Neural Computing and Applications (2023) 35:1379–1394
https://doi.org/10.1007/s00521-022-07803-3

ORIGINAL ARTICLE

Deep reinforcement learning for automated search of model parameters: photo-fenton wastewater disinfection case study

Sergio Hernández-García[1] · Alfredo Cuesta-Infante[1] · José Ángel Moreno-SanSegundo[2] · Antonio S. Montemayor[1]

Received: 20 May 2022 / Accepted: 6 September 2022 / Published online: 26 September 2022

Abstract
Numerical optimization solves problems that are analytically intractable at the cost of arriving at a sufficiently good but rarely optimal solution. To maximize the result, optimization algorithms are run with the guidance and supervision of a human, usually an expert in the problem. Recent advances in deep reinforcement learning motivate interest in an artificial agent capable of learning to do the expert's task. Specifically, we present a proximal policy optimization agent that learns to optimize in a real case study such as the modeling of the photo-fenton disinfection process, which involves a number of parameters that have to be adjusted to minimize the error of the model with respect to the experimental data collected in several trials. The expert spends an average of 4 h to find a suitable set of parameters. On the other hand, the agent we present does not require a human expert to guide or validate the optimization procedure and achieves similar results in $2.5\times$ less time.

Keywords Deep reinforcement learning · Proximal policy optimization · Wastewater disinfection · Photo-fenton process

Mathematics Subject Classification 68T07 · 68T20 · 68T42 · 90C26

1 Introduction

Modeling is a task that requires the participation of an expert not only for translating the process dynamics into math expressions but also, often, to find the best set of model parameters. While the former is self-evident, the latter is sometimes dismissed as a simple optimization problem of minimizing a cost function for which there are several well-known methods such as gradient descent, bayesian optimization [1], ant colony optimization [2], genetic algorithms [3] or variable neighborhood search [4], just to mention a few. Gradient descent based methods are used in combination with different strategies such as multi-start, momentum or variable learning rate to avoid getting stuck into local optima or plateaus, but they usually require expert intervention to guide the process as well as to validate the result. The rest of the methods mentioned above are gradient-free and have proved to be efficient in dealing with gradient descent issues by shifting the effort of guiding the optimization to encoding the solutions and testing large populations, which usually entails high computational costs.

In this paper, we explore an in-between line, which takes advantage of gradient descent but at the same time liberates the expert from guiding and supervising it because is able to learn from its own past experience. Specifically, our goal is to have an iterative algorithm that, at time step t, acts

S. Hernández-García and A. Cuesta-Infante contributed equally to this work.

✉ Alfredo Cuesta-Infante
alfredo.cuesta@urjc.es

Sergio Hernández-García
sergio.hernandez@urjc.es

José Ángel Moreno-SanSegundo
jose.moreno.san@urjc.es

Antonio S. Montemayor
antonio.sanz@urjc.es

[1] Escuela Técnica Superior de Ingeniería Informática, Universidad Rey Juan Carlos, Madrid, Spain

[2] Escuela Superior de Ciencias Experimentales y Tecnología, Universidad Rey Juan Carlos, Madrid, Spain

over x_t, the candidate solution being evaluated, to produce the next candidate solution x_{t+1}.

Let s_t be the fully observable state of the optimization problem, which can be all the information needed to describe how good is a candidate solution. It can be as simple as its value in the objective function or it can incorporate more information, depending on the underlying problem. And let a_t be the action taken at time step t after observing s_t. Such an action not only modifies the candidate solution, but also causes the state of the optimization problem to transition from s_t to s_{t+1} in a deterministic way. This setting complies with the reinforcement learning (RL) framework, in which an agent learns to make decisions in an environment by trial and error based on a reward signal.

Current advances in deep neural networks have led to the emergence of deep reinforcement learning (DRL), in which the agent consists of a neural network that is trained with past experiences and produces a probability distribution across all the possible actions, referred to as *policy*. Since the neural network is expected to generalize, the agent is arguably trained to act similar to an expert guiding and supervising a gradient descent based optimization. In other words, this approach produces two outcomes: (1) a solution to the optimization problem and (2) a policy that leads the agent from any point in the solution space to a *valid* solution, meaning that its evaluation in the cost function is no greater than a given threshold. We separate both because the first is not consequence of the second, since there is no guarantee that the last candidate solution evaluated is better than another evaluated before. Once the optimization is solved, the agent is not necessary anymore for that particular problem, but having it is indeed a great advantage over gradient-free algorithms. If the conditions that shape the solution space change, for example due to non-stationary behaviors, the agent could be retrained using the last configuration to find the new optimum more efficiently. The agent could also be transferred to a *similar* problem, meaning that the state observed by the agent has the same representation.

To test the proposal, we carry out an experimental study on a real problem: the wastewater disinfection by a solar photo-fenton process. This is particularly relevant because antibiotic resistant bacteria have become one of the main global health challenges nowadays [5]. Advanced oxidation processes (AOPs), including photo-fenton, have already proved their ability to eliminate antibiotic resistant gene and antibiotics themselves [6] that currently remain untreated in wastewater treatment plants. Specifically, solar photo-fenton at neutral PH avoids three major troubles in the industrial implementation of AOPs, namely changes in water PH, conductivity and temperature [7, 8, 9, 10].

There are two main types of models for the solar photo-fenton process: empirical and mechanistic. Empirical models try to fit some pathogen inactivation curves over time to experimental conditions. This allows a better understanding of the process mechanisms, the relative relevance of each studied variable and their possible synergistic or antagonist effect. But those models are restricted to interpolation in the studied range of variables that are themselves limited to the laboratory scale. On the other hand, only mechanistic models, defining the elemental reactions and their rates, can cross the barrier of experimental ranges and allow an accurate simulation of the process industrial application. Hence, a whole methodology allowing to define and fit mechanistic models of processes at the research stage would cheapen and shorten their way to real applications in the daily live.

This paper focuses on the latter. Specifically, our case study is the photo-fenton model developed by Casado et al. [11] which involves 13 kinetic parameters for 512 reactions between 140 different species (135 bacteria at different steps of radiation or radical damage and 5 compounds of the Fenton process), but the methodological approach can be applied in a wide range of chemical processes in research and development.

Our goal is to find the optimal set of parameters that best match the mechanistic model to the experimental data. This problem presents several challenges:

- The solution space is non-convex, and has infeasible zones (with infinite or null values), plateaus and several local optima.
- Real experimental data are used for fitting the model which implies a non-exact correlation between the fitted model and the true data.
- The obtained result will be useful only if it is found in less, or at least, the same time than the engineer need using other optimization methods that requires intervention.
- Running the model represents a bottleneck, so it is important to reduce the number of executions.

In order to deal with the challenges listed above, we try up to 12 different solvers, all of them based on a proximal policy optimization (PPO) agent [12]. The best performing agent in this broad comparison is referred to as *reinforcement learning with direct actions and balanced memory*. In summary, this paper presents the following contributions:

- We provide a method of solving the photo-fenton model that can be used by the average chemical engineer with little additional effort. Since the engineer has to code the chemical model in any case, it is only necessary to refactor that code to match the specifications of the reinforcement learning environment [13].
- We show how to translate well-stablished results in AI and DRL into real problems, as recommended in [14].

Indeed, these kind of algorithms are mainly tested on benchmarks. Although these provide a good empirical proof of an algorithm's performance, they do not tell anything of how to adapt them to real-life problems so performance is usually poorer.

- We also introduce a novel and simple technique to sample and balance the training data collected during the online interaction with the environment. This technique helps the reinforcement learning agent to avoid getting stuck when the reward received is not informative enough.

The rest of the paper is organized as follows. Section 2 summarizes works related with our proposal. Section 3 provides the terminology and background of reinforcement learning and the agent used. Section 4 exposes the problem addressed and the solution proposed in terms of deep reinforcement learning. Section 5 discusses the experiments carried out. Finally, Sect. 6 points out the conclusions.

2 Related works

From an academic point of view, neural networks have been used in optimization within the *leaning to learn* [15] framework. According to this proposal, a neural network is trained in a supervised manner to act as a surrogate for a gradient descent based optimizer to update the weights of another neural network in regression and classification problems.

Another approach is learning an optimizer by means of deep reinforcement learning (DRL). In [16], an agent was trained to optimize linear and quadratic regression algorithms in 101 benchmark tasks and another agent is trained to be able to optimize neural networks with thousand of parameters. DRL has also succeeded in combinatorial optimization benchmarks such as the travelling salesman problem [17] and the max-cut problem [18].

This approach to optimization has also been applied to real-world problems and, in particular, to chemical industry, for example in real-time optimization of hydrocracking [19], in batch bio-process optimization for finding alternatives to fossil based materials [20], in batch optimization of bioreactors for food industry [21], in real-time detection of pollution risk due to wastewater [22] and in the analysis of material qualities like hardness of aluminum alloys [23]. It has also been applied in other domains such as health care, for melanoma's gene regulation [24] or protein folding problems in the fight against hereditary diseases [25], or in the field of energy, as in [26] to manage the electric power in a building or a small city, or in [27] to maximize electrical energy generation with acceptable emission levels. A review on reinforcement learning applied to process control can be found in [28].

With regard to the photo-fenton process, there were a lot of efforts in the development of complex kinetochemical models [7, 10] but less in its optimization, which is sometimes reduced to the application of some old-fashioned stochastic gradient descent method or heuristics, mainly because it is beyond the competences of chemical engineers [11, 29].

Another line of work attempts to develop simpler models based on the geometry of the disinfection curve rather than on a reaction mechanism. These models are optimized through linear and exponential regression [6, 8, 9]. Some work along these line incorporates neural networks that are responsible for generating concentration curves over time [30, 31, 32, 33]. These methods outperform the latter by being capable of optimizing multiple parameters, but they do not allow to obtain an analytical expression of the model. This makes these methods unable to manage the model if experimental parameters change over time and they would need to be retrained with new data. As they use neural networks as a surrogate of a kinetochemical model, they are also unable to extrapolate when the order of magnitude of the experimental conditions changes [34].

Our proposal is the first, to our knowledge, to present a methodology that allows extrapolation to new data and avoids human intervention by using DRL to find the optimal set of parameters to fit a kinetochemical model to experimental data.

3 Background

The purpose of this section is to provide the terminology and essential concepts about reinforcement learning and the specific agent used in this article. Deeper and comprehensive explanations can be found in [12, 35, 36].

3.1 Reinforcement learning

Reinforcement learning (RL) is a machine learning (ML) framework in which an *Agent* learns to carry out a task by interacting with an *Environment*. The fundamental assumption is that this goal can be achieved maximizing the cumulative reward along the sequence of actions taken or decisions made.

The basic training process consists of the loop depicted in Fig. 1. At time t, the agent observes the state of the environment s_t and takes an action a_t. The action produces the environment to transition from state s_t to state s_{t+1}. Additionally, the environment also implements a function $R(s_t, a_t, s_{t+1})$ that produces a *Reward* signal r_t which helps

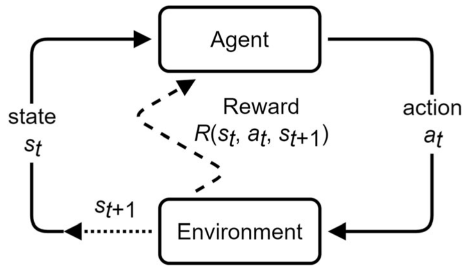


Fig. 1 Reinforcement learning loop

the agent to gain knowledge about how to act. The loop is repeated until a stop criterion is triggered. Typically, this happens when the task is done or after a maximum number of iterations if not. At that point, an *Episode* ends. Typically, the training process consists of executing a number of episodes, each one starting from different, random, initial state.

During this process, the agent is learning to map states into actions, referred to as a *Policy* π, that can be either deterministic, such that $a_t = \pi(s_t)$, or stochastic, so $a_t \sim \pi(a|s_t)$. Typically, the policy is a parametric function and learning consists of finding a set of parameters that optimize an objective function. Thus, let G_t be the *Return*, defined as the cumulative reward along the forthcoming sequence of states due to the future actions taken according to the policy,

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = r_{t+1} + \gamma G_{t+1},$$

where $0 < \gamma < 1$ is the *Discount* factor, included to prefer rewards closer in time. Given that the return is defined for future time steps, it makes sense to consider its expected value. Thus, the *Value* of the current state s_t according to a policy π is defined as the expectation of the return at that time step. Taking into account the recursive property, it can be expressed as

$$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]|s_t, a_t \sim \pi(s_t)]. \tag{1}$$

Then, according to the RL framework, the optimal policy π^* is the one that maximizes the expected return, which is equivalent to maximizing the value of the current state. Thus, once trained, given any initial state, s_0, the agent will take a sequence of actions according to π^*, resulting into a sequence of states until the task is done.

In RL, the human must model the dynamics of a problem in the environment, which consists of: deciding the state variables, designing how states transition from one to the next and shaping the reward function. Reward shaping is the decision made about the reward function $R(s_t, a_t, s_{t+1})$. Tiny but frequent rewards usually make the agent to accumulate them in circular trajectories that do not reach the true goal, whereas big but sparse rewards may difficult the learning process as the dimensionality of the problem increases [35].

On the other hand, the agent consists of an algorithm that runs episodes in the loop described above. In practice, it usually incorporates an *exploration–exploitation* strategy. The purpose of Exploration is to act randomly, enabling the agent to discover useful actions. On the contrary, Exploitation consists of acting according to the policy learned so far. Then, a strategy is to schedule a combination of both during the training process. For example, an ϵ-greedy strategy [35] consists of sampling from a uniform distribution over the set of possible actions with a time variable probability ϵ.

3.2 Deep Reinforcement Learning

Mnih et al. presented an agent capable of mastering many Atari video games without any previous knowledge about the game itself [37]. Such an agent used a deep neural network as a tool for mapping states into actions. Since then, deep reinforcement learning (DRL) has become the *de facto* standard for addressing numerous RL problems, with several solutions proposed such as deep Q networks (DQN) [37], double DQN [38] or dueling DQN [39], deterministic policy gradient (DPG) [40], REINFORCE [41], asynchronous advantage actor-critic (A3C) [42] or proximal policy optimization (PPO) [12]. Besides, there are research efforts to make neural networks converge faster by fighting ill-conditioned problems, saddle points or vanishing gradients [43].

All these DRL agents incorporate a *Replay memory*, also known as *Replay buffer*, that stores tuples (s_t, a_t, r_t, s_{t+1}), or variations of it, resulting from the interaction with the environment, referred to as *experiences*. The Replay memory is used as the data set for training the neural networks embedded in the agent.

The standard management of the Replay memory simply consists of storing the last n experiences. A more elaborated management is presented in [44], where experiences are prioritized to replay important transitions more frequently. In [45], every experience (s_t, a_t, r_t, s_{t+1}) is extended with extra information in order to create secondary objectives for the agent to learn more efficiently when the reward is sparse. In this paper, we propose a way for sampling experiences from the Replay memory based on the reward distribution and test it against the Standard management and the Hindsight Experience Replay (HER) proposed in [45].

3.3 Proximal policy optimization (PPO)

The ultimate goal of this paper is to solve an optimization problem with DRL. To this end, a wish list of agent skills would have the following elements:

- *Parallel execution* Candidate evaluation with the objective function is often the bottleneck in numerical optimization, so parallelization is a key requirement.
- *Continuous actions* The landscape of candidate solutions for the problem addressed in this paper is continuous, so the agent must be able to produce infinite possible outcomes.
- *Stability and convergence* It is a non-convex and ill-posed optimization problem, so the agent should be able to learn how to handle infeasible areas, plateaus and local optima.

Several DRL algorithms in the literature meet these specifications to a greater or lesser extent such as deep deterministic policy gradient (DDPG) [46], trust region policy optimization (TRPO) [47], proximal policy optimization (PPO) [12] or soft actor-critic (SAC) [48] just to mention a few. We choose PPO as suggested in [49], arguing that it has become one of the most widely used, has a simple and modular pipeline, incorporates a gradient penalty that makes it robust and less sensitive to hyperparameters and allows multiple environments to be run on simultaneous threads [14, 50].

A PPO agent consists of two neural networks, the *Actor* and the *Critic*, although the architecture can have common layers as in [42]. The input of both is the current state. The Actor estimates the parameters of the policy $\pi(a|s_t)$. Specifically, we take the usual assumption of the policy being a Normal distribution with fixed standard deviation σ, so the actor output estimates the mean $\hat{\mu}$. The Critic estimates the value of the state, $\hat{v}_\pi(s_t)$.

Every training iteration of PPO consists of two stages. In the first one, the agent acts according to the exploration–exploitation strategy defined, and experiences are stored in the buffer. Besides, none of the neural networks are updated, so it is fully parallelizable. In the second stage the weights of the neural networks are updated using only experiences from the buffer. Finally, before beginning a new training iteration, the buffer is emptied.

Each stored experience contains all the necessary data to calculate the targets for training both networks. Experiences extend the standard tuple with $\hat{\mu}_t$ and $\hat{v}_\pi(s_t)$, becoming $(s_t, a_t, s_{t+1}, r_t, \hat{\mu}_t, \hat{v}_\pi(s_t))$. Notice that a_t may be different from $\hat{\mu}_t$, as in this paper. The former is the action taken by the agent at time t, while the latter is the parameter that defines the policy of the actor at time t. In other words, a_t is a realization of the policy during the exploration. Hence, such a PPO agent is able to produce continuous actions.

Finally, PPO also remembers the previous policy, in order to compare the likelihood of the action proposed with respect to it against current policy. An action much more likely in the current policy can lead to excessive weight updates. To fight this issue, PPO clips ratio between the current policy and the old policy when both are evaluated on the pair (s_t, a_t).

4 Case study

In this paper, we propose to use PPO to conduct an intelligent and efficient exploration of the parameter space for the kinetochemical model of the solar photo-fenton process proposed in [11].

Solar photo-fenton is a wastewater treatment process that belongs to the Advanced Oxidation Technologies group. It is based in a RedOx cycle of iron salts with hydrogen peroxide as oxidant and solar radiation as reductant agent. The cycle produces hydroxil radicals able to damage a wide range of pathogens and pollutants. This cycle is coupled with direct solar disinfection (SoDis), which produces a parallel route of pathogens inactivation.

Hence, the solar photo-fenton kinetochemical model consists of three parametric submodels: the SoDis model, the Bacteria model and the Peroxide model, denoted $\mathcal{M}_S$, $\mathcal{M}_B$, $\mathcal{M}_P$, respectively; and let $\mathbf{K}_S$, $\mathbf{K}_B$, $\mathbf{K}_P$ be the set of parameters for each one of them, summarized in Table 1 with the same names given in [11].

Table 1 Experimental setup with the solar photo-fenton process

Subprocess	Model	Par.	N_I	N_T	N_M
Sodis	Sodis	3	7	120	18
Photo-Fenton	Bacteria	5	12	120	$9-24$
Photo-Fenton	Peroxide	5	12	120	$11-14$

Model	Set of parameters for a model ($\mathbf{K}_{\text{Model}}$)
Sodis	k_{solar}, $k_{\text{solar,repair}}$, n
Bacteria	k_{HO}, $k_{\text{HO,repair}}$, $k_{\text{HO,repair2}}$, $k_{\text{solar,repair2}}$, m
Peroxide	k_1, k_2, k_3, k_4, k_5

Detail of the parameters of each model in the whole solar photo-fenton model. More details about their meaning and the model can be found in [11]

Par. number of parameters in the model, N_I number of trials in which the disinfection process has been carried out in the reactor, each trial with different initial conditions, N_T duration (in minutes) of a single trial, N_M number of measurements taken during each trial (it is not always the same)

4.1 Fitting the photo-fenton kinetochemical model

For the sake of compactness, we do not repeat the complete explanation of the model in [11]. Instead, we briefly recap here how to proceed for fitting the model.

First, a series of trials is carried out with different initial concentrations of bacteria in a reactor exposed to solar radiation, which causes the bacterial concentration to decrease over time. During this process, the bacteria concentration is measured several times. The set of parameters for $\mathcal{M}_S$ can then be found comparing the true measured concentrations with the expected concentrations according to the model and the parameters proposed. This comparison is the core of the optimization problem that is explained below. Let $\mathbf{K}_S^*$ be the resulting set of parameters.

Second, a new series of trials is carried out, this time with different initial concentrations of peroxide as well as of bacteria in the reactor, and also exposed to solar radiation. Thus, the decrease of bacteria over time is now also due to the presence of peroxide. Similarly, during this process both bacteria and peroxide concentrations are measured several times. Then, by setting $\mathbf{K}_B = \{\forall k\_ = 10^{-5}, m = 1\}$ and $\mathbf{K}_S = \mathbf{K}_S^*$, $\mathbf{K}_P^*$ is found comparing the true measures of the peroxide with the outcome from $\mathcal{M}_P$. And finally, by setting $\mathbf{K}_S = \mathbf{K}_S^*$ and $\mathbf{K}_P = \mathbf{K}_P^*$, $\mathbf{K}_B^*$ is found in the same fashion.

The tuple $(\mathbf{K}_S^*, \mathbf{K}_B^*, \mathbf{K}_P^*)$ is the optimal set of parameters, i.e., those that best explain the experimental data collected with the model given. Since [11] presented the model already fitted using the sequential quadratic programming (SQP) method provided by GNU Octave, we will use that solution as baseline to compare our fully automated optimization method.

4.1.1 Experimental data and trial details

Photo-fenton trials There were 12 trials of 120 minutes each, all with different initial conditions as shown in the left panel of Fig. 2. Peroxide and bacteria concentrations are measured evenly in time, and its number ranges from 9 to 24 for bacteria and from 11 to 14 for peroxide. The middle and right panels in Fig. 2 show the measures and the model fitted according to [11]. All the odd-numbered trials are in the top panels and the even-numbered are in the bottom panels.

Solar disinfection trials There were 7 trials of 120 minutes each, with different solar radiation and fixed initial concentration of bacteria through all experiments. During each trial, the bacteria concentration was measured evenly 18 times.

Wallclock time per fitting The average time for fitting the whole model, guided and supervised by a human expert is 4 h.

4.1.2 The optimization problem

Let $\mathcal{M}$ be any of the models involved (Sodis, Bacteria or Peroxide), and let $\mathbf{K}_\mathcal{M}$ be the set of parameters for $\mathcal{M}$, as listed in Table 1. The ith trial with a given $\mathcal{M}$ is defined by a set of initial conditions $\mathbf{I}_\mathcal{M}$. Model $\mathcal{M}$ returns the estimated bacteria or peroxide concentration (depending on $\mathcal{M}$) at any given time step t, represented as $c_{i,t} = \mathcal{M}(\mathbf{K}_\mathcal{M}, \mathbf{I}_\mathcal{M}, i, t)$.

Since the experimental data are collected at specific times, we denote $\tau(j)$ to the time step at which the jth measure was taken, so $\chi_{i,\tau(j)}$ is the experimental data from the jth measurement in the ith trial. The concentration estimated by a model for the ith trial is then redefined to the same time step than the real concentration as $c_{i,\tau(j)} = \mathcal{M}(\mathbf{K}_\mathcal{M}, \mathbf{I}_\mathcal{M}, i, \tau(j))$.

Let $\mathcal{E}_\mathcal{M}$ be the error of model $\mathcal{M}$, defined as:

$$\mathcal{E}_\mathcal{M} = \sum_{i=1}^{N_I} \sum_{j=1}^{N_M} \left(\log_{10}(\chi_{i,\tau(j)}) - \log_{10}(c_{i,\tau(j)}) \right)^2, \tag{2}$$

then the goal is to find the set of parameters $\mathbf{K}_\mathcal{M}^*$ that best explain the experimental data at every step of the fitting process. Formally,

$$\mathbf{K}_\mathcal{M}^* = \arg\min_{\mathbf{K}_\mathcal{M}} \mathcal{E}_\mathcal{M}\left(\mathbf{K}_\mathcal{M}, \{\chi_{i,\tau(j)}\}_{\forall i,j}\right). \tag{3}$$

The Sodis model is indeed easy to optimize and a gradient descent quickly converges into $\mathbf{K}_S^*$. But Bacteria and Peroxide models are quite challenging, with infeasible regions and large plateaus, which makes it very difficult both to find a valid starting point for numerical optimization and to drive it through the parameter space. For this reason, this paper only focuses on searching $\mathbf{K}_B^*$ and $\mathbf{K}_P^*$.

To this end, in this paper we propose an automated optimization method based on DRL, which requires introducing an agent and modeling an environment. Additionally, we introduce several variations for the solver.

4.2 RL environment modeling

The environment consists of a mechanism to transition from state s_t to s_{t+1} and a reward function.

In this problem, the environment contains the full solar photo-fenton model $\{\mathcal{M}_S, \mathcal{M}_B, \mathcal{M}_P\}$, such that for any given set of parameters $\{\mathbf{K}_S, \mathbf{K}_B, \mathbf{K}_P\}$ it returns the estimated bacteria and peroxide concentration and the error $\mathcal{E}_\mathcal{M}$ for every trial and every time step.

Initial conditions for each trial

Trial Num.	Solar rad.	Initial Fe_2^+	E.coli conc.	H_2O_2 conc.
1	10	10	10^6	30
2	10	20	10^6	30
3	15	10	10^6	30
4	15	20	10^6	30
5	20	10	10^6	30
6	20	20	10^6	30
7	25	10	10^6	30
8	25	20	10^6	30
9	30	10	10^6	30
10	30	20	10^6	30
11	50	10	10^6	30
12	50	20	10^6	30
	$W \cdot m^{-2}$,	$mg \cdot L^{-1}$,	$CFU \cdot mL^{-1}$,	$mg \cdot L^{-1}$

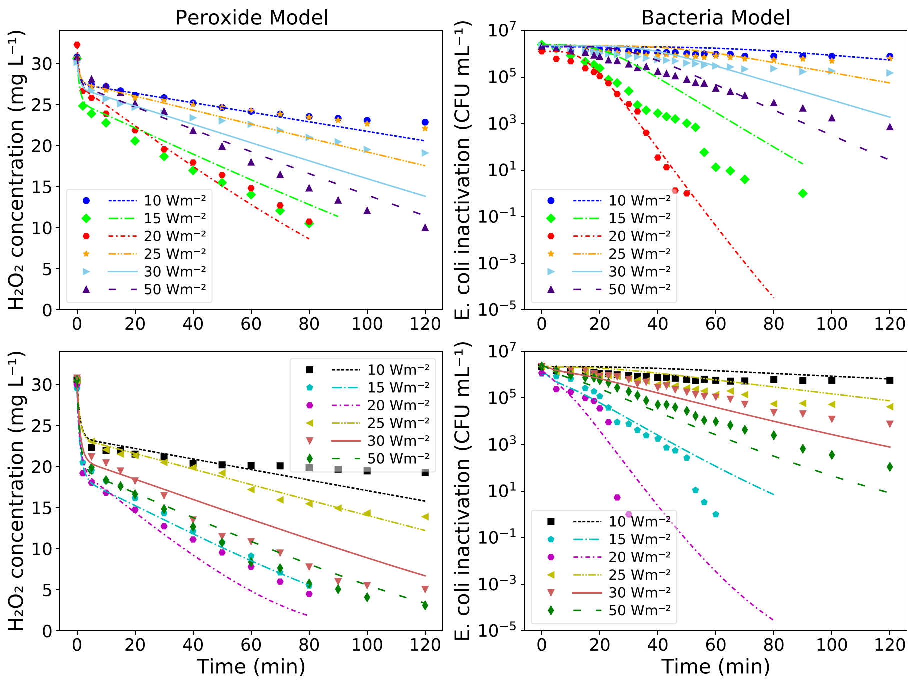


Fig. 2 Initial conditions for each trial and reference solution for the photo-fenton model from [11]. *Left:* Initial amount of solar radiation, iron (Fe_2^+), concentration of bacteria (*E. coli*) and concentration of hydrogen peroxide (H_2O_2) for each trial over the peroxide and bacteria models. *Center:* Experimental data ($\chi_{i,\tau(j)}$) and concentration curves generated by the model ($c_{i,\tau(j)}$) for reference solution to the peroxide model with error $\mathcal{E}_{\mathcal{M}_P} = 0.297$. *Right:* Experimental data ($\chi_{i,\tau(j)}$) and concentration curves ($c_{i,\tau(j)}$) for reference solution to the bacteria model with error $\mathcal{E}_{\mathcal{M}_B} = 0.520$. Odd rows shows the odd experiments and even rows the even experiments for visualization clarity in both center and right panels

The state s_t encodes all the information about the optimization problem at some time step t that is considered necessary for the agent to make a decision. According to Sect. 4.1, Peroxide and Bacteria models are optimized sequentially, and $\mathbf{K}_S^*$ is known; so, let $\mathcal{M} \in \{\mathcal{M}_B, \mathcal{M}_P\}$ be the model being optimized, then we define s_t, the current state of that optimization, as the pair of all the concentrations estimated by the model and all the parameters used in the model, i.e., $\left(\{c_{i,\tau(j)}\}, \mathbf{K}_\mathcal{M}\right)_t$, for all $i \in N_I$ and $j \in N_M$, at time step t.

The agent will modify $\mathbf{K}_\mathcal{M}$ (see below) so the transition mechanism is simply running the model with the new set of parameters.

The reward function produced at every time step is

$$R = \begin{cases} 1/\mathcal{E}_\mathcal{M} & 0 < \mathcal{E}_\mathcal{M} < +\infty \\ -10 & \text{otherwise.} \end{cases} \tag{4}$$

The *otherwise* case includes null values of $\mathcal{E}_\mathcal{M}$, if any.

4.3 PPO agent

For this particular problem, it is useful for the agent to have a history of successive states until the current one. This history supplies implicit information about the gradient and the parameter space landscape. Thus, we define an *Observation* as a sequence of 20 states, $O_t = \{s_{t-19}, s_{t-18}, \ldots, s_t\}$.

Our PPO agent consists of two independent dense neural networks: one for the Actor and another for the Critic. Each has a long short-term memory (LSTM) input layer in order to process each observation. The Critic network has a single output neuron with identity activation. The Actor network has as many outputs as parameters are in the model, i.e., as many elements in $\mathbf{K}_\mathcal{M}$. Each output produces μ, the center of a normal distribution that is sampled during exploration, depicted in Fig. 3. The array whose elements are these samples is related to the action taken by the agent at that time step according to two variants that we present.

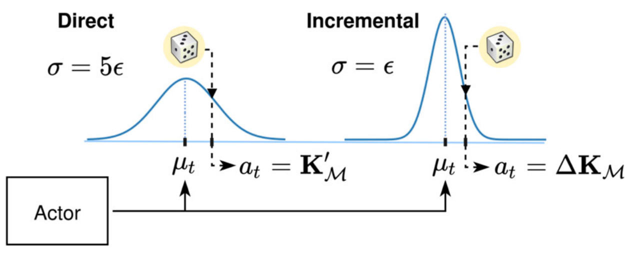


Fig. 3 Representation of how actions are obtained from actor's output in exploration. Dice mean sampling from the distribution

- *Incremental* The actor's output activation is a tanh and the normal distribution has standard deviation $\sigma = \epsilon$, with $0 < \epsilon < 1$. The sampled array is then $a_t = \Delta\mathbf{K}_{\mathcal{M}}$, meaning that the actions are increments with respect to the last set of parameters. The next state becomes $s_{t+1} = \left(\{c_{i,\tau(j)}\}, \mathbf{K}_{\mathcal{M}} + a_t\right)_{t+1}$.
- *Direct* The actor's output activation is the identity and the normal distribution has standard deviation $\sigma = 5\epsilon$, with $0 < \epsilon < 1$. The sampled array is then $a_t = \mathbf{K}'_{\mathcal{M}}$, meaning that the actions are the new parameters for the next time step, regardless of their previous values. The next state becomes $s_{t+1} = \left(\{c_{i,\tau(j)}\}, a_t\right)_{t+1}$.

Hiperparameter ϵ has been included so the standard deviation can be controlled with a single value ranging the unit interval in any of the two variants. Finally, $a_t = \mu$ during exploitation for both.

To train the agent, we also try other solving algorithms, including exploration strategies, balancing the Memory Replay, using Hindsight Experience Replay and incorporating expert knowledge to the agent. They are all presented below.

4.3.1 Exploration strategies

Hiperparameter ϵ allows to customize the exploration strategy. Thus, we test three different ones, depicted in Fig. 4.

- *Single annealing* At each iteration, ϵ decays by a fixed damping factor $d \in (0, 1)$, i.e., $\epsilon_{t+1} = d \cdot \epsilon_t$
- *Multi-annealing* The single annealing is repeated four cycles with the same initial $\epsilon_0 = 1$.
- *Meta-annealing* Same than multi-annealing but linearly decreasing ϵ_0, i.e., $\epsilon_0^{(\text{cycle } k+1)} = \epsilon_0^{(\text{cycle } k)} - 0.2$

In all of them, as ϵ decreases the actions sampled are closer to the center of the distribution, thus closer to the actions taken during exploitation.

4.3.2 Balancing the replay memory

The standard Replay memory management leads to an unbalanced data set for training the agent because most of the data collected comes from experiences with low reward (high error) or actions that produce parameters leading to numerical errors such as division by or log of zero, while just a few experiences achieve a high reward (low error). This unbalance is more noticeable at the beginning of training, when the agent behavior is mostly random and makes the agent to waste a lot of time learning what not to do instead of what to do.

To solve this issue, in this paper we propose a *Balanced Replay memory* based on the histogram that approximates the distribution of rewards. Specifically, we use 30 bins, so that each experience will belong to only one of them depending on its reward. Then, we randomly sample the same number of experiences from each bin obtaining a balanced data set in terms of rewards for training. Since the reward function is intrinsically related with the error values, the data will be also balanced in terms of error.

4.3.3 Hindsight experience replay

HER [45] introduces secondary goals to lessen the issues rising due to sparse rewards. To this end, we first define the following: the *goal* state, the *stop* state and the *state extension* operation. Let $s_{\text{goal}} = \left(\chi_{i,\tau(j)}\right)$ for $i \in N_I$ and $j \in N_M$ be the *goal* state, consisting of the experimental data used to fit the model, and let $s_\Omega = \left(c_{i,\tau(j)}\right)_{t_{\text{end}}}$ also for $i \in N_I$ and $j \in N_M$ be the *stop* state, which consists of all the concentrations measured in the last iteration of an episode (i.e., when the episode stops). Since we run 12 episodes per round, there will be 12 s_Ω, so let Ω be the set that contains them all. Both the goal and the stop states are arranged in vectors, and we define the *state extension* operation of two states s_a and s_b as their serial concatenation, resulting into an *extended* state denoted as $(s_a || s_b)$.

HER acts extending all the current and the next states in the Replay memory both with the goal and the stop state. In other words, each tuple $(s_t, a_t, s_{t+1}, r_t, \hat{\mu}_t, \hat{v}_\pi(s_t))$ yields two:

$$\begin{cases} \left((s_t||s_{\text{goal}}), a_t, (s_{t+1}||s_{\text{goal}}), r_t, \hat{\mu}_t, \hat{v}_\pi((s_t||s_{\text{goal}}))\right), \\ \left((s_t||s_\Omega), a_t, (s_{t+1}||s_\Omega), r_t, \hat{\mu}_t, \hat{v}_\pi((s_t||s_\Omega))\right), \end{cases}$$

which are stored in the Replay memory.

Finally, we replace the reward function given in (4) with

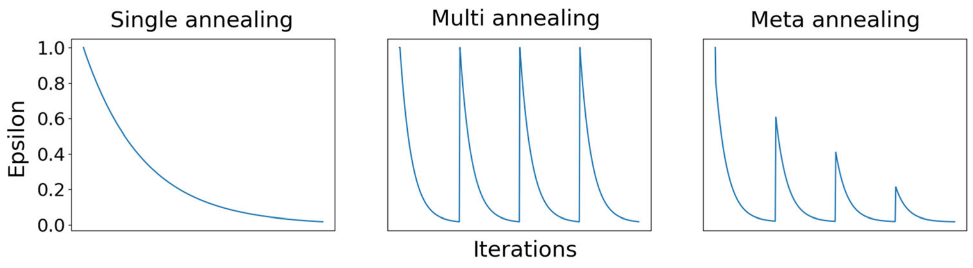


Fig. 4 Value of epsilon for different exploration strategies through iterations. *Left* Single annealing. *Center* Multi-annealing. *Right* Meta-annealing

Table 2 All the different PPO agents used

Type of action	Replay mem. management	Expert knowledge	Starting point	
			Valid	Random
Increments	Standard	None	I	I-R
	Balanced	None	IB	IB-R
Direct values	Standard	None	D	D-R
	HER	None	DHER	DHER-R
	Balanced	None	DB	DB-R
		Optimizer	DBO	DBO-R

The two rightmost columns give the acronym of every agent solver resulting from the combination on its left, with a valid starting point or a random (not necessarily valid) one

$$R = \begin{cases} (1/\mathcal{E}_{\mathcal{M}}) - 1 & \text{if } 0 < \mathcal{E}_{\mathcal{M}} < +\infty, \\ 0 & \text{if } s_{t+1} \text{ is any } s_\Omega \in \Omega, \\ -10 & \text{otherwise.} \end{cases} \quad (5)$$

Since the error $\mathcal{E}_{\mathcal{M}}$ is always nonnegative, the lower it gets, the greater the reward, and for $\mathcal{E}_{\mathcal{M}} > 1$, the reward is negative down to -1. Besides, reaching a stop state has the same reward than $\mathcal{E}_{\mathcal{M}} = 1$. Hence, the stop states are our secondary goals for HER. As in the rest of variants, after training the agent's networks, the Replay memory is emptied.

4.3.4 Incorporating expert knowledge

The agent is learning from its own trial-and-error experience in a continuous action space, so it is very unlikely for it to take the best possible action. Since actions are due to the Actor, which is trained with the Replay memory, a possible extra aid is to store experiences that come from a numerical optimization algorithm. We refer to this variant as Optimizer, for *Expert knowledge from an optimizer*. Specifically, we include an additional step of the Nelder–Mead simplex optimizer [51] at the beginning of the experiences collection stage of PPO. Thus, the Replay memory has 12% of expert experiences each time the Actor is trained.

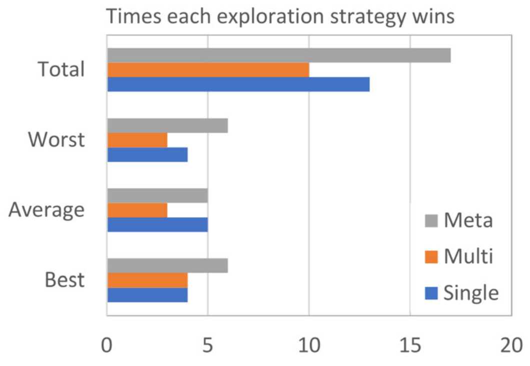
5 Experiments

We present two experiments. The first one aims at selecting one exploration strategy out of the three proposed. In the second experiment, we use PPO to fit the photo-fenton model to the experimental data holding the exploration strategy chosen previously.

Both experiments follow the same schedule: For a chosen agent, we execute a total of 9 tests, each one of them with 12 episodes of, at most, 40 iterations; and all the episodes are executed in parallel. The first three tests are repeated for 11 rounds, the next three for 21 rounds and the last three for 42 rounds. The number of rounds is chosen to mimic a total execution time of 1, 2 and 4 hours per test, although there is not a perfect match because many episodes may end before the 40th iteration. Indeed, every continuous parameter is modified with increments of 0.1 within the range $(-20, 20)$ and every integer parameter is

Table 3 Experimental results of the different exploration strategies for I, B, IB and DB agents in the peroxide smodel (colour figure online)

Agent	Rounds	Best	Average	Worst
I	42	Multi	Multi	Multi
	21	Meta	Meta	Meta
	11	Multi	Multi	Multi
IB	42	Single	Single	Single
	21	Single	Single	Single
	11	Multi	Single	Single
D	42	Meta	Meta	Meta
	21	Meta	Meta	Meta
	11	Meta	Meta	Meta
DB	42	Single-Meta	Single-Meta	Single-Meta
	21	Multi	Multi	Multi
	11	Single-Meta	Single	Meta



The table shows the exploration strategy that gets the lowest error. The plot below shows the occurrence of each strategy in the table in the interval [1, 40]. Hence, at the end of every test, an agent has tested about 5,000 candidate set of parameters in 11 rounds and about 40,000 in 42 rounds, which is much less than a grid search on the parameter space.

Finally, we take the human *agent* as a reference value both for the error reached by the best solution proposed for each model and for the total execution time, which is 4 hours as referred to in Sect. 4.1.1.

5.1 Exploration strategy selection

In this first experiment, we make an ablation study about how those different exploration strategies described in Section 4.3.1 impact on the agent's performance. To this end, we try the agents identified as I, IB, D and DB; in other words, we cross the two variants for taking actions (incremental and direct) with and without balancing the Replay memory. Besides, for the sake of compactness and computational efficiency, the agents only optimize the peroxide model. The reason is that the exploration strategy is a meta-algorithm independent of the agent used, which is PPO and its variants, and the optimization problem. Hence, we use this experiment as a model to estimate the best strategy.

The experiment follows the schedule given above. We run 3 tests per strategy, for each agent and each choice of rounds, and report the errors for each agent split in three ways: the lowest (Best), the greatest (Worst) and the average error. For the sake of clarity, in Table 3 we only report the winner strategy, i.e., the one with less error in each split. Next, we count the number of times each strategy is both in every split (the three rightmost columns) and in the whole table, and show the results in the plot below.

Table 3 clearly shows that meta-annealing strategy is a better choice both in the worst and best test and it is similar to single annealing in average. Hence, we select the meta-annealing strategy for the next experiment.

5.2 Model fitting

In this section, we present the results using the complete photo-fenton process as modeled in [11]. Specifically, we aim at optimizing only the Peroxide and the Bacteria models sequentially, as explained in Sect. 4.1.

5.2.1 Peroxide model optimization

First, we search for the set of parameters $\mathbf{K}_P^*$ that minimizes $\mathcal{E}_{\mathcal{M}_P}$, the error of the peroxide model as defined in (2). To this end, we follow the schedule given above, but this time testing all the agents listed in Table 2. We report the lowest error attained in the nine tests carried out by each agent split as in Sect. 5.1, and the results are shown in Table 4. In that table, bars in the Time column are scaled to the interval [0:0:0] to [0:59:59], in `hh:mm:ss` format,

Table 4 Lowest errors attained on the peroxide Model with different agents during a given number of rounds

Peroxide model ($k1$, $k2$, $k3$, $k4$, $k5$)					
Agent	Time	Rounds	Best	Average	Worst
Human			0.297	0.297	0.297
I	0:23:15	42	0.177	0.280	0.335
	0:10:19	21	0.215	0.278	0.324
	0:05:30	11	0.151	0.672	1.471
IB	0:22:07	42	0.157	0.221	0.284
	0:11:46	21	0.179	0.197	1.477
	0:05:39	11	0.482	0.837	1.481
D	0:51:39	42	0.148	0.153	0.158
	0:32:07	21	0.157	0.174	0.186
	0:18:44	11	0.224	0.244	0.254
DB	0:51:01	42	0.144	0.150	0.160
	0:31:02	21	0.150	0.170	0.185
	0:17:02	11	0.150	0.177	0.195
DHER	0:54:04	42	0.142	0.152	0.161
	0:31:19	21	0.153	0.179	0.212
	0:17:11	11	0.250	0.247	0.250
DBO	6:01:36	42	0.149	0.155	0.167
	1:11:06	21	0.168	0.200	0.246
	0:21:57	11	0.181	0.243	0.339
I-R	0:12:57	42	0.329	0.377	0.473
	0:06:37	21	0.284	0.392	0.562
	0:03:19	11	0.442	0.738	1.153
IB-R	0:14:59	42	0.234	0.266	0.307
	0:06:56	21	0.281	0.667	1.418
	0:03:28	11	0.200	0.988	1.481
D-R	0:13:01	42	0.142	0.150	0.157
	0:06:31	21	0.163	0.188	0.201
	0:03:22	11	0.162	0.241	0.295
DB-R	0:12:33	42	0.136	0.151	0.161
	0:05:58	21	0.156	0.166	0.179
	0:03:08	11	0.151	0.208	0.287
DHER-R	0:13:04	42	0.142	0.150	0.157
	0:06:01	21	0.161	0.175	0.186
	0:03:08	11	0.162	0.184	0.223
DBO-R	6:02:24	42	0.149	0.208	0.262
	1:10:52	21	0.143	0.171	0.202
	0:22:34	11	0.148	0.189	0.226

After repeating three tests, the table reports the best, average and worst error achieved by the solutions of each test. The agents are identified according to Table 2

Time is in format hh:mm:ss

whereas bars in the three rightmost columns are scaled between the minimum and maximum of every column.

A look at the Time column indicates that the use of expert knowledge requires more than one hour per test when running 21 or 42 rounds of it. We blame this poor performance on the extra time the optimizer uses. Thus, in comparison with the errors achieved with other agents, a first conclusion is that DBO and DBO-R are candidates to be dismissed. In the same line, agents D, DB and DHER are also at the bottom of the ranking because their performance in terms of error is similar to others that consume much less execution time. These three agents require a valid starting point, which is usually hard to find, so extra time is required. The reason why I and IB perform faster is that incremental actions are more conservative than direct actions; hence, after finding a valid starting point, it is less likely to move to an invalid one.

However, when the starting point is not required to be valid, aggressive actions are more effective in moving to another area of the parameter space in case of an invalid start. Thus, D-R, DB-R and DHER-R achieve better performance than I-R and IB-R, beating the human agent (error of 0.297). Moreover, it takes barely 3 min to achieve a $\mathbf{K}_P^*$ of such quality even in the worst of the three tests performed with only 11 rounds. In other words, the agent consumes 3 minutes out of the 4 hours set as an upper bound, leaving almost the entire time budget for tuning the bacterial model.

5.2.2 Bacteria model optimization

In this section, we search for the set of parameters $\mathbf{K}_B^*$ that minimizes $\mathcal{E}_{\mathcal{M}_B}$. We follow the same schedule as before, but all agents without a random starting point are discarded because of the excessive execution time, even those with fewer rounds. For this reason, we modify the number of rounds during the tests for the rest of agents, making then to be 21, 42 and 84. Results are presented in Table 5.

DBO-R still takes a prohibitively long time for the same reason than in Sect. 5.2.1; the optimizer needs that extra time for providing expert knowledge to the Replay memory. The tests with 84 rounds also exceed the time budget. Since the method proposed liberates the human of this task entirely, one could consider leaving the computer to work overnight if it achieves similar performance. Thus, given around 6 hours, I-R and IB-R show to be able to approximate the human performance (error 0.520) in the best and the average of the three tests. For the rest of tests, the average error is between 80 and 100% greater than the human. However, errors lower than 1 result into a model than predicts a drop on the bacteria concentration in the same order of magnitude than those trials in which the disinfection is more effective (trials 3–6).

Table 5 Lowest errors attained on the Bacteria Model with different agents during a given number of rounds

Bacteria Model (k_{HO},$k_{HO,repair}$,$k_{HO,repair\ 2}$,$k_{solar,repair\ 2}$,m)					
Agent	Time	Rounds	Best	Average	Worst
Human			0.520	0.520	0.520
I-R	5:44:34	84	0.574	0.698	0.832
	3:17:48	42	0.894	0.986	1.123
	1:33:49	21	0.631	0.912	1.273
IB-R	6:27:13	84	0.596	0.716	0.760
	2:58:31	42	0.771	0.841	0.894
	1:45:27	21	0.567	0.718	0.959
D-R	5:26:56	84	0.803	0.878	0.945
	2:43:40	42	0.931	0.989	1.028
	1:45:23	21	0.916	1.026	1.163
DB-R	5:22:09	84	0.875	0.994	1.064
	2:46:30	42	0.672	0.837	0.945
	1:22:34	21	0.643	0.847	0.994
DHER-R	5:25:40	84	0.851	1.018	1.169
	2:46:29	42	0.865	0.971	1.045
	1:23:49	21	0.596	1.257	2.322
DBO-R	33:15:28	84	0.882	0.927	1.045
	14:13:45	42	1.056	1.076	1.134
	6:43:02	21	0.968	1.000	1.043

After repeating three tests, the table reports the best, average and worst error achieved by the solutions of each test. The agents are identified according to Table 2

Time is in format hh:mm:ss

5.2.3 Joint optimization

To decide which agent performs better jointly in both optimizations, we can only compare those tried in both models, i.e., those with "Random" starting point, denoted with suffix -R. To this end, we plot the results of Tables 4 and 5 in the left and right panels of Fig. 5, respectively. Both represent the error versus the agent. The blue dotted and red dashed lines are the average error over three tests for 21 and 42 rounds, respectively. The solid black line is the sum of these two. The blue and orange areas are bounded by the worst and the best error.

Thus, agent DB-R has the lowest errors in the peroxide model and the second lowest average errors in the bacteria model. Both IB-R and DB-R result into a similar average error for 42 rounds, which takes almost 3 hours: one hour less than the human. But IB-R, which is the best option in the bacteria model, performs much worst in the peroxide model lasting the same time than DB-R. Altogether, DB-R

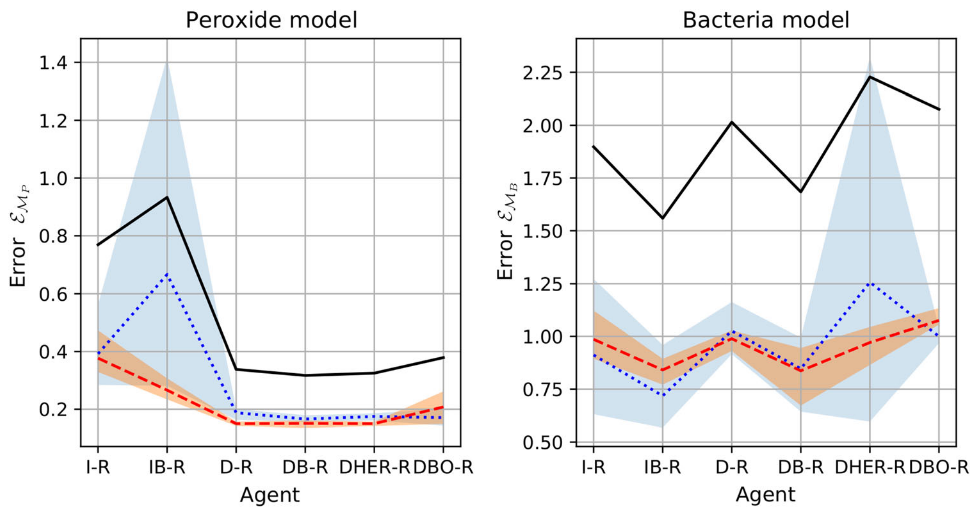


Fig. 5 Average, best and worst error of each agent for peroxide (left) and bacteria (right) models for tests of 21 (blue) and 42 (red) rounds. Blue dotted and red dashed lines are average error, blue and orange areas bound the best and worst error, and solid black line is the sum of both average errors (colour figure online)

is proposed as the agent with the best performance, able to fit the experimental data in 21 rounds, with a total time slightly below 1 hour and 30 min. Hence, with a time budget of 4 hours and a half it is possible to run it three times to have a notion of the uncertainty in the errors attained.

6 Conclusion

In this paper, we show that DRL can be used to carry out an optimization problem in a real-life task that requires a human expert. Specifically, we use a PPO agent with up to 12 different variants and three exploration strategies. Results show that the method proposed could free the human from this task with little effort because the reward is a simple relation with the error, the environment is the model being fitted and the agent is the same that can be used for any other DRL problem. Thus, the method could work on any other environment modeled and programmed as the one we deal with.

On the other hand, although we made some first steps, further research in this area would be useful to completely cover the needs of the engineers in which the use of advanced optimizers is beyond their competencies. Therefore, we suggest two possible guidelines for future work in this area: (1) the integration of new DRL algorithms beyond PPO. There are many possibilities when it comes to selecting a reinforcement learning algorithm and the benchmarks do not show a clear winner [14]. An ablation study of some of state of the art DRL agent would shed light on which one is preferable to use. (2) Introducing techniques like activated gradients to boost the neural networks convergence would encourage the DRL agent to explore faster.

Appendix A Network's architectures

Our PPO agent uses two independent neural networks: one for the Actor and another for the Critic.

Actor network architecture

We use an input layer of 128 LSTM neurons with hyperbolic tangent activation, followed by a first hidden layer of 256 dense neurons with ReLU activation, a second hidden layer of 256 dense neurons with ReLU activation and a third hidden layer of 128 dense neurons with ReLU activation. The output layer is dense with as many neurons as number of parameters to optimize depending on the model, and its activation depends on the type of agent. Thus, I, I-R, IB and IB-R use the hyperbolic tangent activation, whereas all the others use the idendity activation.

Critic network architecture

The input and the first hidden layers are exactly the same as in the Actor network. Then We use a second hidden layer of 128 dense neurons with ReLU activation. The output consists of a single dense neuron with identity activation.

Appendix B Data preprocessing

State normalization

Before being processed by the agent, the state is scaled to range $[-1, 1]^n$, where n depends on the agent used.

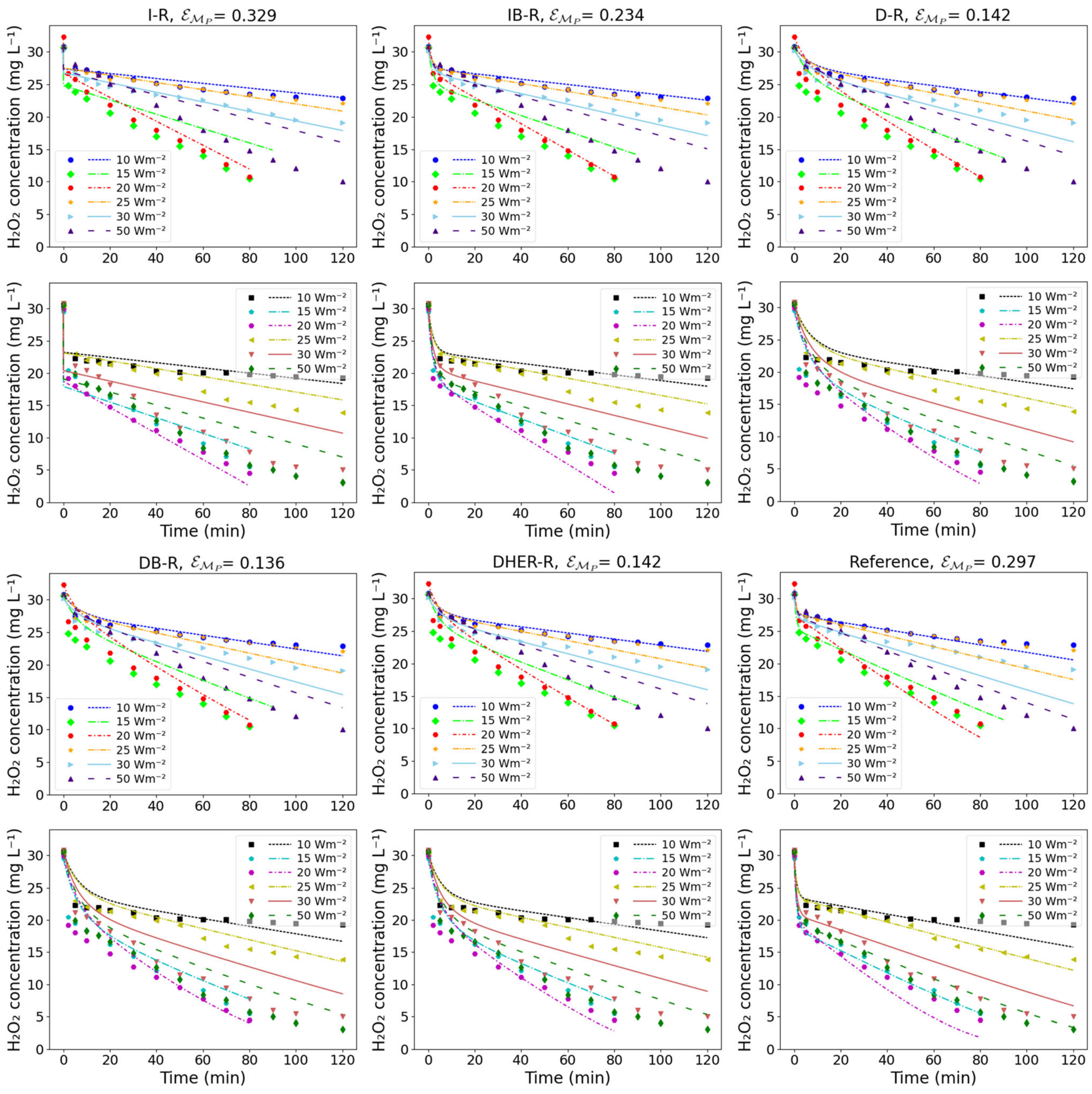


Fig. 6 Experimental H_2O_2 concentration measured in 12 different trials ($\chi_{i,\tau(j)}$, represented with marks) vs. estimated concentrations ($c_{i,\tau(j)}$, represented with lines) due to the model best fitted (lowest error) with five versions of the PPO optimization agent, together with the reference model [11]. For the sake of clarity, the 12 trials are split in two panels, 6 above and 6 below

Peroxide data normalization

We scale every $c_{i,\tau(j)}$, the H_2O_2 concentration values estimated by the model, to keep them within the interval [0, 1]. It is possible because we know that the first estimated value is greater than the rest for a given trial, except if it is a non-valid one, such as infinite or null. In that case it is replaced by -1.

Bacteria data normalization

The bacteria concentration ranges within 6 orders of magnitude, between 10^6 and 0, which is ideal event of total bacteria inactivation. For this reason, we use the base-10 log of the concentration instead. Once more, non-valid data are replaced by -1

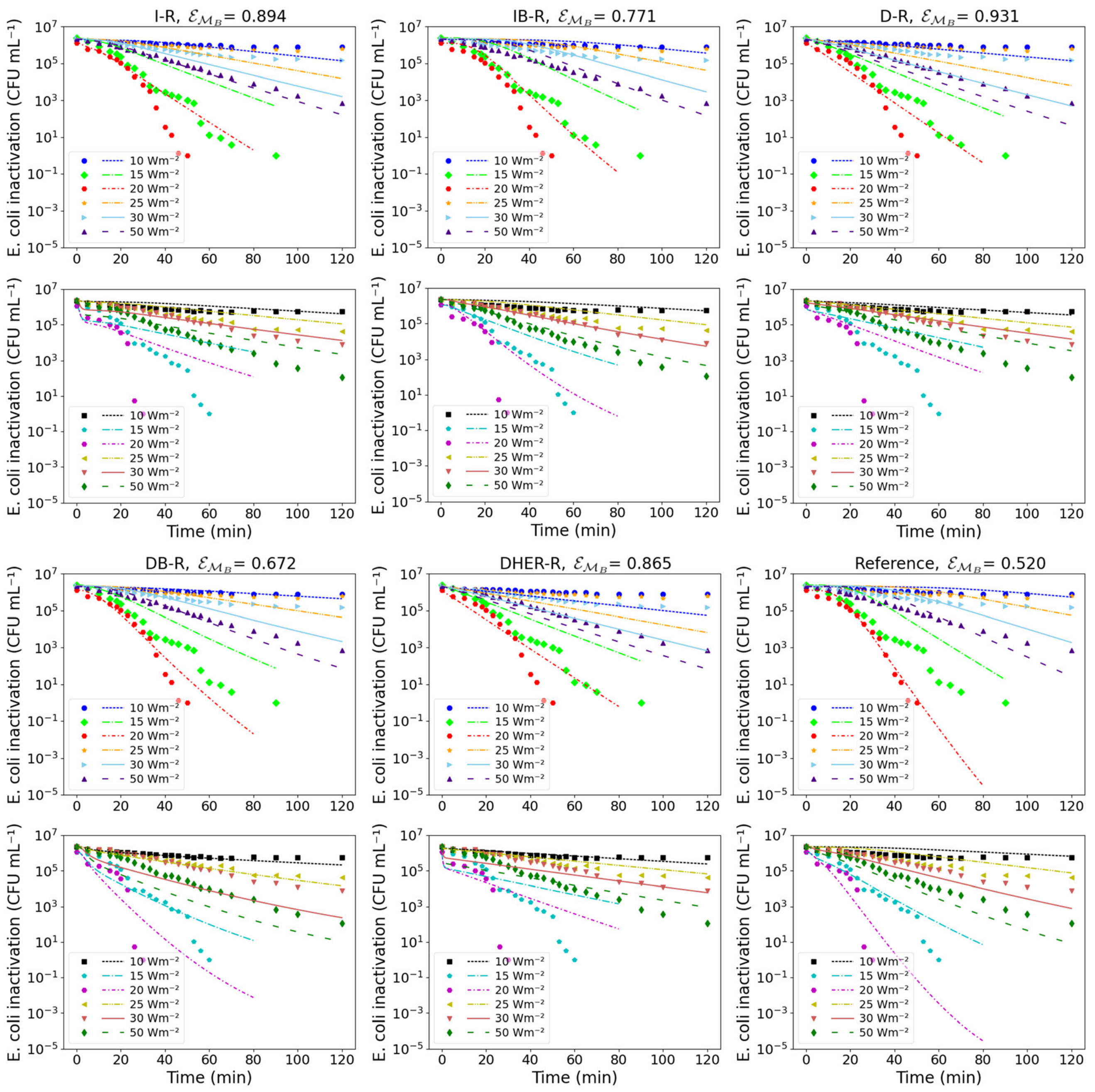


Fig. 7 Experimental E.coli concentration measured in 12 different trials ($\chi_{i,\tau(j)}$, represented with marks) vs. estimated concentrations ($c_{i,\tau(j)}$, represented with lines) due to the model best fitted (lowest error) with five versions of the PPO optimization agent, together with the reference model [11]. For the sake of clarity, the 12 trials are split in two panels, 6 above and 6 below

Appendix C Qualitative results

Figures 6 and 7 show some of the best Peroxide and Bacteria models, respectively, tuned with the best parameters found with the 42-round tests. We only show agents I-R, IB-R, D-R, DB-R and DHER-R because these are used on both models and give a total execution time less or equal to three hours, which is 1 hour less than the reference [11], also shown in the lower right corner.

Acknowledgements This work has been funded by Comunidad de Madrid Y2018/EMT-5062 and Ministerio de Ciencia, Innovación y Universidades RTI2018-098743-B-I00 (MICINN/FEDER), both in Spain.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The authors did not receive support from any organization that may gain or lose financially through publication of this manuscript.

Code availability The code is available at https://github.com/SergioHdezG/RLPhotoFentonOptimization.

Declarations

Conflict of interest The authors have no financial or proprietary interests in any material discussed in this article.

References

1. Packwood D (2017) Bayesian optimization for materials science. Springer
2. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press
3. Haupt RL, Haupt SE (2004) Practical genetic algorithms, 2nd edn. Wiley
4. Charpentier A, Mignon D, Sophie Barbe TS, Juan C, Simonson T, Allouche D (2019) Variable neighborhood search with cost function networks to solve large computational protein design problems. J. Chem. Inf. Model 59(1):127–136
5. Larsson D, Flach C-F (2022) Antibiotic resistance in the environment. Nat Rev Microbiol 20(5):257–269
6. Giannakis S, Le T-TM, Entenza JM, Pulgarin C (2018) Solar photo-fenton disinfection of 11 antibiotic-resistant bacteria (ARB) and elimination of representative AR genes. Evidence that antibiotic resistance does not imply resistance to oxidative treatment. Water Res 143:334–345. https://doi.org/10.1016/j.watres.2018.06.062
7. García-Fernández I, Polo-López MI, Oller I, Fernández-Ibáñez P (2012) Bacteria and fungi inactivation using fe3+/sunlight, h2o2/sunlight and near neutral photo-fenton: a comparative study. Appl Catal B 121–122:20–29. https://doi.org/10.1016/j.apcatb.2012.03.012
8. Spuhler D, Andrés Rengifo-Herrera J, Pulgarin C (2010) The effect of fe2+, fe3+, h2o2 and the photo-fenton reagent at near neutral ph on the solar disinfection (sodis) at low temperatures of water containing escherichia coli k12. Appl Catal B 96(1):126–141. https://doi.org/10.1016/j.apcatb.2010.02.010
9. Ortega-Gómez E, Martín MMB, García BE, Pérez JAS, Ibáñez PF (2016) Wastewater disinfection by neutral pH photo-fenton: the role of solar radiation intensity. Appl Catal B 181:1–6. https://doi.org/10.1016/j.apcatb.2015.06.059
10. Rodríguez-Chueca J, Polo-López MI, Mosteo R, Ormad MP, Fernández-Ibáñez P (2014) Disinfection of real and simulated urban wastewater effluents using a mild solar photo-fenton. Appl Catal B 150–151:619–629. https://doi.org/10.1016/j.apcatb.2013.12.027
11. Casado C, Moreno-SanSegundo J, De la Obra I, Esteban García B, Sánchez Pérez JA, Marugán J (2021) Mechanistic modelling of wastewater disinfection by the photo-fenton process at circumneutral pH. Chem Eng J 403:126335. https://doi.org/10.1016/j.cej.2020.126335
12. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. ArXiv:1707.06347
13. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI gym. ArXiv:1606.01540
14. Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D (2018) Deep reinforcement learning that matters. In: Proc. of the 32d AAAI conference on artificial intelligence, pp 3207–3214 . https://doi.org/10.1609/aaai.v32i1.11694
15. Andrychowicz M, Denil M, Colmenarejo SG, Hoffman MW, Pfau D, Schaul T, Shillingford B, de Freitas N (2016) Learning to learn by gradient descent by gradient descent. In: Proc. of the 30th int. conf. on neural information processing systems. NeurIPS, pp 3988–3996 . https://doi.org/10.5555/3157382.3157543
16. Li K, Malik J (2017) Learning to optimize. In: 5th int. conf. on learning representations. ICLR
17. Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. In: 5th int. conf. on learning representations. ICLR
18. Barrett TD, Clements WR, Foerster JN, Lvovsky A (2020) Exploratory combinatorial optimization with reinforcement learning. In: The 34th AAAI conf. on artificial intelligence, pp 3243–3250
19. Oh D-H, Adams D, Vo ND, Gbadago DQ, Lee C-H, Oh M (2021) Actor-critic reinforcement learning to estimate the optimal operating conditions of the hydrocracking process. Comput Chem Eng 149:107280. https://doi.org/10.1016/j.compchemeng.2021.107280
20. Petsagkourakis P, Sandoval IO, Bradford E, Zhang D, del Rio-Chanona EA (2020) Reinforcement learning for batch bioprocess optimization. Comput Chem Eng 133:106649. https://doi.org/10.1016/j.compchemeng.2019.106649
21. Ma Y, Noreña-Caro DA, Adams AJ, Brentzel TB, Romagnoli JA, Benton MG (2020) Machine-learning-based simulation and fed-batch control of cyanobacterial-phycocyanin production in plectonema by artificial neural network and deep reinforcement learning. Comput Chem Eng 142:107016. https://doi.org/10.1016/j.compchemeng.2020.107016
22. Hasan MM, Lwin K, Imani M, Shabut A, Bittencourt LF, Hossain MA (2019) Dynamic multi-objective optimisation using deep reinforcement learning: benchmark, algorithm and an application to identify vulnerable zones based on water quality. Eng Appl Artif Intell 86:107–135. https://doi.org/10.1016/j.engappai.2019.08.014
23. Fribourg G, Bréchet Y, Deschamps A, Simar A (2011) Microstructure-based modelling of isotropic and kinematic strain hardening in a precipitation-hardened aluminium alloy. Acta Mater 59(9):3621–3635. https://doi.org/10.1016/j.actamat.2011.02.035
24. Imani M, Ghoreishi SF (2021) Scalable inverse reinforcement learning through multifidelity bayesian optimization. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2021.3051012
25. Jafari R, Javidi MM (2020) Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning. SN Appl Sci 2(2):1–13
26. Mocanu E, Mocanu DC, Nguyen PH, Liotta A, Webber ME, Gibescu M, Slootweg JG (2019) On-line building energy optimization using deep reinforcement learning. IEEE Trans Smart Grid 10(4):3698–3708. https://doi.org/10.1109/TSG.2018.2834219
27. Adams D, Oh D-H, Kim D-W, Lee C-H, Oh M (2021) Deep reinforcement learning optimization framework for a power generation plant considering performance and environmental

issues. J Clean Prod 291:125915. https://doi.org/10.1016/j.jclepro.2021.125915

28. Nian R, Liu J, Huang B (2020) A review on reinforcement learning: introduction and applications in industrial process control. Comput Chem Eng 139:106886. https://doi.org/10.1016/j.compchemeng.2020.106886
29. Audino F, Conte LO, Schenone AV, Pérez-Moya M, Graells M, Alfano OM (2019) A kinetic study for the fenton and photo-fenton paracetamol degradation in an annular photoreactor. Environ Sci Pollut Res 26(5):1614–7499. https://doi.org/10.1007/s11356-018-3098-4
30. Afolabi IC, Popoola SI, Bello OS (2020) Modeling pseudo-second-order kinetics of orange peel-paracetamol adsorption process using artificial neural network. Chemom Intell Lab Syst 203:104053. https://doi.org/10.1016/j.chemolab.2020.104053
31. Gholizadeh AM, Zarei M, Ebratkhahan M, Hasanzadeh A (2021) Phenazopyridine degradation by electro-fenton process with magnetite nanoparticles-activated carbon cathode, artificial neural networks modeling. J Environ Chem Eng 9(1):104999. https://doi.org/10.1016/j.jece.2020.104999
32. Oladipo AA, Abureesh MA, Gazi M (2016) Bifunctional composite from spent "cyprus coffee" for tetracycline removal and phenol degradation: solar-fenton process and artificial neural network. Int J Biol Macromol 90:89–99. https://doi.org/10.1016/j.ijbiomac.2015.08.054. Special Issue on Biomedical Engineering
33. Gazi M, Oladipo AA, Ojoro ZE, Gulcan HO (2017) High-performance nanocatalyst for adsorptive and photo-assisted fenton-like degradation of phenol: modeling using artificial neural networks. Chem Eng Commun 204(7):729–738. https://doi.org/10.1080/00986445.2017.1311253
34. Durán A, Monteagudo JM, Mohedano M (2006) Neural networks simulation of photo-fenton degradation of reactive blue 4. Appl Catal B 65(1):127–134. https://doi.org/10.1016/j.apcatb.2006.01.004
35. Sutton RS, Barto AG (2018) Reinforcement learning, second edition: an introduction. The MIT Press, Cambridge
36. François-Lavet V, Henderson P, Islam R, Bellemare MG, Pineau J (2018) An introduction to deep reinforcement learning. Found. Trends Mach. Learn. 11(3–4):219–354. https://doi.org/10.1561/2200000071
37. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
38. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proc. of the AAAI conf. on artificial intelligence, vol 30
39. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: Int. conf. on machine learning. PMLR, pp 1995–2003
40. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: Int. conf. on machine learning. PMLR, pp 387–395
41. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8(3–4):229–256
42. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: Proc. of The 33rd int. conf. on machine learning, vol 48. PMLR, pp 1928–1937
43. Liu M, Chen L, Du X, Jin L, Shang M (2021) Activated gradients for deep neural networks. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2021.3106044
44. Schaul T, Quan J, Antonoglou I, Silver D (2016) Prioritized experience replay. In: 4th int. conf. on learning representations. ICLR
45. Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Pieter Abbeel O, Zaremba W (2017) Hindsight experience replay. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. NeurIPS
46. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. In: 4th international conference on learning representations. ICLR
47. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: International conference on machine learning. PMLR, pp 1889–1897
48. Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th international conference on machine learning, vol 80. PMLR, pp 1861–1870
49. Andrychowicz M, Raichuk A, Stańczyk P, Orsini M, Girgin S, Marinier R, Hussenot L, Geist M, Pietquin O, Michalski M, Gelly S, Bachem O (2021) What matters for on-policy deep actor-critic methods? A large-scale study. In: International conference on learning representations
50. Engstrom L, Ilyas A, Santurkar S, Tsipras D, Janoos F, Rudolph L, Madry A (2019)Implementation matters in deep rl: a case study on ppo and trpo. In: International conference on learning representations
51. Gao F, Han L (2012) Implementing the nelder-mead simplex algorithm with adaptive parameters. Comput Optim Appl 51(1):259–277