A survey on deep transfer learning. In Proceedings of the International Conference on Artificial Neural Networks, Rhodes, Greece, 4–7 October 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 270–279.
73. Taylor, M.E.; Stone, P. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.* **2009**, *10*, 1633–1685.
74. Peirelinck, T.; Kazmi, H.; Mbuwir, B.V.; Hermans, C.; Spiessens, F.; Suykens, J.; Deconinck, G. Transfer learning in demand response: A review of algorithms for data-efficient modelling and control. *Energy AI* **2022**, *7*, 100126. [CrossRef]
75. Joshi, G.; Chowdhary, G. Cross-domain transfer in reinforcement learning using target apprentice. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 7525–7532.
76. Zhu, Z.; Lin, K.; Dai, B.; Zhou, J. Learning sparse rewarded tasks from sub-optimal demonstrations. *arXiv* **2020**, arXiv:2004.00530.

77. Yan, M.; Frosio, I.; Tyree, S.; Kautz, J. Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control. *arXiv* **2017**, arXiv:1712.03303.
78. Christiano, P.; Shah, Z.; Mordatch, I.; Schneider, J.; Blackwell, T.; Tobin, J.; Abbeel, P.; Zaremba, W. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv* **2016**, arXiv:1610.03518.
79. Kostrikov, I.; Agrawal, K.K.; Dwibedi, D.; Levine, S.; Tompson, J. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arXiv* **2018**, arXiv:1809.02925.
80. Spielberg, S.; Tulsyan, A.; Lawrence, N.P.; Loewen, P.D.; Bhushan Gopaluni, R. Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE J.* **2019**, *65*, e16689. [CrossRef]
81. Hausknecht, M.; Stone, P. Deep reinforcement learning in parameterized action space. *arXiv* **2015**, arXiv:1511.04143.
82. Hou, Y.; Liu, L.; Wei, Q.; Xu, X.; Chen, C. A novel ddpg method with prioritized experience replay. In Proceedings of the 2017 IEEE international conference on systems, man, and cybernetics (SMC), Banff, AB, Canada, 5–8 October 2017; pp. 316–321.
83. Wang, X.; Ye, X. Consciousness-driven reinforcement learning: An online learning control framework. *Int. J. Intell. Syst.* **2022**, *37*, 770–798. [CrossRef]
84. Feise, H.J.; Schaer, E. Mastering digitized chemical engineering. *Educ. Chem. Eng.* **2021**, *34*, 78–86. [CrossRef]
85. Hua, J.; Zeng, L.; Li, G.; Ju, Z. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors* **2021**, *21*, 1278. [CrossRef]
86. Hussein, A.; Gaber, M.M.; Elyan, E.; Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 1–35. [CrossRef]
87. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* **2014**, *27*.
88. Hutter, F.; Hoos, H.H.; Leyton-Brown, K.; Stützle, T. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res.* **2009**, *36*, 267–306. [CrossRef]
89. Hutter, F. Automated Configuration of Algorithms for Solving Hard Computational Problems. Ph.D. Thesis, University of British Columbia, Vancouver, BC, Canada, 2009.
90. Coates, A.; Ng, A.Y. The importance of encoding versus training with sparse coding and vector quantization. In Proceedings of the 28th International Conference on Machine Learning (ICML), Washington, DC, USA, 28 June–2 July 2011.
91. Coates, A.; Ng, A.; Lee, H. An analysis of single-layer networks in unsupervised feature learning. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 215–223.
92. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2623–2631.
93. Rapin, J.; Teytaud, O. Nevergrad—A Gradient-Free Optimization Platform. 2018. Available online: https://GitHub.com/FacebookResearch/Nevergrad (accessed on 10 September 2022).
94. Liaw, R.; Liang, E.; Nishihara, R.; Moritz, P.; Gonzalez, J.E.; Stoica, I. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv* **2018**, arXiv:1807.05118.
95. Bergstra, J.S.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. In Proceedings of the Advances in Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; pp. 2546–2554.
96. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NA, USA, 3–6 December 2012; pp. 2951–2959.
97. Li, L.; Jamieson, K.; Rostamizadeh, A.; Gonina, E.; Hardt, M.; Recht, B.; Talwalkar, A. Massively parallel hyperparameter tuning. *arXiv* **2018**, arXiv:1810.05934.
98. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **2017**, *18*, 6765–6816.
99. Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W.M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; Simonyan, K.; et al. Population based training of neural networks. *arXiv* **2017**, arXiv:1711.09846.
100. Bergstra, J.; Bardenet, R.; Kégl, B.; Bengio, Y. Implementations of algorithms for hyper-parameter optimization. In Proceedings of the NIPS Workshop on Bayesian Optimization, Sierra Nevada, Spain, 16–17 December 2011; p. 29.
101. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
102. Das, L.; Sivaram, A.; Venkatasubramanian, V. Hidden representations in deep neural networks: Part 2. Regression problems. *Comput. Chem. Eng.* **2020**, *139*, 106895. [CrossRef]