Reinforcement Learning for Batch Bioprocess Optimization

P. Petsagkourakis[a,b], I.O. Sandoval[c], E. Bradford[d], D. Zhang[a,e,*], E.A. del Rio-Chanona[e,**]

[a] *School of Chemical Engineering and Analytical Science,The University of Manchester, M13 9PL, UK*
[b] *Centre for Process Systems Engineering (CPSE), Department of Chemical Engineering, University College London, Torrington Place, London WC1E 7JE, United Kingdom*
[c] *Instituto de Ciencias Nucleares, Universidad Nacional Autónoma de México, A.P. 70-543, C.P. 04510 Ciudad de México, Mexico*
[d] *Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway*
[e] *Centre for Process Systems Engineering (CPSE), Department of Chemical Engineering, Imperial College London, UK*

Abstract

Bioprocesses have received a lot of attention to produce clean and sustainable alternatives to fossil-based materials. However, they are generally difficult to optimize due to their unsteady-state operation modes and stochastic behaviours. Furthermore, biological systems are highly complex, therefore plant-model mismatch is often present. To address the aforementioned challenges we propose a Reinforcement learning based optimization strategy for batch processes.

In this work we applied the Policy Gradient method from batch-to-batch to update a control policy parametrized by a recurrent neural network. We assume that a preliminary process model is available, which is exploited to obtain a preliminary optimal control policy. Subsequently, this policy is updated based on measurements from the *true* plant. The capabilities of our proposed approach were tested on three case studies (one of which is nonsmooth) using a more complex process model for the *true* system embedded with adequate process disturbance. Lastly, we discussed advantages and disadvantages of this strategy compared against current existing approaches such as nonlinear model predictive control.

Keywords: Machine Learning, Batch optimization, Recurrent Neural Networks, Bioprocesses, Policy Gradient, Uncertain dynamic systems, nonsmooth

1. Introduction

The synthesis of sustainable bioproducts is a promising research field of international interest to replace a broad range of chemicals derived from fossil synthetic routes [11, 21]. Biochemical processes

*Corresponding author
**Corresponding author

Email addresses: dongda.zhang@manchester.ac.uk (D. Zhang), a.del-rio-chanona@imperial.ac.uk (E.A. del Rio-Chanona)

employ microorganisms to produce platform chemicals and high-value products from renewable resources [22]. Biosystems are considerably more complex than traditional chemical processes due to the convoluted relationship between metabolic reaction networks and culture fluid dynamics [16]. In addition, biological metabolic pathways are highly sensitive to changes of the process operating conditions. Therefore, bioprocesses display stochastic behaviour in the macro-scale [57, 50]. Consequently, the development of a physics-based model to accurately represent large-scale bioprocesses is challenging. For these reasons the control and optimization of biosystems is still an open research question. To address this problem we propose a data-driven approach, which avoids the critical limitations of mechanistic models.

An efficient optimization approach for a bioprocess needs to subsequently be able to handle both the inherent stochasticity of the system (e.g. process disturbances) and plant-model mismatches. To accomplish this we exploit a method from *Reinforcement learning* (RL) called *Policy Gradients* as an alternative to currently utilised approaches. RL has been shown to be a powerful control approach, which is one of the few control techniques able to handle nonlinear stochastic optimal control problems [6]. Solution methods for dynamic optimization problems exploiting RL have been divided into two categories.

The first category is based on *Dynamic Programming* (DP), hence termed Approximate Dynamic Programming (ADP). DP relies on the Hamilton-Jacobi-Bellman equation (HJBE), the solution of which becomes intractable for small size problems with nonlinear dynamics and continuous state and control actions. Because of this, past research has relied on using ADP techniques to find approximate solutions to these problems [47].

The second category is to use *Policy Gradients*, which directly obtains a policy by maximizing a desired performance index. This approach is particularly well suited to deal with problems where both the state space and the control space are continuous. Given the advantages that Policy Gradients can offer when confronted with bioprocess optimization, we have adopted this approach in the current work. Policy Gradient methods, along with their benefits, are further explained in Section 2.2.

1.1. Related Work

Given that chemical engineers have always dealt with complex and uncertain systems there have been several approaches that address specific instances of the aforementioned problems, we highlight some related previous work in the sections below.

One example to track stochastic batch-to-batch systems is Iterative Learning Control (ILC) which was initially introduced for robot manipulators [2], and later implemented by the process control community [55]. ILC deals with the problem of tracking the control performance in batch processes given a reference trajectory for runs that last a fixed time, and where the process state is reset to the same value at the start of each run. An overview of ILC strategies in process control can be found in [28].

Real-time optimization (RTO) is another method that deals with uncertain processes. The main idea is to represent the process dynamics by a nonlinear input/output mapping where the disturbances are explicitly accounted for. This mapping is then used to optimize some desired performance index. For the interested reader, further details can be found in [8] and [12]. A recent review on this topic can be found in [32]. For the dynamic systems, these methodologies are usually referred to as Dynamic Real-time optimization (DRTO) which is closely related to NMPC. More details can be found in [41, 42].

Another technique that deals with stochastic systems is model predictive control (MPC), and its extension to nonlinear systems, NMPC. NMPC has a vast variety of methods that can incorporate

uncertainty or maintain properties under the presence of stochastic environments. The most common paradigms are the *stochastic* NMPC [33] and the *Robust* NMPC [4], where the former incorporates the uncertainty by minimizing (usually) the expectation of the objective function, whilst the latter approach solves a min-max optimization by minimizing the worst case scenario of the uncertainty. Both of these approaches require knowledge regarding the nature of the uncertainty in order to proceed.

There are different approaches that have been proposed for NMPC frameworks, including scenario [5] based multi-stage schemes for nonlinear systems [31, 25], where stochastic programming is utilized and future information is incorporated in an adaptive manner. Another approach is the use of Gaussian processes [9, 10] or using (generalized) polynomial chaos expansions [23] to model effectively the uncertainties of the process. In the case where no proper information for the uncertainty is available, e.g. there is not enough data to conduct uncertainty quantification, optimal control is explored using the nominal linear or nonlinear available model. In terms of solution procedures for the dynamic optimization problem, it is common to use a direct approach after parametrizing and discretizing the control inputs [51] or the system dynamics [7] resulting in a nonlinear programming problem. Although much less common, indirect approaches can also be used, where the necessary conditions of optimality are solved explicitly [3]. If no information on structural information is known, conservative assumptions can be made in order to establish stability conditions [19, 39, 38].

Reinforcement Learning (in an Approximate Dynamic Programming philosophy), has lately caught significant attention for chemical process control. For example, in [30] a model-based strategy and a model-free strategy for control of nonlinear processes were proposed, in [37] ADP strategies were used to address fed-batch reactor optimization, in [27] mixed-integer decision problems were addressed with applications to scheduling. In [49] with the inclusion of distributed optimization techniques, an input-constrained optimal control problem solution technique was presented, among other works (e.g. [13], [44]). All these approaches rely on the (approximate) solution of the HJBE, and have been shown to be reliable and robust for several problem instances.

In this paper, we present another take on RL, that of using Policy Gradients. Policy Gradient methods directly estimate the control policy, without the need of a model, or the solution of the HJBE, its advantages are highlighted in the following section.

In addition to the above, for recent reviews of Machine Learning and Artificial Intelligence applied to chemical engineering the reader is referred to [29] and [52]. A shorter review focused on modelling bioprocesses with ML tools can be found in [17].

1.2. Motivation

The process systems engineering community has been dealing with stochastic batch-to-batch systems for a long time. For example, nonlinear dynamic optimization and particularly NMPC are a powerful methodology to address uncertain dynamic systems, however there are several properties that make its application less attractive. All the approaches in NMPC require the knowledge of a detailed model that describe the system dynamics, and stochastic NMPC additionally requires an assumption for the uncertainty quantification/propagation. Furthermore, the online computational time may be a bottleneck for real time applications since a (possibly) nonlinear optimization problem has to be solved.

In contrast, RL directly accounts for the effect of future uncertainty and its feedback in a proper closed-loop manner, whereas conventional NMPC assumes open-loop control actions at future time points in the prediction, which can lead to overly conservative control actions [30]. In addition, policy gradients can establish a policy in a model-free fashion and excel at on-line computational time. This

is because the online computations require only evaluation of a policy, since all the computational cost is shifted off-line.

As mentioned previously, Real-time optimization (RTO) has been used to address many instances of batch-to-batch problems. Interestingly, some recent approaches have suggested a hybrid modeling strategy, where function approximates are used in conjunction with a pre-existing model [14, 20]. From some perspectives these recent algorithms could be thought of as model-based Reinforcement learning approaches. However, there is not yet a clear consensus on how to address problems with plant-model mismatch, measurement noise, and disturbances in an RTO framework.

In terms of previous RL approaches in chemical engineering to address process control and optimization, they have relied on action-value methods (e.g. Q-learning, solution of the HJBE). However, to address continuous nonlinear action domains Policy Gradient methods present several advantages:

- In Policy Gradient methods, the approximate policy can naturally approach a deterministic policy, whereas action-value methods (that use epsilon-greedy or Boltzmann functions) select a random control action with some heuristic rule [47].
- Although it is possible to estimate the objective value of state-action pairs in continuous action spaces by function approximators, this does not help choose a control action. Therefore, online optimization over the action space for each time-step should be performed, which can be slow and inefficient. Policy Gradient methods work directly with policies that emit probability distributions, which is much faster and does not require an online optimization step.
- Policy Gradient methods are guaranteed to converge at least to a locally optimal policy even in high dimensional continuous state and action spaces, unlike action-value methods where convergence to local optima is not guaranteed [47].
- Policy Gradient methods enable the selection of control actions with arbitrary probabilities. In such cases, the best approximate policy may be stochastic [47].

Due to the above advantages, in this work we propose an optimization strategy that uses a Policy Gradient algorithm to optimize batch-to-batch bioprocesses. This work extends our proposed methodology in [40], the new approach presents a much faster adaptation time by implementing *transfer learning* for the efficient adaptation of the policies. Additional more complex case studies and a comparison against NMPC are included. Furthermore, we exemplify both approaches (NMPC and our approach) in a system described by a nonsmooth differential equation model. The difficulty for nonsmooth models is highlighted in [45].

2. Methodology

2.1. Problem Statement

In this work, we assume that the system's dynamics are given by an (generally) unknown probability distribution, following a Markov process:

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \tag{1}$$

This system can be approximated by the following discrete time stochastic nonlinear system represented as a state-space model:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{d}_t) \tag{2}$$

where t represents the discrete time, $\mathbf{x}_t \in \mathbb{R}^{n_x}$ is the vector of states, $\mathbf{u}_t \in \mathbb{R}^{n_u}$ is the vector of inputs, $\mathbf{d}_t \in \mathbb{R}^{n_d}$ is the the vector of process disturbances, and $f(\cdot)$ are the nonlinear dynamics of the physical system.

Our strategy seeks to find the optimal policy for a batch process under the presence of disturbances and measurement noise. Then, the problem can be written as an Optimal Control Problem (OCP):

$$\mathcal{P}(\pi(\cdot)) := \begin{cases} \max\limits_{\pi(\cdot)} \mathbb{E}[J(\mathbf{x}_t^k, \mathbf{u}_t^k)] \\ \text{s.t.} \\ \mathbf{x}_0^k = \mathbf{x}^k(0) \\ \mathbf{x}_{t+1}^k = f(\mathbf{x}_t^k, \mathbf{u}_t^k, \mathbf{d}_t^k) \quad \forall t \in \{1, ..., T-1\} \\ \mathbf{u}_t \sim \pi(\mathbf{x}_t^k) \\ \mathbf{u} \in \mathbb{U} \\ \text{given} \\ \mathbf{x}_t^j \quad \forall j \in \{0, ..., k-1\} \quad \forall t \in \{1, ..., T\} \end{cases} \tag{3}$$

The objective is to maximize the expectation of an economic criterion J, where k is the current batch, while j refers to previous batch realizations. Additionally, the optimization problem (3) searches for a set of functions $\pi(\cdot)$ that maps the probability of u_t^k given x_t^k. Notice that in problem (3) we make no assumptions about the nature of $\mathbf{d}$. Even in the case where the dynamics are fully known, the solution of problem (3) may be intractable for medium size systems.

To overcome this limitation a novel strategy is proposed, where a policy $\pi_\theta(\cdot)$, parametrized by the parameters θ, is constructed that maximizes the expectation of a performance index J. The states at the time $t+1$ are assumed to be given by the probability density $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. The interaction with the policy can be depicted as a closed-loop, see Fig. 1.

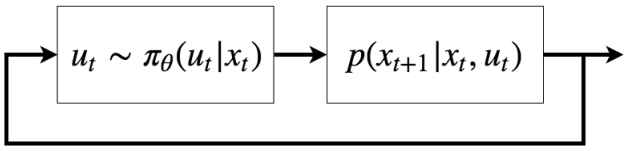


Figure 1: Representation of interaction between policy and the physical system

Let $\boldsymbol{\tau}$ denote a joint random variable of states and controls defining a trajectory with a time horizon T: $\boldsymbol{\tau} = (\mathbf{x}_0, \mathbf{u}_0, R_0, ..., \mathbf{x}_{T-1}, \mathbf{u}_{T-1}, R_{T-1}, \mathbf{x}_T, R_T)$, the performance index being

$$J(\boldsymbol{\tau}) = \sum_{t=0}^{T} \gamma^t R_t(\mathbf{u}_t, \mathbf{x}_t) \tag{4}$$

where $\gamma \in (0, 1]$ is the *discount factor* and R_t a given reward at the time instance t for the values of u_t, x_t. We represent the probability density of a trajectory as:

$$p(\boldsymbol{\tau}|\theta) = \hat{\mu}(\mathbf{x}_0) \prod_{t=0}^{T-1} [\pi(\mathbf{u}_t|\mathbf{x}_t, \theta) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)] \tag{5}$$

where $\hat{\mu}(\mathbf{x}_0)$ is the probability density of the initial state. We can therefore state the following optimization problem:

$$\max_{\pi(\cdot)} \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\theta)}[J(\boldsymbol{\tau})] \tag{6}$$

notice that the process dynamics are implicit in $\boldsymbol{\tau}$. To solve problem (6) we turn our attention to policy gradient methods.

2.2. Policy Gradient Methods

Policy gradient methods compute a policy that maximizes the expectation over some objective function (*i.e.* problem (6)). They rely on a parametrized policy function $\pi_\theta(\cdot)$ that returns an action $\mathbf{u}$ given a state of the system $\mathbf{x}$ and a set of intrinsic parameters θ of the policy. In the case of stochastic policies, the policy function returns the defining parameters of a probability distribution over all possible actions, from which the actions are sampled:

$$\mathbf{u} \sim \pi_\theta(\mathbf{u}|\mathbf{x}) = \pi(\mathbf{u}|\mathbf{x},\theta) = p(\mathbf{u}_t = \mathbf{u}|\mathbf{x}_t = \mathbf{x}, \theta_t = \theta). \tag{7}$$

In this work, a Recurrent neural network (RNN) is used as the parametrized policy, which takes (a number of past) states and control actions as inputs and returns the moments of a probability distribution. Then the next control action is drawn from the corresponding probability distribution. For example, if the control actions live in a normal distribution then a mean and a variance are computed, from these mean and variance a control action can be drawn. In this setting, the exploitation-exploration trade-off is represented explicitly by the value of the variance of the underlying distribution of the policy. Deterministic policies can be seen as a limiting case where the variance converges to zero.

To learn the optimal policy, we seek to maximize our performance metric, and hence we can follow a gradient ascent strategy:

$$\theta_{m+1} = \theta_m + \alpha_m \nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\theta)}[J(\boldsymbol{\tau})] \tag{8}$$

where m is the current iteration that the parameters are updated (epoch), $\mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\theta)}[J(\boldsymbol{\tau})]$ is the expectation of J over $\boldsymbol{\tau}$ and α_m is the step size (also termed learning rate in the RL community) for the m^{th} iteration. Computing $\hat{J}(\theta) = \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\theta)}[J(\boldsymbol{\tau})]$ directly is difficult, therefore we use the *Policy Gradient Theorem* [48], which shows the following:

$$\begin{aligned}
\hat{J}(\theta) = \nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\theta)}[J(\boldsymbol{\tau})] &= \nabla_\theta \int p(\boldsymbol{\tau}|\theta)\ J(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && \text{(9a)}\\
&= \int \nabla_\theta p(\boldsymbol{\tau}|\theta)\ J(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && \text{(9b)}\\
&= \int p(\boldsymbol{\tau}|\theta)\ \frac{\nabla_\theta p(\boldsymbol{\tau}|\theta)}{p(\boldsymbol{\tau}|\theta)}\ J(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && \text{(9c)}\\
&= \int p(\boldsymbol{\tau}|\theta)\ \nabla_\theta \log\left(p(\boldsymbol{\tau}|\theta)\right) J(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} && \text{(9d)}\\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[J(\boldsymbol{\tau}) \nabla_\theta \log\left(p(\boldsymbol{\tau}|\theta)\right)\right] && \text{(9e)}
\end{aligned}$$

Notice from (9b) that, $p(\boldsymbol{\tau}|\theta)\ J(\boldsymbol{\tau})$ is an objective function value multiplied by its probability density, therefore, integrating this over all possible values of $\boldsymbol{\tau}$ we obtain the expected value. From there we

arrive at (9e), where, dropping the explicit distribution of $\boldsymbol{\tau}$, gives us an unbiased gradient estimator, (8) now becomes:

$$\theta_{m+1} = \theta_m + \alpha_m \mathbb{E}_{\boldsymbol{\tau}} \left[J(\boldsymbol{\tau}) \nabla_\theta \log \left(p(\boldsymbol{\tau}|\theta) \right) \right] \tag{10}$$

Using the expression for $p(\boldsymbol{\tau}|\theta)$ in (5) and taking its logarithm, we obtain:

$$\nabla_\theta \log \left(p(\boldsymbol{\tau}|\theta) \right) = \nabla_\theta \sum_{t=0}^{T-1} \log \left(\pi(\mathbf{u}_t|\mathbf{x}_t, \theta) \right) \tag{11}$$

Note that since $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ and $\hat{\mu}(\mathbf{x}_0)$ are independent of θ they disappear from the above expression. Then we can rewrite (9e) for a trajectory as:

$$\nabla_\theta \mathbb{E}_{\boldsymbol{\tau}}[J(\boldsymbol{\tau})] = \mathbb{E}_{\boldsymbol{\tau}} \left[J(\boldsymbol{\tau}) \nabla_\theta \sum_{t=0}^{T-1} \log \left(\pi(\mathbf{u}_t|\mathbf{x}_t, \theta) \right) \right] \tag{12}$$

Notice that expression (12) does not require the knowledge of of the dynamics of the physical system. However, the above update presents two challenges: the selection of the policy $\pi(\mathbf{u}_t|\mathbf{x}_t, \theta)$ and the computation of the expectation. To address these possible issues, in this work, recurrent neural networks are used to parametrize the policy of the policy gradient (presented in Section 2.3), while a Monte-Carlo method is utilized to approximate the expectation (presented in Section 2.4).

2.3. Recurrent Neural Network

Recurrent neural networks (RNNs) were developed to efficiently represent sequential data, which are a type of artifical neural network tailored to this task. RNNs produce an output at each time step and have recursive connections between hidden units. This allows them to fully account for previous data and hence are ideal to simulate time-series. In general, RNNs can be depicted as a folded computational graph as presented in Fig. 2.

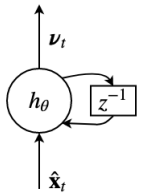


Figure 2: Computational graph of recurrent neural network

Fig. 2 shows how an input $\hat{\mathbf{x}}_t$ is presented to the network as well as the recursive state of RNN, $\mathbf{u}_{t-1}$ and outputs $\boldsymbol{\nu}_t$. A more detailed representation of an RNN is depicted in Fig. 3, which is equivalent to a series of unfolded nodes associated with a particular time instance.

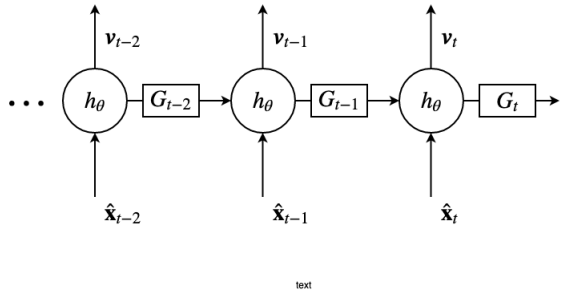


Figure 3: Recurrent neural networks as unfolded computational graph, with one step delay (z^{-1})

In Fig. 3 we can appreciate that each node receives two inputs $\mathbf{x}_{t-i}$ and $\mathbf{u}_{t-i-1}$. Generally speaking, the input $\mathbf{x}_{t-i}$ corresponds to the data supplied to that node, such as in a traditional artificial neural network (also referred to as feedforward). The unfolded computational graph in this case can be represented as a dynamic system:

$$\begin{aligned} \boldsymbol{\nu}_t &= h_\theta(\hat{\mathbf{x}}_t, \mathbf{u}_{t-1}) \\ \mathbf{u}_t &= G_t(\boldsymbol{\nu}_t) \end{aligned} \tag{13}$$

where $\hat{\mathbf{x}}_t$ is the vector that contains all the external variables for the RNN, G the function that computes the output of the network u, and h_θ represents the layers of the neural networks. Deep structures (which means having more than one hidden layer) can be employed to enhance the performance of the network (deep neural networks) which have been combined with Reinforcement learning recently in [35, 36]. Previous realizations of the states x and the controls u are also used as input variables to the network, e.g. $\hat{\mathbf{x}}_t = \left[\mathbf{x}_t^T, \ldots, \mathbf{x}_{t-N}^T, \mathbf{u}_{t-2}^T, \ldots, \mathbf{u}_{t-N-1}^T\right]^T$, to model dynamic systems. RNNs have previously been applied either as a surrogate model of the process dynamics [46] or as a parametrization of the agent (control policy) [34].

In this work, RNNs are applied to parameterize the stochastic policy. We must remark that in theory the Markov decision process does not require RNNs (due to the Markov property), however in practice the use of RNNs can improve the performance of the policy by exploiting additional memory that is provided. In the current work the RNN initially computes the mean and the variance of a multivariate normal distribution where the control actions live. Subsequently, the *actual* control action is drawn. Precisely, $\boldsymbol{\nu}_t = [\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t]$ where the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean and variance, respectively, and $\mathbf{u}_t = G_t(\boldsymbol{\nu}_t)$ is substituted by $\mathbf{u}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ making it a stochastic policy. Under the presence of uncertainty (stochastic in nature) a deterministic policy will fail as the control action will always be the same for the same states since it learns a deterministic mapping from states to control actions at the exact same state. On the contrary, a stochastic policy draws a control action from a probability distribution which can account for stochastic environments.

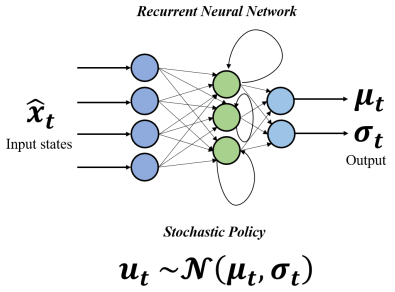


Figure 4: Graphical representation of a stochastic policy network

In Figure 4, a schematic representation of a policy network where the stochastic policy follows a Gaussian distribution is depicted. In this figure, we can observe how the states $\hat{x}_t$ are used as input to the network, and how the network computes the mean μ_{t+1} and standard deviation σ_{t+1} for the subsequent time step. Then, the control action $\mathbf{u}_{t+1}$ is drawn from the distribution defined by the outputs of the network.

2.4. Reinforce Algorithm

Given that the parametrized policy used in this work is a RNN, it must be trained to adjust its weights so that the output corresponds to an optimal control action. To this end, we use the steepest ascent strategy mentioned in (8). However, computing the expectation in (12) can be an intractable problem, and this expression is needed to compute the steepest ascent update (10). Therefore, we propose to use the Reinforce algorithm [54] approximates the gradient of the policy to maximize the expected reward with respect to the parameters θ without the need of a dynamic model of the process. To compute the expectation we take several sample trajectories and then approximately calculate $\mathbb{E}_{\boldsymbol{\tau}}\left[J(\boldsymbol{\tau})\nabla_\theta \sum_{t=0}^{T-1} \log\left(\pi(\mathbf{u}_t|\mathbf{x}_t,\theta)\right)\right]$ as an average of K samples:

$$\nabla_\theta \mathbb{E}_{\boldsymbol{\tau}} \approx \frac{1}{K}\sum_{k=1}^{K}\left[J(\boldsymbol{\tau}^k)\nabla_\theta \sum_{t=0}^{T-1} \log\left(\pi(\mathbf{u}_t^k|\hat{\mathbf{x}}_t^k,\theta)\right)\right] \tag{14}$$

where we denote the sample k as a super-index. The variance of this estimation can be reduced with the aid of an action-independent baseline b, which does not introduce a bias [47]. A simple but effective baseline is the expectation of reward under the current policy, approximated by the mean over the sampled paths:

$$b = \hat{J}(\theta) \approx \frac{1}{K}\sum_{k=1}^{K} J(\boldsymbol{\tau}^{(k)}), \tag{15}$$

which leads to:

$$\nabla_\theta \hat{J}(\theta) \approx \frac{1}{K}\sum_{k=1}^{K}\left[(J(\boldsymbol{\tau}^k) - b)\nabla_\theta \sum_{t=0}^{T-1} \log\left(\pi(\mathbf{u}_t^k|\hat{\mathbf{x}}_t^k,\theta)\right)\right] \tag{16}$$

This selection increases the log likelihood of an action by comparing it to the expected reward of the current policy. (16) is the gradient that we can now incorporate into our steepest ascent strategy. The algorithm that trains the RNN and obtains the optimal policy network is the following.

Algorithm 1 Policy Gradient Algorithm

Input: Initialize policy parameter $\theta = \theta_0$, with $\theta_0 \in \Theta_0$, learning rate, its update rule α, $m := 0$, the number of episodes K and the number of epochs N.
Output: policy $\pi(\cdot|\cdot, \theta)$ and Θ

for m = 1,..., N **do**

1. Collect $\mathbf{u}_t^k, \mathbf{x}_t^k$ for T time steps for K trajectories along with $J(\mathbf{x}_T^k)$, also for K trajectories.
2. Update the policy, using a policy gradient estimate $\theta_{m+1} = \theta_m + \alpha_m \frac{1}{K} \sum_{k=1}^{K} \left[(J(\boldsymbol{\tau}^k) - b) \nabla_\theta \sum_{t=0}^{T-1} \log \left(\pi(\mathbf{u}_t^k | \hat{\mathbf{x}}_t^k, \theta) \right) \right]$
3. $m := m + 1$

The steps in the Algorithm 1 are explained below.

Initialization: The RNN policy network and its weights θ are initialized, along with the algorithm's hyperparameters such as learning rate, number of episodes and number of epochs.

Training loop: The weights on the RNN are updated by a policy gradient scheme for a total of N epochs. In **Step 1** K trajectories are computed, each trajectory consists of T time steps, and states and control actions are collected. In **Step 2** the weights of the RNN are updated based on the policy gradient framework. In **Step 3**, either the algorithm terminates or returns to Step 1.

2.5. Reinforcement Learning for Bioprocess Optimization under Uncertainty

The methodology presented aims to overcome plant-model mismatch in uncertain dynamic systems, a usual scenario in bioprocesses. It is common to construct simple deterministic models according to a hypothesized mechanism, however the real system is more complex and presents disturbances. We propose the following methodology to address this problem (following from Algorithm 2).

Step 0, Initialization: The algorithm is initialized by considering an initial policy network (e.g. RNN policy network) with untrained parameters θ_0.

Step 1, Preliminary Learning (Off-line): It is assumed that a preliminary mechanistic model can be constructed from previous existing process data, hence, the policy learns this preliminary mechanistic model. This is done by running Algorithm 1 in a simulated environment by the mechanistic model. This allows the policy to incorporate previously believed knowledge about the system. The policy will therefore end with an optimal control policy for the mechanistic model. The termination criteria can be defined either by the designer or by the difference from the solution of the OCP, since the process model is known.

Given that the experiments are in silico, a large number of episodes and trajectories can be generated that corresponds to different actions from the probability distribution of $\mathbf{u}_t$, and a specific set of parameters of the RNN, respectively. The resulting control policy is a good approximation of the optimal policy. Notice that if a stochastic preliminary model exists, this approach can immediately exploit it, contrary to traditional NMPC approaches. This finishes the in silico part of the algorithm, subsequent steps would be run in the true system. Therefore, emphasis after this step is given on sampling as least as possible, as every new sample would result in a new batch run from the real plant.

Step 2-3, Transfer Learning: The policy could directly be retrained using the true system and adapt all the weights according to the Reinforce algorithm. However, this may result in undesired effects. The control policy proposed in this work has a deep structure, as a result a large number of weights could be present. Thus, the optimization to update the policy may easily be stuck in a low-quality local optima or completely diverge. To overcome this issue the concept of transfer learning

is adopted. In transfer learning, a subset of training parameters is kept constant to avoid the use of a large number of epochs and episodes, applying knowledge that has been stored in a different but related problem. This technique is originated from the task of image classification, where several examples exists, *e.g.* in [26], [43], [18].

Using transfer learning, the current work only retrained the last hidden layers, and the policy is able to adapt to new situation without losing previously obtained knowledge, as shown in Fig.5. Alternatively, an additional set of layers could be added on the top of the network.

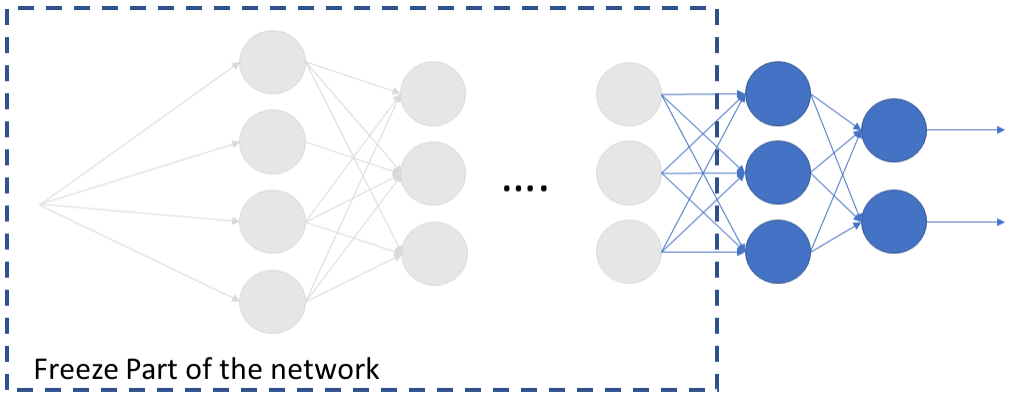


Figure 5: Part of the network is kept frozen to adapt to new situations more efficiently

Step 4, Transfer Learning Reinforce (On-line): In this step, Algorithm 1 is applied again, but now, on the true system. This algorithm aims to maximize a given reward (e.g. product concentration, economic objective)

Step 5: Terminate policy update and output Θ that defines the optimal RNN policy.

The methodology is described in Algorithm 2 and depicted in Fig. 6.

Algorithm 2 Batch to batch algorithm

Input: Initialize the set of policy parameter Θ_0, learning rate and its update rule α, epochs $N := N_0$, maximum number of epochs N_{max}, epochs for the *true* system N_{true}, episodes K_0, and episodes for the *true* system K with $K_0 >> K$.

1. **while** $N \leq N_{max}$ **do**:
   (a) Apply Algorithm 1 using an approximate model and get the trained parameters $\hat{\Theta}_0$ using N epochs and T_0 episodes.
   (b) increase N.
2. $\hat{\Theta}_1$:=$\hat{\Theta}_0$ Initial values of the parameters $\hat{\Theta}_1$ are set as those identified in **Step 1**
3. **Transfer Learning:** Pick $\hat{\Theta}_1^* \subset \hat{\Theta}_1$ to be constant
4. For $i = 1, \ldots, N_{true}$ **do**:
   (a) Apply Algorithm 1 on the true system and get the trained parameters $\hat{\Theta}_1$ for one epoch and K episodes.
5. $\Theta = \hat{\Theta}_1$

Output: Preliminary trained policy network with parameters Θ that takes states as inputs (e.g. $\mathbf{x}_t$) and outputs the statistical parameter (e.g. $\boldsymbol{\mu}_{t+1}$, $\boldsymbol{\sigma}_{t+1}$) of an optimal stochastic action.
Note: We denote Θ_0 as the set of parameters of the RNN before any training, $\hat{\Theta}_0$ the set of parameters after the training in **Step 1**. $\hat{\Theta}_1$ denotes the set of parameters passed along to the training by the *true* system, and subsequent set of parameters during **Step 4** as $\hat{\Theta}_i$, where i is the current epoch.

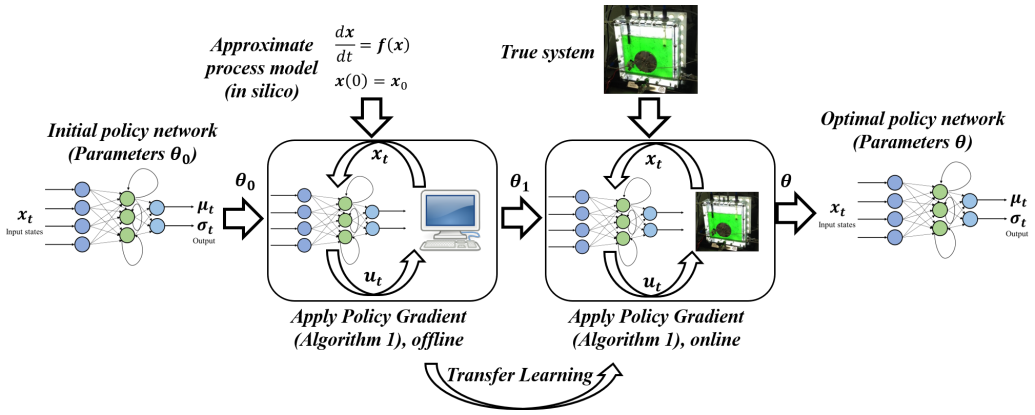


Figure 6: Batch-to-Batch algorithm (Algorithm 2)

3. Computational Case Studies

In this section three case studies are presented to illustrate the effectiveness of the proposed batch-to-batch strategy. Our strategy is applied to 3 fed-batch bioreactors, where the objective is to maximize the concentration of a target product (y_2 or c_q) at the end of the batch time, using light and an inflow rate (u_1 or I and u_2 or F_N) as manipulated variables.

3.1. Case Study 1 - Ordinary Differential Equations

In the first case study, the "real" photo-production system (plant) is described by the following equations plus an additional random disturbance:

$$\frac{dy_1}{dt} = -(u_1 + 0.5\ u_1^2)y_1 + 0.5\frac{u_2 y_2}{(y_1 + y_2)} \tag{17}$$

$$\frac{dy_2}{dt} = u_1\ y_1 - 0.7u_2\ y_1 \tag{18}$$

where u_1, u_2 and y_1, y_2 are the manipulated variables and the outlet concentrations of the reactant and product, respectively. The batch operation time course is normalized to 1. Additionally, a random disturbance is assumed, which is given by a Gaussian distribution with mean value 0 and standard deviation 0.02 on the states y_1 and y_2. We discretize the time horizon into 10 intervals of the dimensionless time, with one constant control input in each interval, resulting in a total of 20 control inputs.

The exact model is usually not known, and a simplified deterministic model is assumed according to some set of parameters. This preliminary model, given in (19 - 20), is utilized in an extensive offline training in order to construct the control policy network. As illustrated in the previous section 2.4, there is a potential to have a close approximation of the solution of the OCP using the RNN-Reinforce.

$$\frac{dy_1}{dt} = -(u_1 + 0.5\ u_1^2)y_1 + u_2 \tag{19}$$

$$\frac{dy_2}{dt} = u_1\ y_1 - u_2\ y_1 \tag{20}$$

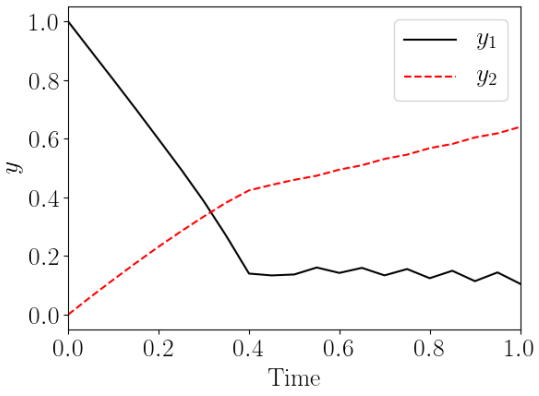
The training consist of 100 epochs and 800 episodes using the simplified model to search the optimal control policy that maximizes the reward of (19 - 20) in this case the concentration of y_2 at the final instance (21).

$$\begin{aligned} R_t &= 0, \quad t \in \{0, T-1\} \\ R_T &= y_2(T). \end{aligned} \tag{21}$$

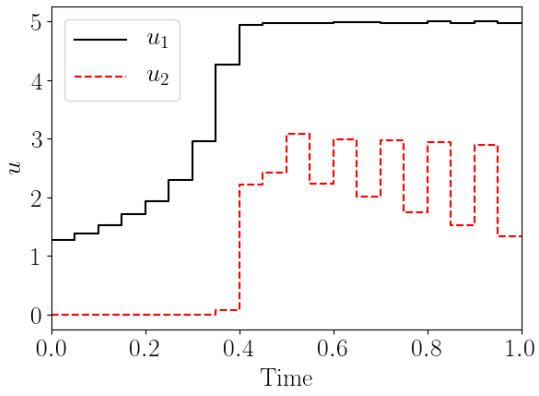
The control actions are constrained to be in the interval $[0, 5]$. The control policy RNN is designed to contain 2 hidden layers, each of which comprises 20 neurons embedded by a hyperbolic tangent activation function. It was observed that 2 hidden layers are sufficient to approximate the optimal control policy, however there is the potential to use deeper structures with more layers for more complex systems. Furthermore, we employed two policy networks instead of one for simplicity. This approach assumes that the two manipulated variables are independent resulting in diagonal variance.

The algorithm is implemented in Pytorch version 0.4.1. Adam [24] is employed to compute the network's parameter values using a step size of 10^{-2} with the rest of hyperparameters at their default values. After the training, using the simplified model the reward has the same value with the one computed by the optimal control problem, as expected. It should be noted that the computation cost of the control action using the policy is insignificant since it only requires the evaluation of the corresponding RNN, and does not depend directly on the complexity or the number of variables. In contrast, the solution of the OCP scale very badly with respect to both the complexity and the number of variables. Precisely, the maximum rewards for RL and OCP for both cases is 0.64. The reward

for its epoch is illustrated in Fig. 8 and the process trajectories after the final update of the policy networks are shown in Fig. 7.



(a) Solution of y for the nominal system



(b) Solution of u for the nominal system

Figure 7: The time trajectories of of the output variables of the approximate model and the piecewise constant control actions associated with the preliminary trained policies.

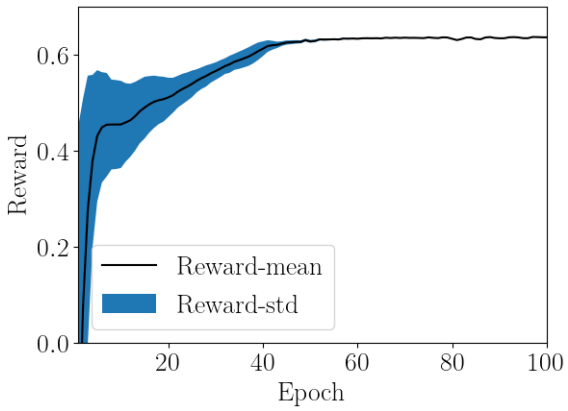
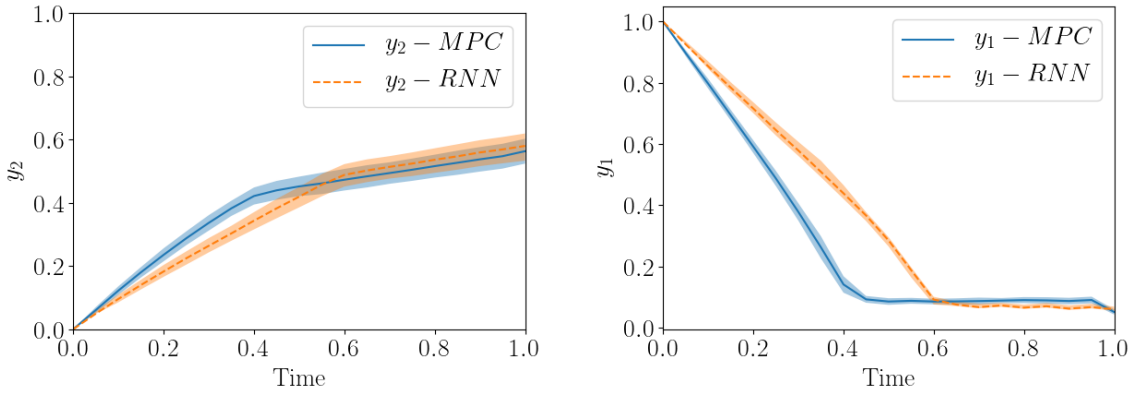


Figure 8: The reward computed for the approximate model for each epoch

Fig. 8 shows that the reward has a large variance at the beginning of the training but is undetectable at the end. This can be explained as the trade-off between exploration and exploitation, where initially there is a lack of information and policy explores possible control actions, while at the end the policy exploits the available information. This policy can be considered as an initialization of the Reinforce algorithm which uses transfer learning to incorporate new knowledge gained from the true plant (Steps 3-5 in Algorithm 2). New data-sets from 25 batches are used (*i.e.* 25 real plant epochs) to update

the true plant's RL policy. The solution after only 4 epochs is 0.591 while the stochastic-free optimal solution identified using the *unknown* (complex) model of the plant is 0.583. This results show that the stochastic nature of the system can also affect the performance. The reward for each epoch is depicted in Fig. 10 and the process trajectories after the last epoch are depicted in Fig. 9. Notice that even before having any interaction with the "real" system the proposed approach has a superior performance than NMPC. This is because RL directly accounts for the effect of future uncertainty and its feedback in a proper closed-loop manner, whereas NMPC assumes open-loop control actions at future time points in the prediction.



(a) Solution of y_2 for the "real" system

(b) Solution of y_1 for the "real" system

Figure 9: The time trajectories produced by the real plant using our approach (dash) and NMPC (solid).

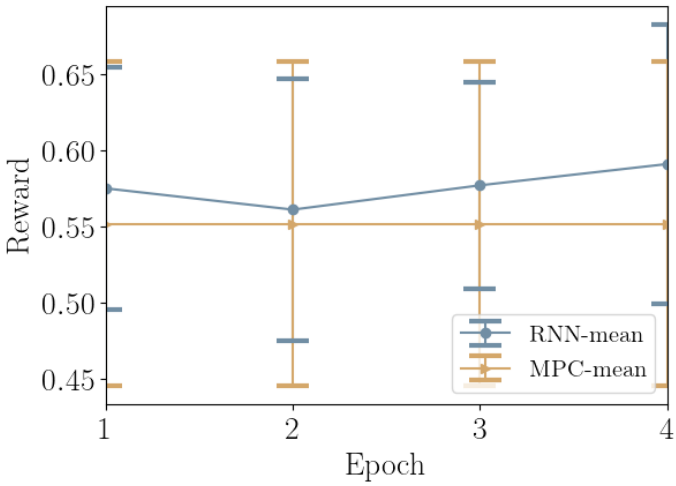
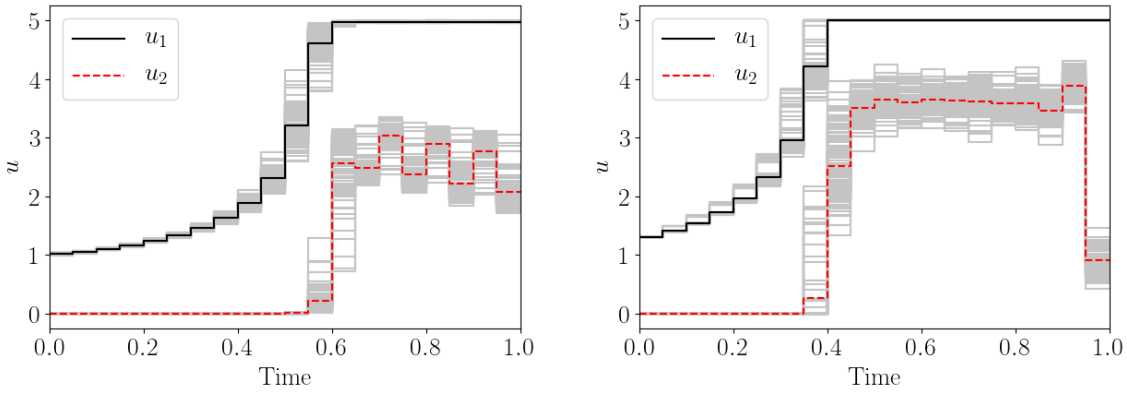


Figure 10: The reward computed by the updated training using the plant ("real" system) for each epoch (circle) and the average performance of the NMPC (triangle) with 2 times the standard deviation.

There is a variation on the results after the last batch upon which the policy is updated. This makes sense, since the system appears to have some additive noise (i.e. Gaussian disturbance) and the policy maintains its stochastic nature.

The results are also compared with the use of NMPC using shrinking horizon. The results can be seen in Fig. 10, where 100 Monte-Carlo simulations were conducted. The optimization using our approach appears to be superior to the one given by the NMPC, showing the significance of our result. Furthermore, it should be noted that the performance of our proposed policy is better even in epoch 1, before the adaptation is started. In addition, in Fig. 11, the comparison between the control inputs of that are computed using our approach and the NMPC.



(a) Solution of RNN for the "real" system

(b) Solution of NMPC for the "real" system

Figure 11: Comparison of the time trajectories of the piecewise constant control actions between our approach (left) and NMPC (right)

3.2. Case Study 2 - Stochastic Differential Equations

In this case study the same type of reaction is assumed to follow a stochastic differential equations:

$$dy_1 = \left[-(u_1 + 0.5\ u_1^2)y_1 + 0.5\frac{u_2 y_2}{(y_1 + y_2)}\right] dt$$
$$dy_2 = [u_1\ y_1 - 0.7u_2\ y_1]\, dt + [0.1\sqrt{y_1}]\, dW$$

where W is Wiener stochastic process. The simplified model is assumed to be the same with the previous case study. As a result the same policy that is trained off-line is used here. The purpose of this case study is to observe how the same policy can adapt in different environments. Now the model that describes the real system is not only structurally different, but also stochastic in nature.

The same hyperparmeters and networks are utilized for the policies in both stages, in order to show that the same policy can adapt to different environments successfully.

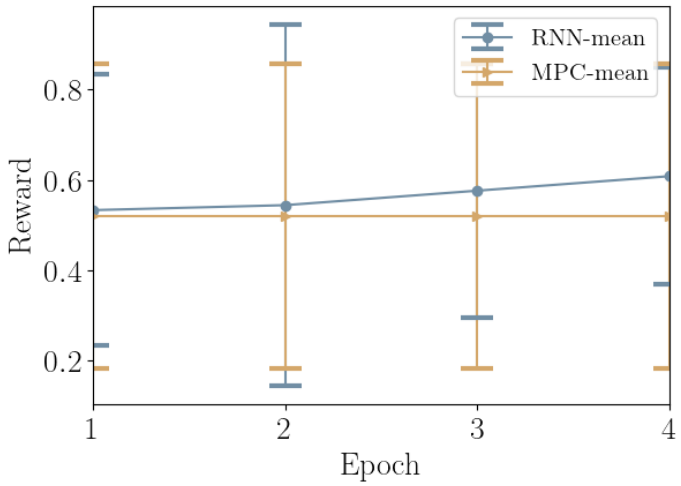
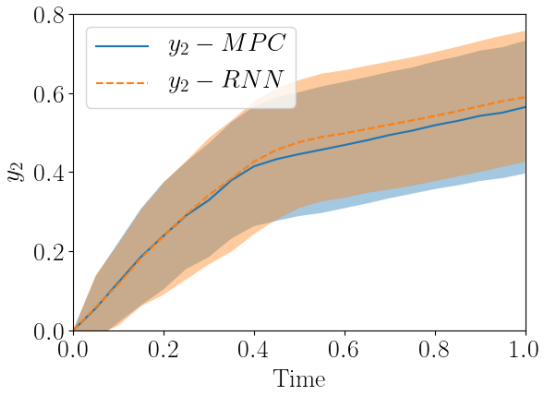
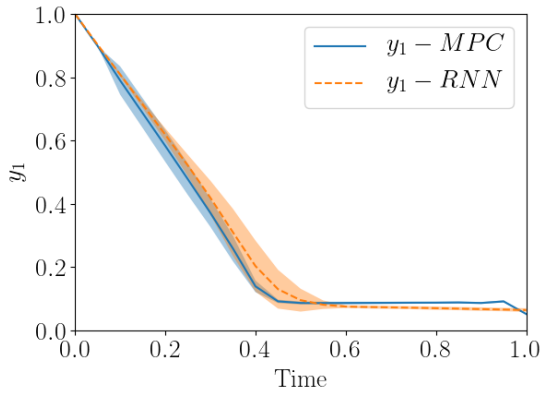


Figure 12: The reward computed by the updated training using the plant ("real" system) for each epoch (circle) and the average performance of NMPC (triangle) with 2 times the standard deviation.

The same validation is conducted here using 100 Monte-Carlo simulations. Through comparison, our approach is found to be superior to the NMPC. In this case, our proposed algorithm adapts more rapidly to the new conditions, reducing significantly the requirement for a large number of episodes and epochs, as it can be seen in Fig. 12. This is attributed to the systematic transfer learning proposed in our algorithm. The computationally intensive part has been shifted off-line where the preliminary inaccurate model was used to train the policy. Then the (deep) recurrent neural network adapts successfully to the new environment that consists of a system of stochastic differential equations. The comparison is also depicted in Fig. 13. This result is also observed in the previous case study where the stochastic part of the physical system has a different nature. The control inputs are depicted in Fig. 14.

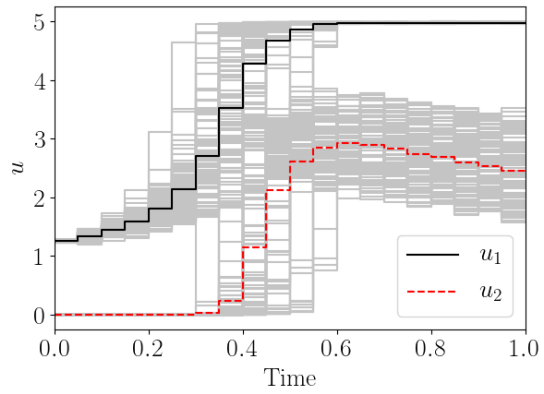


(a) Solution of y_2 for the "real" system

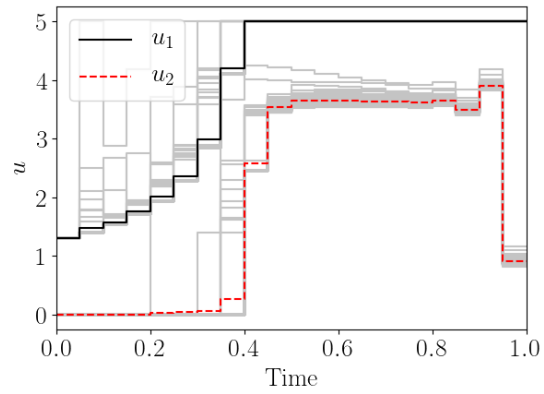


(b) Solution of y_1 for the "real" system

Figure 13: The time trajectories produced by the real plant using our approach (dash) and NMPC (solid).



(a) Solution of RNN for the "real" system



(b) Solution of NMPC for the "real" system

Figure 14: Comparison of the time trajectories of the piecewise constant control actions between our approach and NMPC

It should be noted that in both case studies the NMPC produced very similar control actions, with the only difference being the variance, compared to our approach which shapes the control actions to fit the needs of the different dynamics and uncertainty.

The methods used in the Reinforce algorithm usually require substantial number of episodes and epochs, therefore a good initial solution in combination with transfer learning is paramount so that Step 4 can be completed with a few batch-to-batch runs. In order to keep the problem realistic, only a small number of batches is utilized in Step 2-3 to refine the policy network.

3.3. Case Study 3 - Nonsmooth Model

The last case study in this paper focuses on the photo-production of phycocyanin synthesized by cyanobacterium Arthrospira platensis. Phycocyanin is a high-value bioproduct and its biological function is to enhance the photosynthetic efficiency of cyanobacteria and red algae. It has applications as a natural colorant to replace other toxic synthetic pigments in both food and cosmetic production. Additionally, the pharmaceutical industry considers it as beneficial because of its unique antioxidant, neuroprotective, and anti-inflammatory properties.

Both the "real" and simplified model in this case are considered to be nonmooth. Due to different growth phases, nonmooth behaviour is observed for the physical system. To accommodate this difficulty, switching functions have been proposed [15, 56]. In this work the nonsmooth behaviour is modelled using a sign$(\cdot)$ function. The "real" dynamic system consists of three nonsmooth ODEs describing the evolution of the concentration of biomass (X), nitrate (N), and product (q). The dynamic model is based on Monod kinetics, which describes microorganism growth in nutrient sufficient cultures, where intracellular nutrient concentration is kept constant because of the rapid replenishment. We assume a fixed volume fed-batch. The manipulated variables as in the previous examples are the light intensity (I) and inflow rate (F_N). The mass balance equations are

$$\frac{dc_x}{dt} = u_m \frac{I}{I + k_s + I^2/k_i} c_x \frac{c_N}{c_N + K_N} - u_d c_X \tag{22}$$

$$\frac{dc_N}{dt} = -Y_{N/X} u_m \frac{I}{I + k_s + I^2/k_i} c_x \frac{c_N}{c_N + K_N} + F_N \tag{23}$$

$$\frac{dc_q}{dt} = \begin{cases} k_m \frac{I}{I + k_{sq} + I^2/k_{iq}} c_x \frac{c_N}{c_N + K_N} - k_d \frac{c_q}{C_N + K_N q} & \text{, if } c_N \leq 500 mg L^{-1} \& c_X \geq 10 g L^{-1} \\ 0 & \text{, otherwise,} \end{cases} \tag{24}$$

where the parameters are given in Table 1. The real physical system consists of additive disturbance

$$w(t) = sin(t)\sigma_d + \sigma_n \tag{25}$$

$$\sigma_d = diag(4 \times 10^{-3}, 1., 1 \times 10^{-7}) \tag{26}$$

$$\sigma_n \sim \mathcal{N}(\mathbf{0}, \sigma_d), \tag{27}$$

and measurement noise

$$noise(t) \sim \mathcal{N}(\mathbf{0}, diag(4 \times 10^{-4}, .1, 1 \times 10^{-8})). \tag{28}$$

Additionally, uncertainty is assumed for the initial concentration, where

$$\begin{bmatrix} c_x(0) & c_N(0) & c_q(0) \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 1. & 150. & 0. \end{bmatrix}, diag(1 \times 10^{-3}, 22.5, 0.)). \tag{29}$$

The reward is additionally penalized by the change of the control actions $\mathbf{u}(t) = [I, F_N]^T$. As a result the reward is given as:

$$\begin{aligned} R_t &= -\Delta \mathbf{u}_t^T diag(3.125 \times 10^{-8}, 3.125 \times 10^{-6}) \Delta \mathbf{u}_t^T, \qquad t \in \{0, T-1\} \\ R_T &= c_q(T), \end{aligned} \tag{30}$$

where $\Delta \mathbf{u}_t = \mathbf{u}_t - \mathbf{u}_{t-1}$.

The simplified deterministic model is assumed without the noise or the additive disturbance. This preliminary model, is utilized in an extensive offline training in order to construct the control policy network. As illustrated in the previous section 3.4, there is a potential to have a close approximation of the solution of the OCP using RNN-Reinforce.

The training consists on 100 epochs and 500 episodes and the optimal control policy that maximizes the reward in equation 30. The control actions are constrained to be in the interval $0 \leq F_N \leq 40$ and $120 \leq T \leq 400$. The control policy RNN is designed to contain 4 hidden layers, each of which comprises 20 neurons embedded by a leaky rectified linear unit (ReLU) activation function. Furthermore, in this case a unified policy network with diagonal variance is utilized such that the control actions share memory and the previous states are used from the RNN (together the current measured states).

The algorithm is implemented in Pytorch with the same configurations. It should be noted that the computational cost of computing the control action online is insignificant since it only requires the evaluation of the corresponding RNN, and does not depend directly on the complexity or the number of variables. In contrast, the solution of the OCP scales badly in this case due to the presence of integer variables. The reward for each epoch is depicted in Fig 15. In this case the probability density is shown due to the nonsmoothness of the model, that may result in multiple peaks. The lines are faded out towards earlier epochs. Additionally, there is no guarantee of global optimality in the current work, as a result the reward may get stuck in other local minima. It should be noted that in this case the uncertain initial conditions are applied during this training phase.

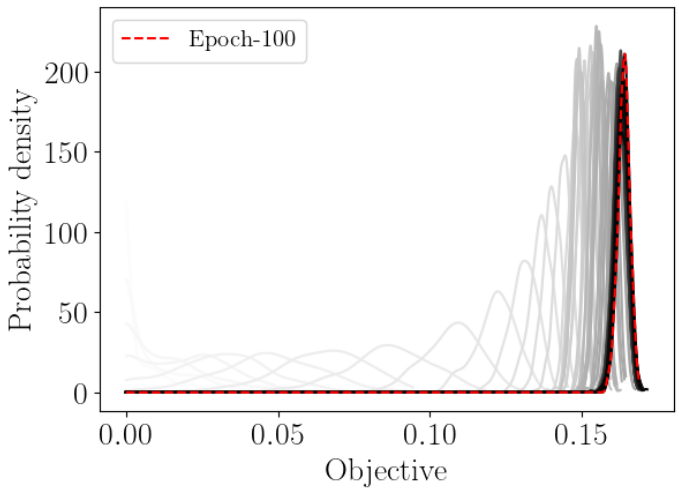
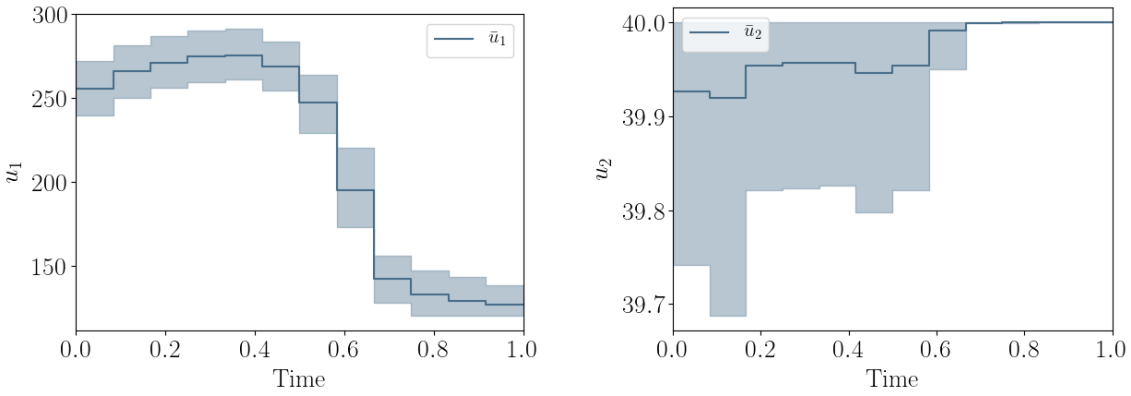


Figure 15: The reward computed for the approximate model for each epoch

The nominal control actions are depicted in Fig. 16, where the shaded areas are the 98% and 2% percentiles. The corresponding states are depicted in Fig.17 with their 98% and 2% percentiles. The nominal behaviour is subject to the corresponding initial conditions since no other uncertainty is taken into account in the offline procedure. The probability density of the product c_q has clearly only one peak, this is shown in Fig. 17c.



(a) Nominal solution for I

(b) Nominal solution for F_N

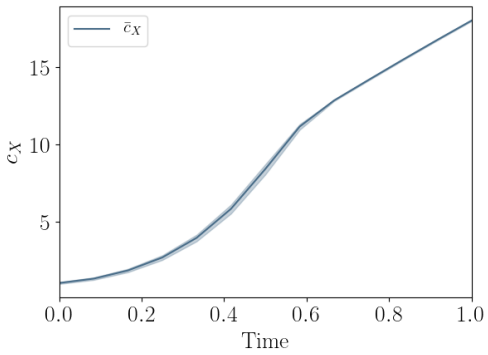
Figure 16: Solution for control actions of the nominal system using RNN

Table 1: Parameter values for physical system (22 - 24)

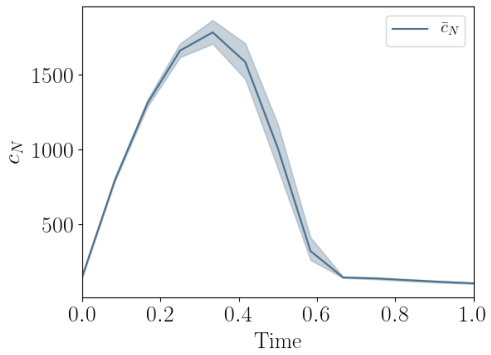
Parameter	Values	
u_m	0.0572	h^{-1}
u_d	0.0	h^{-1}
K_N	393.1	mg/L
Y_{NX}	504.1	mg/g
k_m	0.00016	$mg/g/h$
k_d	0.281	h^{-1}
k_s	178.9	$\mu mol/m^2/s$
k_i	447.1	$\mu mol/m^2/s$
k_{sq}	23.51	$\mu mol/m^2/s$
k_{iq}	800	$\mu mol/m^2/s$
K_{NP}	16.89	mg/L

As in the previous case studies, the results are compared with the use of NMPC using shrinking horizon. The optimization is a mixed integer nonlinear programming problem (MINLP). Local optimization is used in order to be numerically tractable. Orthogonal collocation is implemented and integer variables have been used to model the switches. It should be noted that this MINLP takes 2-4 mins to be solved.

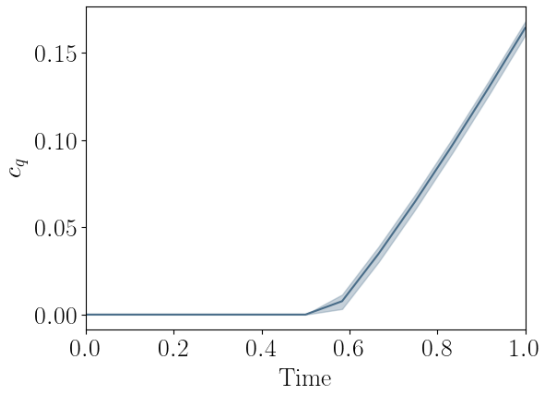
The results can be seen in Fig. 18 with their 98% & 2% percentile respectively, where 100 Monte-Carlo simulations were conducted. The optimization using our approach appears to be superior to the one given by the NMPC, showing the significance of our result. After the adaptation the probability densities are depicted in Fig. 19. Next, the control inputs are depicted in Fig. 20 with their 98% & 2% percentile respectively. It is clear that the NMPC control actions have large variance compare to the ones produce by our proposed methodology. This is due to the nonsmoothness of the model and the uncertainty which the NMPC struggles with.



(a) The nominal solution for c_X

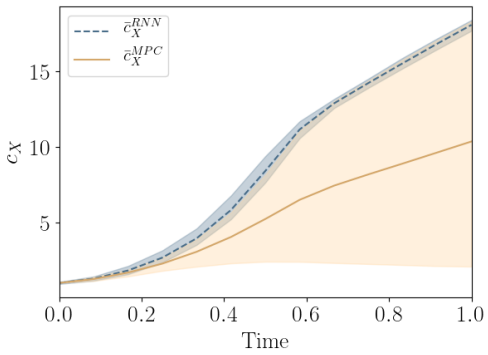


(b) The nominal solution for c_N

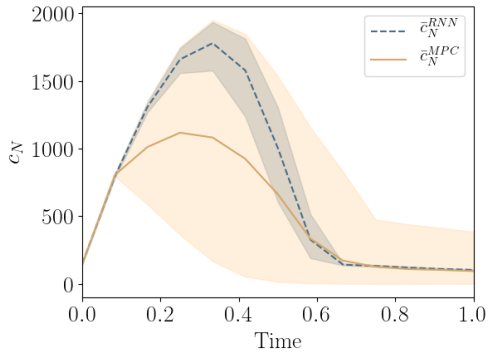


(c) The nominal solution for c_q

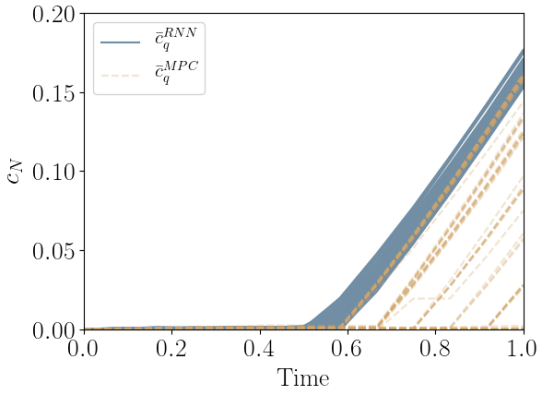
Figure 17: Comparison of the responses when RNN and NMPC are applied to the "real" physical system.



(a) The solution for c_X



(b) The solution for c_N



(c) The solution for c_q

Figure 18: Comparison of the responses when RNN and NMPC are applied to the "real" physical system.

In addition, in Fig. 20, the comparison between the control inputs of our approach and the NMPC is presented.

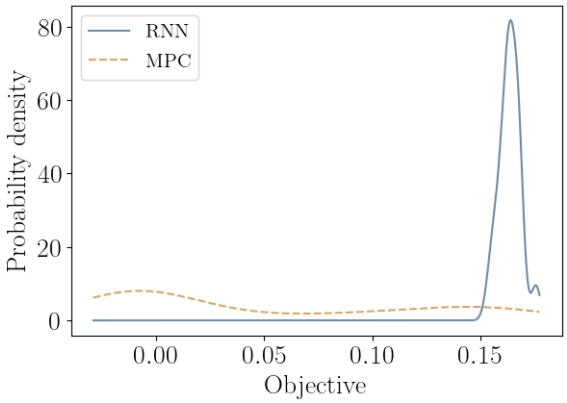
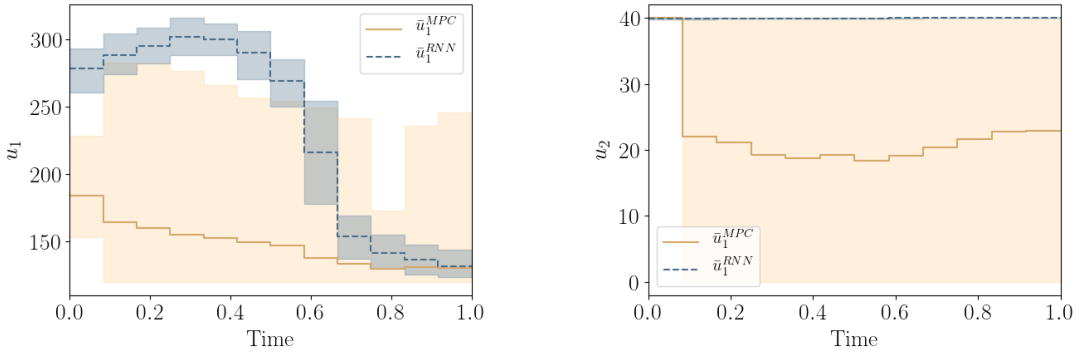


Figure 19: Probability density function for the reward computed by the updated training using the plant ("real" system) for each epoch and performance of NMPC.



(a) Solution for I of the "real" system

(b) Solution for F_N of the "real" system

Figure 20: Comparison of the time trajectories of the piecewise constant control actions between our approach and NMPC

4. Conclusions and Future Work

In this work we propose a new methodology for batch-to-batch learning by adapting Reinforcement learning techniques to uncertain and complex bioprocesses. The results reveal that it is possible to obtain a near optimal policy for a stochastic system when the true dynamics are unknown. In real systems with the absence of a true model, it is impossible to generate highly accurate datasets to train the policy network. As a result we propose a 2-stage framework where first an approximate (possibly stochastic) model is used to train the policy network. Subsequently, this policy is implemented into the *true* system. In this way, there is no need for a large number of evaluations of the true system which can be costly and time consuming.

A systematic adaptation to the new environment is achieved using transfer learning. In Step 4: Transfer Learning Reinforce the policy is trained using $T << T_0$ episodes conducting the Steps 1-3 of Algorithm 1. The proposed algorithm is validated using two case studies for different nature of stochastic processes. Our proposed methodology results in a policy that overcomes the performance of the NMPC, where only simple policy evaluations are needed.

The off-line CPU time is 3 hours, however the online implementation of the needs only 0.002 secs. This means that all the computational complexity is shifted offline and an efficient optimal control policy is constructed. One should also keep in mind that after the off-line training the solution to a nonlinear stochastic dynamical system is provided, in the form of a stochastic policy. This is a more complete and efficient solution as it is a closed-loop solution, rather than an open-loop optimization. Furthermore, a nonsmooth system was integrated with Casadi [1], which is more time consuming than integrating a smooth dynamic system.

For both the case studies 4 epochs and 25 batches were implemented. In this work, the training was stopped after the 4^{th} epoch, but the training could have been continued or stopped earlier. Here, the total number of batches is $4 \times 25 = 100$; however, the policy for all case studies performs better from the beginning of the online implementation. This means that a smaller number of batches can be used and still outperform NMPC.

We emphasize that our considered systems contain both stochasticity and plant-model mismatch, and there is no process structure available. The optimisation of such systems is generally known to be intractable. Given the early stage of this research, there are still disadvantages of this method which must be accommodated in the future, including the robust satisfaction of constraints. In addition, there is a wide discussion regarding the safety in reinforcement learning [53], which is also a result of the difficulty of robust satisfaction of constraints. Future work will focus on the robust satisfaction of constraints in RL methods.

The codes are available at: https://gitlab.com/Panos108/rl-with-nonsmooth

Acknowledgements

This project has received funding from the EPSRC project (EP/P016650/1).

Bibliography

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, M. Diehl, 2019. CasADi – A software framework for nonlinear optimization and optimal control. Mathematical Programming Computation 11 (1), 1–36.

[2] S. Arimoto, S. Kawamura, F. Miyazaki, 1984. Bettering operation of robots by learning. Journal of Robotic Systems 1 (2), 123–140.

[3] E. Aydin, D. Bonvin, K. Sundmacher, 2018. Toward fast dynamic optimization: An indirect algorithm that uses parsimonious input parameterization. Industrial & Engineering Chemistry Research 57 (30), 10038–10048.

[4] A. Bemporad, M. Morari, 1999. Robust model predictive control: A survey. Robustness in identification and control 245, 207–226.

[5] D. Bernardini, A. Bemporad, 2012. Stabilizing model predictive control of stochastic constrained linear systems. IEEE Transactions on Automatic Control 57 (6), 1468–1480.

[6] D. P. Bertsekas, 2000. Dynamic Programming and Optimal Control, 2nd Edition. Athena Scientific.

[7] L. T. Biegler, 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

[8] D. Bonvin, B. Srinivasan, D. Ruppen, 2001. Dynamic optimization in the batch chemical industry. Chemical Process Control-VI.

[9] E. Bradford, L. Imsland, 2018. Stochastic nonlinear model predictive control using gaussian processes. In: 2018 European Control Conference (ECC). pp. 1027–1034.

[10] E. Bradford, A. M. Schweidtmann, D. Zhang, K. Jing, E. A. del Rio-Chanona, 2018. Dynamic modeling and optimization of sustainable algal production with uncertainty using multivariate Gaussian processes. Computers & Chemical Engineering, 37.

[11] L. Brennan, P. Owende, 2010. Biofuels from microalgaeA review of technologies for production, processing, and extractions of biofuels and co-products. Renewable and Sustainable Energy Reviews 14 (2), 557–577.

[12] B. Chachuat, B. Srinivasan, D. Bonvin, 2009. Adaptation strategies for real-time optimization. Computers and Chemical Engineering 33 (10), 1557–1567.

[13] D. Chaffart, L. A. Ricardez-Sandoval, 2018. Optimization and control of a thin film growth process: A hybrid first principles/artificial neural network based multiscale modelling approach. Computers & Chemical Engineering 119, 465–479.

[14] E. del Rio Chanona, J. Alves Graciano, Bradford, B. Chachuat, 2019. Modifier-Adaptation Schemes Employing Gaussian Processes and Trust Regions for Real-Time Optimization. IFAC-PapersOnLine.

[15] E. A. del Rio-Chanona, P. Dechatiwongse, D. Zhang, G. C. Maitland, K. Hellgardt, H. Arellano-Garcia, V. S. Vassiliadis, 2015. Optimal operation strategy for biohydrogen production. Industrial & Engineering Chemistry Research 54 (24), 6334–6343.
URL `https://doi.org/10.1021/acs.iecr.5b00612`

[16] E. A. del Rio-Chanona, J. L. Wagner, H. Ali, D. Zhang, K. Hellgardt, 2018. Deep learning based surrogate modelling and optimization for microalgal biofuel production and photobioreactor design. AIChE Journal, in press.

[17] E. A. Del RioChanona, X. Cong, E. Bradford, D. Zhang, K. Jing, 2018. Review of advanced physical and datadriven models for dynamic bioprocess simulation: Case study of algaebacteria consortium wastewater treatment. Biotechnology and Bioengineering, bit.26881.

[18] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, 2013. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.
URL `http://arxiv.org/abs/1310.1531`

[19] C. Feller, M. Ouerghi, C. Ebenbauer, 2016. Robust output feedback model predictive control based on relaxed barrier functions. In: 2016 IEEE 55th Conference on Decision and Control (CDC). pp. 1477–1483.

[20] W. Gao, S. Wenzel, S. Engell, 2015. Modifier adaptation with quadratic approximation in iterative optimizing control. In: 2015 European Control Conference (ECC). pp. 2527–2532.

[21] I. Harun, E. A. Del Rio-Chanona, J. L. Wagner, K. J. Lauersen, D. Zhang, K. Hellgardt, 2018. Photocatalytic Production of Bisabolene from Green Microalgae Mutant: Process Analysis and Kinetic Modeling. Industrial & Engineering Chemistry Research 57 (31), 10336–10344.

[22] K. Jing, Y. Tang, C. Yao, E. A. del Rio-Chanona, X. Ling, D. Zhang, 2018. Overproduction of L-tryptophan via simultaneous feed of glucose and anthranilic acid from recombinant Escherichia coli W3110: Kinetic modeling and process scale-up. Biotechnology and Bioengineering 115 (2), 371–381.

[23] K. K. K. Kim, R. D. Braatz, 2013. Generalised polynomial chaos expansion approaches to approximate stochastic model predictive control. International Journal of Control 86 (8), 1324–1337.
[24] D. P. Kingma, J. Ba, 2014. Adam: A Method for Stochastic Optimization.
[25] D. Krishnamoorthy, M. Thombre, S. Skogestad, J. Jäschke, 2018. Data-driven Scenario Selection for Multistage Robust Model Predictive Control. IFAC-PapersOnLine 51 (20), 462–468.
[26] A. Krizhevsky, I. Sutskever, G. E. Hinton, 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.
[27] J. H. Lee, J. M. Lee, 2006. Approximate dynamic programming based approach to process control and scheduling. Computers & Chemical Engineering 30 (10-12), 1603–1618.
[28] J. H. Lee, K. S. Lee, 2007. Iterative learning control applied to batch processes: An overview. Control Engineering Practice 15 (10), 1306 – 1318, special Issue - International Symposium on Advanced Control of Chemical Processes (ADCHEM).
[29] J. H. Lee, J. Shin, M. J. Realff, 2018. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Computers & Chemical Engineering 114, 111–121.
[30] J. M. Lee, J. H. Lee, 2005. Approximate dynamic programming-based approaches for inputoutput data-driven control of nonlinear processes. Automatica 41 (7), 1281–1288.
[31] S. Lucia, S. Engell, 2012. Multi-stage and two-stage robust nonlinear model predictive control. Vol. 4. IFAC.
[32] A. Marchetti, G. François, T. Faulwasser, D. Bonvin, 2016. Modifier Adaptation for Real-Time OptimizationMethods and Applications. Processes 4 (4), 55.
[33] A. Mesbah, 2016. Stochastic model predictive control: An overview and perspectives for future research. IEEE Control Systems Magazine 36 (6), 30–44.
[34] V. Mnih, N. Heess, A. Graves, K. Kavukcuoglu, 2014. Recurrent Models of Visual Attention.
URL http://arxiv.org/abs/1406.6247
[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, 2013. Playing Atari with Deep Reinforcement Learning.
URL http://arxiv.org/abs/1312.5602
[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, 2015. Human-level control through deep reinforcement learning. Nature 518, 529.
[37] C. Peroni, N. Kaisare, J. Lee, 2005. Optimal control of a fed-batch bioreactor using simulation-based approximate dynamic programming. IEEE Transactions on Control Systems Technology 13 (5), 786–790.
[38] P. Petsagkourakis, W. P. Heath, J. Carrasco, C. Theodoropoulos, 2019. Input-output stability of barrier-based model predictive control.
URL http://arxiv.org/abs/1903.03154
[39] P. Petsagkourakis, W. P. Heath, C. Theodoropoulos, 2018. Stability analysis of piecewise affine systems with multi-model model predictive control.
URL http://arxiv.org/abs/1808.00307
[40] P. Petsagkourakis, I. O. Sandoval, E. Bradford, D. Zhang, E. del Rio-Chanona, 2019. Reinforcement learning for batch-to-batch bioprocess optimisation. In: A. A. Kiss, E. Zondervan, R. Lakerveld, L. zkan (Eds.), 29th European Symposium on Computer Aided Process Engineering. Vol. 46 of Computer Aided Chemical Engineering. Elsevier, pp. 919 – 924.

URL http://www.sciencedirect.com/science/article/pii/B9780128186343501545
[41] J. B. Rawlings, D. Q. Mayne, M. M. Diehl, 2017. Model Predictive Control: Theory, Computation, and Design.
[42] F. Rossi, F. Manenti, G. Buzzi-Ferraris, G. Reklaitis, 2019. Stochastic NMPC/DRTO of batch operations: Batch-to-batch dynamic identification of the optimal description of model uncertainty. Computers and Chemical Engineering 122, 395–414.
URL https://doi.org/10.1016/j.compchemeng.2018.08.014
[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115 (3), 211–252.
[44] H. Shah, M. Gopal, 2016. Model-Free Predictive Control of Nonlinear Processes Based on Reinforcement Learning. IFAC-PapersOnLine 49 (1), 89–94.
[45] P. Stechlinski, M. Patrascu, P. I. Barton, 2018. Nonsmooth differential-algebraic equations in chemical engineering. Computers & Chemical Engineering 114, 52 – 68, fOCAPO/CPC 2017.
URL http://www.sciencedirect.com/science/article/pii/S0098135417303861
[46] H. T. Su, T. J. McAvoy, P. Werbos, 1992. Long-Term Predictions of Chemical Processes Using Recurrent Neural Networks: A Parallel Training Approach. Industrial and Engineering Chemistry Research 31 (5), 1338–1352.
[47] R. Sutton, A. Barto, 2018. Reinforcement Learning: An Introduction Second Edition. MIT Press.
[48] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour, 1999. Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems. NIPS'99. MIT Press, Cambridge, MA, USA, pp. 1057–1063.
[49] W. Tang, P. Daoutidis, 2018. Distributed adaptive dynamic programming for data-driven optimal control. Systems & Control Letters 120, 36–43.
[50] J. Thierie, 2004. Modeling threshold phenomena, metabolic pathways switches and signals in chemostat-cultivated cells: the Crabtree effect in Saccharomyces cerevisiae. Journal of theoretical biology 226 (4), 483–501.
[51] V. S. Vassiliadis, R. W. H. Sargent, C. C. Pantelides, 1994. Solution of a class of multistage dynamic optimization problems. 2. problems with path constraints. Industrial & Engineering Chemistry Research 33 (9), 2123–2133.
[52] V. Venkatasubramanian, 2019. The promise of artificial intelligence in chemical engineering: Is it here, finally? AIChE Journal 65 (2), 466–478.
[53] K. P. Wabersich, M. N. Zeilinger, 2018. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. CoRR abs/1812.05506.
URL http://arxiv.org/abs/1812.05506
[54] R. J. Williams, 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8 (3-4), 229–256.
[55] J.-X. Xu, Y. Chen, T. H. Lee, S. Yamamoto, 1999. Terminal iterative learning control with an application to rtpcvd thickness control. Automatica 35 (9), 1535 – 1542.
[56] D. Zhang, P. Dechatiwongse, E. A. Del-Rio-Chanona, K. Hellgardt, G. C. Maitland, V. S. Vassiliadis, 2015. Analysis of the cyanobacterial hydrogen photoproduction process via model identification and process simulation. Chemical Engineering Science 128, 130 – 146.
URL http://www.sciencedirect.com/science/article/pii/S0009250915000883
[57] D. Zhang, V. S. Vassiliadis, 2015. Chlamydomonas reinhardtii Metabolic Pathway Analysis for Biohydrogen Production under Non-Steady-State Operation. Industrial & Engineering Chemistry

Research 54 (43), 10593–10605.