Computers and Chemical Engineering 140 (2020) 106910

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Design of control framework based on deep reinforcement learning and Monte-Carlo sampling in downstream separation

Soonho Hwangbo*, Gürkan Sin*

Process Systems and Engineering Center (PROSYS), Department of Chemical & Biochemical Engineering, Technical University of Denmark, Building 229, Kongens Lyngby 2800, Denmark

ARTICLE INFO

Article history:
Received 21 October 2019
Revised 30 April 2020
Accepted 4 May 2020
Available online 20 May 2020

Keywords:
Liquid-liquid extraction column
Deep reinforcement learning
Monte-Carlo sampling
Control system
API production
Biopharmaceuticals

ABSTRACT

This paper proposes a systematic framework to develop deep reinforcement learning (RL)-based algorithms for control system of downstream separation in biopharmaceutical process as follows. First, a simulation model as a digital twin is built and Monte-Carlo sampling generates substantial amounts of samples considering disturbances. Second, the deep RL-based control system is designed and the optimization subject to sample datasets is conducted. The methodology is implemented in a prototype software and relevant codes are shared by Mendeley Data. The proposed model is successfully applied to control the liquid-liquid extraction column for the recovery of fusidic acid as part of downstream processing. The resulting deep RL algorithm provides an operation performance with a better API recovery yield (32 % higher than open loop operation) and lower deviations (23 % lower than open loop operation) against disturbances.

1. Introduction

Downstream processing refers to separation processes used for the recovery and the purification of biological products and plays an important role in pharmaceutical production processes (Strube et al., 2011; Weatherley, 2013). Great amounts of solvents for extraction of valuable products from solutions including active pharmaceutical ingredients (APIs) could be used in the event of the downstream processing, and it requires optimal control systems to make sure the sustainability of downstream separation (Jiménez-González & Woodley, 2010). Moreover, economic aspects (i.e., processing costs in biopharmaceutical industries present as high as 70 – 90 % of the total production costs (Heinzle et al., 2007)) also entail reliable control strategies for the synthesis of sustainable biopharmaceutical products.

In some large-scale and/or complex production processes, model-based control design approach using first-principles or model identification is difficult, costly, and time-consuming (Hou & Wang, 2013). As an alternative to model-based control, data-driven control has been proposed and consistently paid attention due to the development of information science and technology (Selvi et al., 2018). Reinforcement learning (RL) is a powerful data-driven method that provides the optimal policy to solve control problems (Radac & Precup, 2019). Various research on RL have been lately highlighted in chemical process control (Petsagkourakis et al., 2020) and the results from the existing literatures motivate further study to extend RL applications to different control problems/area (Shin et al., 2019).

To the best of our knowledge, few studies have been performed addressing the application challenges and strategies for RL on the control of downstream separation process in biopharmaceutical production, which is the focus of this paper. Therefore, this paper aims to propose an intelligent control framework based on deep RL and evaluate the developed model by applying for downstream separation to improve process performance. In particular, we address the following research questions and challenges in this research:

A Develop a control system based on emerging concepts of RL using simulation data. To overcome the challenge that RL and deep-learning (DL) algorithms suffer from data deficiency, Monte-Carlo (MC) sampling approach is proposed to generate a number of feasible samples in combination with digital twin concept.

* Corresponding authors.
E-mail addresses: soohw@kt.dtu.dk (S. Hwangbo), gsi@kt.dtu.dk (G. Sin).

https://doi.org/10.1016/j.compchemeng.2020.106910
0098-1354/

Table 1
Comparison of model-free RL problems; MC: Monte-Carlo, TD: temporal-different, and MDP: Markov decision process.

Category	Method or algorithm	Purpose
Prediction	MC learning (Wiering & Van Otterlo, 2012) TD learning (Taylor et al., 2006)	Estimate the value function of an unknown MDP.
Control	On-policy (Singh et al., 2000) Off-policy (Arulkumaran et al., 2017)	Optimize the value function of an unknown MDP.
Unknown or large MDPs	Value function approximation (Konidaris et al., 2011)	Present a value function with a parameterized function instead of a table.
Policy gradient	REINFORCE (Williams, 1992) Actor-Critic (Houk et al., 1995; Peters & Schaal, 2008)	Parameterize the policy by finding best parameters in a given policy that maximize any policy objective function.

B Develop a prototype software by appropriate tools integration to facilitate the applications and deployment of emerging machine learning branch of AI to chemical engineering processes and in particular downstream separation.
C Evaluate the feasibility of deep RL-based control system for digital twin underpinned by various unit operations in particular focusing on liquid-liquid extraction (LLE) step.

This paper is structured as follows. The Section 2 suggests the review of previous works associated with process control to make sure the feasibility of deep RL-based control framework for downstream processing. In the Section 3, the overall framework integrating MC sampling and deep RL for control system is explained. The Section 4 includes the implementation of the suggested framework and relevant pseudocodes for a case study of the LLE column. The resulting codes are provided on Mendeley Data that is commonly used to host open source projects (Hwangbo & Sin, 2019). In the Section 5, the results from the case study are compared and discussed. Finally, concluding remarks are clarified in the Section 6.

2. Previous works

Data-driven control methods mainly depend on either on-line data (i.e., the controller is updated using each new measurements obtained) or off-line data (i.e., a batch of measurements are used to design the controller before it becomes operational and no additional modification is carried out in the middle of operation) (Tanaskovic et al., 2017) and related works are as follows. Diverse on-line data-driven control methods have been proposed such as radial basis function networks to deal with nonlinear functions (Irwin et al., 1995), simultaneous perturbation stochastic approximation for multiple unknown parameters with high-dimensional problems (Spall & Cristion, 1998), and model-free adaptive control using dynamic linearization technique (Z. Hou & Jin, 2010, 2013). Conventional off-line data-driven control tools such as iterative feedback tuning (Hjalmarsson, 2002), correlation-based tuning (Mišković et al., 2007), and virtual reference feedback tuning (Formentin et al., 2011) have been suggested to prevent control performance degradation. Furthermore, approximate dynamic programing that overcomes the curse-of-dimensionality by combining RL (model-free method) with dynamic programming (model-based method) has been demonstrated in a number of applications related to large and complex problems (Lee, 2014; Lee & Lee, 2006; Powell, 2009).

As many RL theories have been recently developed, the application rage of RL in real industrial markets has consistently expanded (Levine, 2018; Mnih et al., 2013; Mnih et al., 2015). Subsequently, RL as representative data-driven technique has been greatly paid attention to design new control concepts and to solve decision-making issues in the presence of uncertainty (Rocchetta et al., 2019). From the perspective of process control, the design of RL problems including an objective function for satisfying minimum energy consumption or optimizing energy cost via control system has been introduced (Arif et al., 2016; De Somer et al., 2017; Qi et al., 2016; Wang et al., 2017). Various significant potentials for RL-based applications of control system such as heating ventilation/air conditioning/domestic hot water (Al-Jabery et al., 2016; Cheng et al., 2016; Sekizaki et al., 2015; Sun et al., 2015) and distributed generation at the building and electrical storage (Berlink et al., 2015; Raju et al., 2015; Tan et al., 2018) have been addressed.

Reinforcement learning is one of three broad branches of machine learning, the rest of which are supervised learning and unsupervised learning. Both supervised learning and unsupervised learning operate using a static dataset, on the other hand, RL works with data from a dynamic environment and the goal of RL is not to label data or cluster data but to figure out the optimal actions based on goal-oriented learning from interaction (Sutton & Barto, 2018). Therefore, RL aims at optimization by learning experiences through trial and error method, and RL problem is a framing of the problem of learning from interactivity to achieve a goal (Kaelbling et al., 1996). The learner is called the agent and the thing that the agent interacts with is called the environment. The agent takes an action to the environment according to an internal policy in the agent. The environment decides how to act or move based on the action from the agent, and discharges outputs as the form of a state. Any changes in the environment and pertinent results thereof are directly examined by a predefined reward function, and the agent can obtain positive or negative rewards from the environment. Discharged states and rewards are iteratively used in the agent as input components to update the internal policy in the agent. The design of RL problems depends on the way to rapidly and precisely train the agent and optimally operate the environment. Diverse RL sub-problems can be mainly separated by the complexity of a RL problem, characteristic of the policy, and training methods for the agent. Whether complete knowledge of the environment exists or not divides RL into model-based RL and model-free RL (Arulkumaran et al., 2017; Wiering & Van Otterlo, 2012).

Dynamic programming as representative model-based RL approach assumes full knowledge of the Markov chain process and give a general solution method for problems such as planning, scheduling, and shortest path algorithm (Bellman, 1966; Howard, 1960). Most of RL methods rely on model-free models that have been recently the most active area of research and Table 1 show a comparison of specific model-free RL algorithms from the perspective of different purposes. Q-leaning, which is a member of temporal-different learning based on a bootstrapping method, has been greatly investigated in process control problems, indicating that Q-learning outperforms a rule-based control method and improves control performance (Cheng et al., 2016; Yang et al., 2015). However, Q-learning is limited to local control systems due to the costly computation time required for updating the policy in an agent such as large Markov decision process problems (Mocanu et al., 2016). Recently, a novel RL approach called deep Q-network (DQN), which integrates deep-learning with Q-learning and consists of the off-policy model-free method, has been developed (Mnih et al., 2015). Compared to model predictive control that

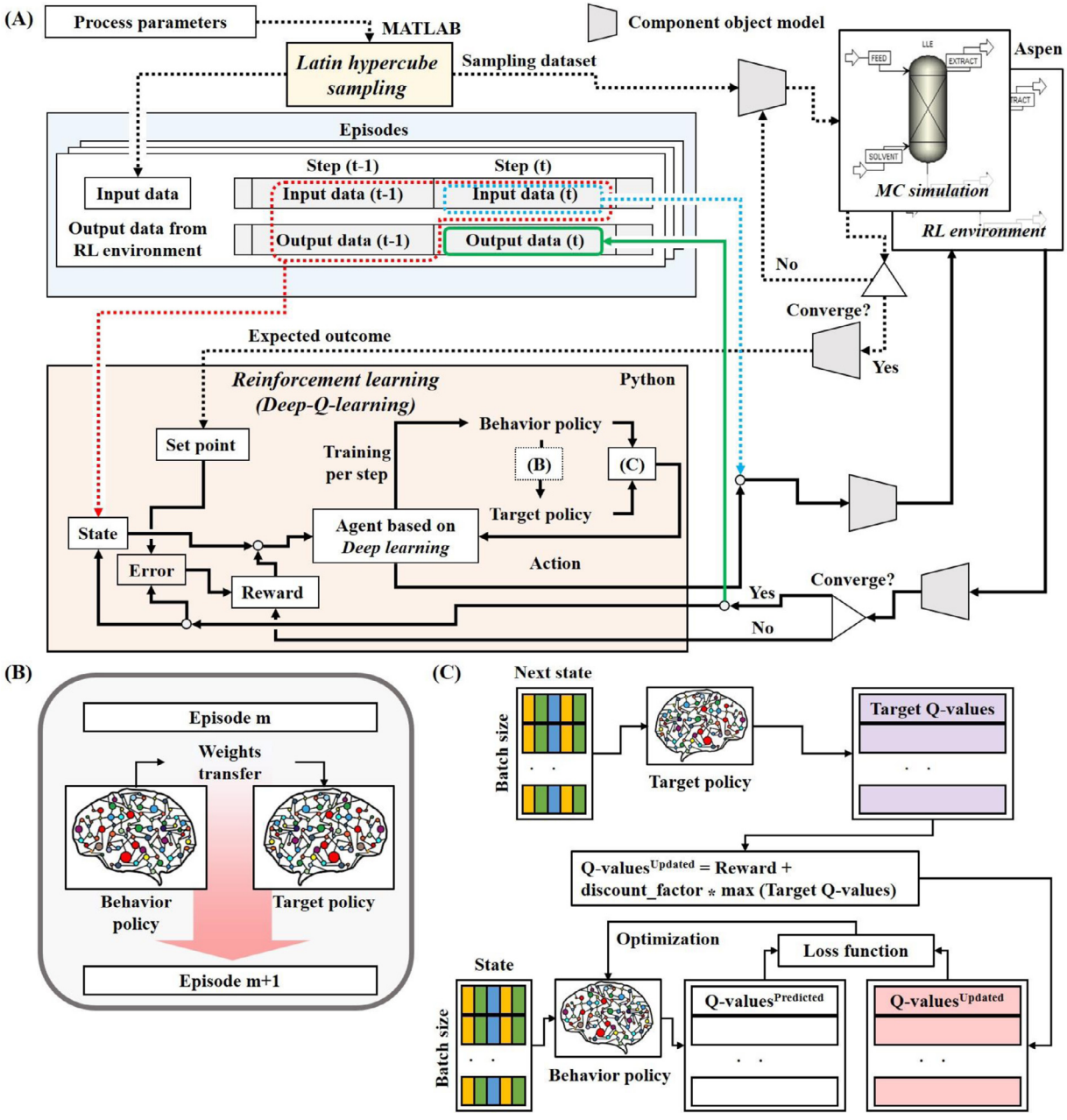


Fig. 1. A framework of the proposed control platform; (A): data generation through Latin hypercube sampling, Monte-Carlo simulation, and deep-Q-network, (B): diagram of weigh parameters transfer in deep-Q-network and (C): parameters update in behavior policy by Q-value update equation.

was suggested by the process control community and is currently the most widely used technology based on the look-ahead optimization, several research have demonstrated advantages of RL-based control systems by applying it for the physical plants such as a reference office building and bioprocesses (Ahn & Park, 2020; Petsagkourakis et al., 2020). However, the application of DQN-based control systems in downstream separation in biopharmaceutical processes has been hardly reported.

The purpose of the developed control framework based on DQN algorithm is ultimately to support digital twin in relation to virtual plants including a cascade of biopharmaceutical processes. This is done through formulating a component object model, which is a platform-independent system and provides inter-process communication object creation in a broad range of programming languages (Djamaluddin et al., 2011; Liu et al., 2002). Component object model plays a crucial role of the integration of virtual plants built on simulation tools with corresponding DQN-based control systems in an object-oriented programming environment. While in this study, we focus on a single unit operation namely LLE column as a means to evaluate the feasibility of the proposed control framework.

3. A framework for integration of Deep-Q-Network and Monte-Carlo sampling for process control

The suggested framework primarily consists of MC sampling method and deep RL-based control systems by a versatile interface based on component object model, which integrates different software (Fig. 1 (A)). Sufficient big data considering process disturbances are produced by the combination of the Latin hypercube

```
Import API property and feed stream parameters
Implement lhsdesign function
X=lhsdesign(N, p)
While p variables do
  Implement inverse cumulative distribution function
  A=icdf('type of distribution function', X, μ, σ)
  Output: n-by-p matrix A
Access COM interface
Open simulation file
Define feasible solvent flowrate range
While solvent flowrate do
  While MC simulation do
    Initialize simulation file
    Simulation.Streams.Feed.Value ← feed stream in matrix A
    Simulation.Properties.Value ← API property parameters in matrix A
    Simulatoin.Streams.Solvent.Value ← solvent flowrate
    Run simulation file
    API mass ratio ← Simulation.Streams.Output.Value
Calculate expected API mass ratio
Reshape feed stream in matrix A
Quit simulation file
Close COM interface
```

Fig. 2. Pseudocode for Monte-Carlo simulation based on Latin hypercube sampling; COM: component object model and MC: Monte-Carlo.

sampling (LHS) method, which has been used to produce massive samples with disturbances in input datasets (McKay et al., 1979; Sin et al., 2009).

DQN, which is employed in the proposed control framework, has two policies: the policy being learning about is called the target policy and the policy used to generate behavior is called the behavior policy, all of which are designed by DL networks from a point of a view of value function approximation. The reason DQN uses two policies is that efficient learning can be accomplished by combining experiences from the other policy. To represent an appropriate update timing of each policy, the entire samples from the MC sampling method should be divided into two different scales (i.e., step scale and episode scale). A set of arbitrary dozens of steps consists of one episode (i.e., the number of steps in the episode can be flexibly determined). The behavior policy is learned per each step and the target policy is improved per each episode by transferring parameters of the behavior policy into the target policy (Fig. 1 (B)). DQN utilizes the concept of supervised learning to optimize the behavior policy, which means that the target policy plays a role of an external supervisor as providing labeled data (Fig. 1 (C)). Precise introduction of mathematical models and the structural relationship of policies corresponding to a case study are elaborated in the sub Section 4.3.

An interface based on component object model is necessary for an inevitable connection with the simulation tool. Samples from the LHS designed by MATLAB are transmitted to Aspen Plus™ to run MC simulation, and a needed outcome should be transferred and saved. Actions from the agent in the DQN model developed by Python and samples from the LHS should be fed into the environment developed by Aspen Plus™. The interface extracts wanted results and information whether or not the environment converges, and the results are transferred to the reward function in the DQN to enhance the agent's performance.

4. Case study of the liquid-liquid extraction column applied to the proposed control framework

4.1. Case study description

Downstream processing is usually the most sophisticated part of manufacturing typically divided into preparation, capture, purification or separation, and polishing (Buyel et al., 2015; McPartland et al., 2012). The LLE column is commonly used as a technology in the separation step to concentrate the API product and is based on the principle of separating components depending on their relative solubilities in two different immiscible liquids. In the LLE, a first liquid feed including two or more components is directly faced with a second liquid phase, which is called the solvent (Seader et al., 1998). The purpose of a case study in this paper aims to evaluate the suggested control system manipulating solvent flowrate in order to achieve a better recovery of API compared to the existing concentration from open loop operation. The process performance of the suggested framework against disturbance rejection related to composition of feed stream and model uncertainties is demonstrated. For the latter, the uncertainty of property model parameters, which are used to calculate solubility of species in feed mixture in solvent and aqueous phases, is considered. The LLE column in this study uses segment parameters that have been empirically disclosed in the existing literature (Molla et al., 2019). Therefore, this research concentrates explanations of MC simulation based on LHS, the DQN model, and DL networks. Table 2 presents crucial parameters and primary components of the case study applied to the proposed control framework.

4.2. Monte-Carlo simulation based on Latin hypercube sampling for the case study

Main purposes of the sampling method in this case study are as follows. First, disturbance or noise in feed stream of the LLE column and segment parameters, which influence the solubility based on eNRTL-SAC property model, should be considered. Definition of input space for MC simulation is of importance because it affects the output behavior from the simulation model. Second, the DQN model should import samples of feed flowrate and feed mass ratio from the LHS and the average of the API concentration distribution from MC simulation runs.

The resulting pseudocode of MC simulation based on the LHS in this case study is depicted in Fig. 2. Samples can be produced by a built-in function in the MATLAB executable and MC simulation runs are performed using Aspen Plus™ where the LLE column is configured to represent the case study. Interface based on component object model makes it possible to transfer information and results between two programs. Reference values of regressed segment parameters of API, feed flowrate, and feed mass ratio are imported (Table 2).

The number of the total input variables, p, and the number of samples per each input variable, N, are selected and the LHS, which is the built-in function expressed as *lhsdesign,* is executed and results in N-by-p matrix A. The LLE simulation file can be open after assessing the Aspen Plus™ via the interface and feed flowrate, feed mass ratio, and segment parameters of the API in each row of the matrix A are given. According to the fixed range of amounts of feasible solvent flowrate, MC simulation runs per each solvent flowrate is iteratively implemented. The number of simulation runs can be handled and as multiplying the number of simulation runs by the number of candidates in solvent flowrates, the total number

Table 2
Parameters and assumed components in the MC sampling method, the DQN model, and DL network.

MC sampling method	
Sampling function (LHS method)	*lhsdesign* (built-in function in MATALB)
Sampling variables	• 5 segment parameters of eNRTL-SAC • Feed flowrate • Feed mass ratio
Number of each sampling variable	40,000
Type of distribution function	Normal
Mean value from MC simulation	0.3145
Standard deviation from MC simulation	0.1045
DQN model	
Total number of episode available	2,000
Step size in one episode	20
ε value in ε-greedy	0.999^t (minimum value: 0.01)
Components in a set of the state	• Previous feed flowrate • Previous feed mass ratio • Previous API mass concentration • Current feed flowrate • Current feed mass ratio.
Candidates in action	17 different solvent flowrates
Discount rate	0.99
Reward function	Eq. (7)
Starting point for agent training	After 1,000 steps (i.e., after 50 episodes)
Batch size	480 steps (i.e., 24 episodes)
DL network in the agent of the DQN model	
Activation functions (Ramachandran et al., 2017)	• Relu in two hidden layers • Linear function in the output layer
Kernel initializer	he_unifrom (He et al., 2015)
Number of epoch	1
Input dimension	5 (equal to the number of components in the state)
Output dimension	17 (equal to the number of candidates in action)
Output dimensions in hidden layers	32 of the first hidden layer 24 of the second hidden layer 17 of the output layer
Loss function	Mean squared error
Optimizer	Adam (Bock et al., 2018)
Learning rate	0.001

of simulation results is computed. After completing all MC simulation runs, a distribution of the API concentration values is obtained where the average and the standard deviation of the API concentration values can be estimated from inference statistics. The average value plays a role of the setpoint parameter in the DQN model. Samples of feed flowrate and feed mass ratio from matrix A excluding samples of segment parameters are separately saved since these data are allocated in the state as the input of the DQN-based control system.

4.3. Deep-Q-network of the control system in the liquid–liquid extraction column

Pseudocode of the DQN model and a specific structure thereof are illustrated in Fig. 3. Initially, feed stream data including feed flowrate and feed mass ratio from the LHS and the expected API concentration value computed by MC simulation runs are imported to the DQN model. Feed stream data should be reshaped as *T*-by-*M* matrix due to the feature of the updating timing of RL as follows. First, states in the conventional RL are changed whenever a step (*T*) moves forward. When the agent reaches for the terminal state, the entire system, which is generally called a game, is completed with finishing an episode (*M*). Another game starts again with the initial state in another episode. However, in the proposed control framework, one single step corresponds to one single operation of the LLE column, indicating that the initial state, the intermediate states, and the terminal state are all the dependent operation. Second, as aforementioned that the DQN requires two policies, the behavior policy is learned per each step and the target policy is improved per each episode. Therefore, step scale and episode scale are determined by artificially reshaping feed stream data into two-dimension matrix to fulfill mentioned previously two conditions. This case study assumes that each process parameter generate 40,000 samples and these are split in 2,000 episodes and 20 steps (i.e., an episode contains 20 steps).

In this research, DL networks are modeled using Keras, which is an open-source neural network and capable of running on top of TensorFlow in Python (Gulli & Pal, 2017). Detailed structure of DL networks and explanation thereof are provided in the next section. The interface is necessary for connecting between the agent and the environment because Aspen Plus™ designs the LLE column as the environment in the proposed DQN model. In principle, the DQN model saves all past experiences into the experience replay memory. The current state, *s*, the current API concentration obtained by operating the LLE column based on the action from the agent, *a*, the reward, *r*, and the next state, *s'*, are consistently accumulated in the experience replay memory, *D*. As the overall length of saved experiences is out of the range of a certain point, *L*, the policy begins to be trained by utilizing saved experiences in a batch size, *j*.

4.3.1. Off-policy using Q-learning and deep learning networks to train the agent

When the agent takes an action at the current state for the next state (i.e., state transition) according to the policy, the agent decides the action that makes sure to provide the greatest Q-value $Q^{\pi}(s, a)$ (also known as action-value). Indeed, Q-value $Q^{\pi}(s, a)$ is the expected return G_t starting from the state *s*, taking the action *a*, and thereafter following policy π (Eq. (1)). The expected return

Import *feed stream data generated from Latin hypercube sampling*
Import *expected API mass ratio calculated by MC simulation*
Reshape *feed stream data into T-by-M matrix*
Initialize *behavior action-value function* Q^b *with random* θ^b
Initialize *target action-value function* Q^π *with random* θ^π
Build Q^b *and* Q^π *in deep-learning network*
Access *COM interface*
Open *simulation file*
For *episode = 1, M* **do**
  For *step = 1, T* **do**
    With ε*-greedy,* **select** *a random action*
    otherwise $a_t = \arg\max_{a'} Q^b(s', a'; \theta^b)$
    Taking action from the agent to the environment
    Initialize *simulation file*
    Simulation.Streams.Feed.Value ← *feed stream data*$_{t,m}$
    Simulation.Streams.Solvents.Value ← *solvent flowrate related to* a_t
    Run *simulation file*
    API mass ratio ← *Simulation.Streams.Output.Value*
    Observe *reward r*
    Store s_t, *API mass ratio, r, and* s_{t+1} *in replay memory D*
    If *length of replay memory* > *L* **then**
      Start *training behavior policy*
      For *batch size = 1, J* **do**
        Extract *randomly mini batch j from D*
        If *s is terminal state in an episode* **then**
          $y = r$
        otherwise $y = r + \gamma \times \max_{a'} Q^\pi(s', a'\ ;\ \theta^\pi)$
        Perform *gradient decent on* $\left\{y - Q^b(s, a\ ;\ \theta^b)\right\}^2$
      If *length of time steps* = *T* **then**
        Reset $Q^\pi = Q^b$
      If $\frac{1}{m}\sum_{M-m}^{M} score > TMS$ *and* $\forall_{m \in M}\ score > TS$ **then**
        Save *model*
Quit *simulation file*
Close *COM interface*

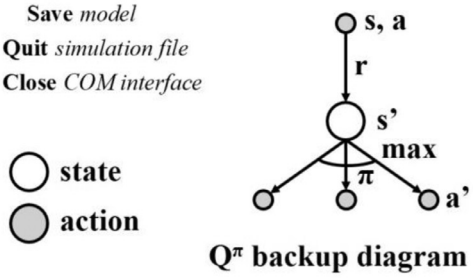


Fig. 3. Pseudocode for designing the DQN-based control system and backup diagram of Q-value update; MC: Monte-Carlo, COM: component object model, TMS: fixed total mean score, and TS: fixed total score.

G_t is the sum of the discounted rewards over the future (Eq. (2)). Q-value $Q^\pi(s, a)$ is continuously modified by updating the policy.

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{1}$$

$$G_t = \sum_{k=0}^{\infty} \left(\gamma^k \times R_{t+k+1}\right) 0 \le \gamma \le 1 \tag{2}$$

where S_t is the state at a step t, A_t is the action at a step t, γ is the discount rate, and $R_t$is the reward at a step t. In off-policy algorithm, Q-value from the behavior policy $Q^b(s, a)$ is updated towards maximizing a direction of Q-value from the target policy $Q^\pi(s, a)$, which is equal to $R + \gamma \times \max_{a\prime} Q^\pi(s\prime, a\prime)$ (backup diagram in Fig. 3). Therefore, Q-value learning algorithm in off-policy methods can be defined by Eq. (3) (Watkins, 1989).

$$Q^b(s, a) \leftarrow Q^b(s, a) + \alpha \times \left[R + \gamma \times \max_{a\prime} Q^\pi(s\prime, a\prime) - Q^b(s, a)\right] \tag{3}$$

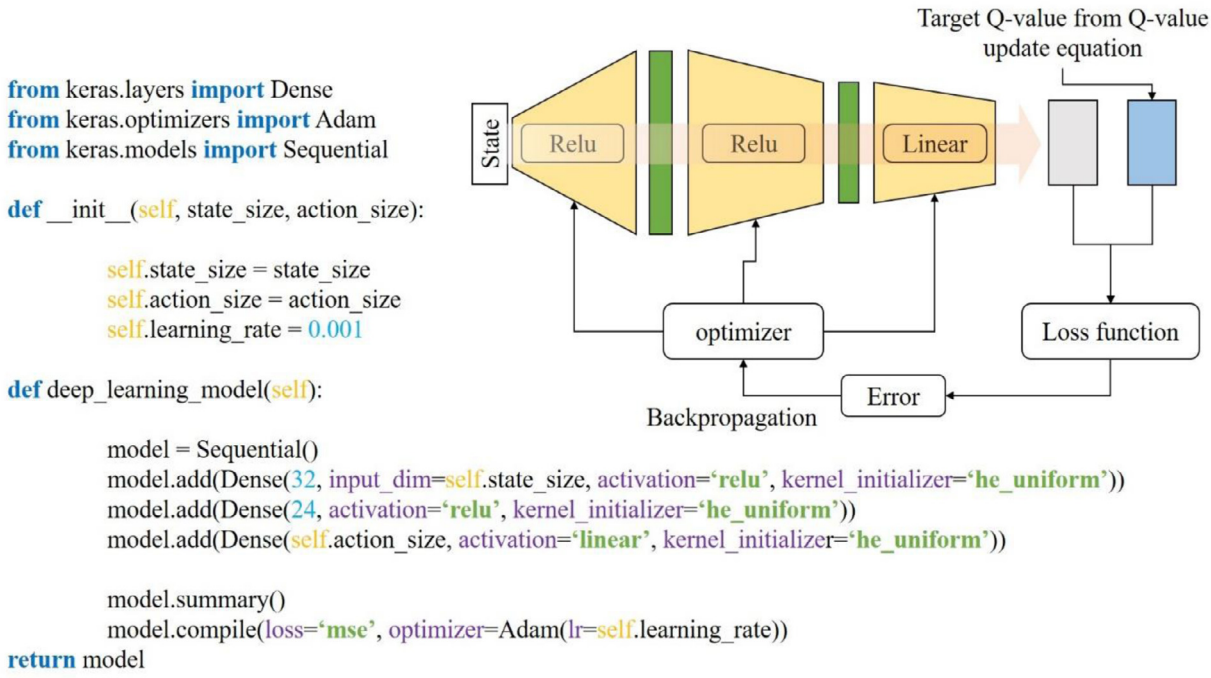
where α is a constant step-size parameter, b is the behavior policy, and π is the target policy. However, the real Q-values in the DQN are hardly known because the DQN is based on model-free problem with arbitrarily large state spaces. Therefore, in terms of approximate solution methods, the real Q-value is replaced with the target Q-value from Eq. (3). Parameter theta θ_k^b in the Q-value from the behavior policy is regressed by the loss function and the gradient decent method (Eqs. (4) and (5)) (Sutton et al., 2009): $Q^b(s, a\,; \theta^b)$ is the predicted Q-value from the behavior policy and $R + \gamma \times \max_{a\prime} Q^\pi(s\prime, a\prime\,; \theta^\pi)$ is equal to the target Q-value from the target policy in the DQN.

$$loss\ function\ = \left\{R + \gamma \times \max_{a\prime} Q^\pi(s\prime, a\prime\,; \theta^\pi) - Q^b(s, a\,; \theta^b)\right\}^2 \tag{4}$$

$$\theta_{k+1}^b \leftarrow \theta_k^b - \alpha \times \nabla_{\theta^b} \mathbb{E}_{s\prime \sim P(s\prime|s,a)} \times \left[\left\{R + \gamma \times \max_{a\prime} Q^\pi(s\prime, a\prime\,; \theta^\pi) - Q^b(s, a\,; \theta^b)\right\}^2\right]_{\theta^b = \theta_k^b} \tag{5}$$

Typical DL networks are involved with policies construction to efficiently update parameters in the behavior policy that is expected to be continued for all steps until one episode finishes, and parameters in the target policy θ^π is replaced with updated parameters in the behavior policy. The proposed DL networks consist of two hidden layers with Relu function as the activation function. Hyperparameters and dominant mathematical expressions in DL networks are presented in Fig. 4 and Table 2. Since the state corresponding to the input has 5 variables and candidates in the action are assumed by 17 different solvent flowrates, the input size and the output size of the suggested DL networks are 5 and 17, respectively. Output from the DL networks does not explicitly present the solvent flowrate value, but the action corresponding to the greatest Q-value among outputs is decided as the current optimal solvent flowrate to be transmitted to the environment. In an attempt to increase the output size of the first hidden layer compared to the initial input size and linearly reduce output sizes of the second hidden layer and the output layer, the final output size becomes the same as the number of possible solvent flowrates (Fig. 4). As aforementioned, the target Q-value from the Eq. (3) plays a role of the real Q-value and the mean squared error becomes the loss function. Adam optimizer, which is the most commonly used technique (Bock et al., 2018), is employed to minimize the loss function.

In the event of taking an action, the ε-greedy algorithm should be considered to prevent a problem of local optima, describing that most of the time the agent selects an action that has maximal estimated action-value (exploitation), but with probability ε the agent instead chooses an action at random (exploration) (Table 2). Therefore, as exploring concealed states of the environment in an at-




```
from keras.layers import Dense
from keras.optimizers import Adam
from keras.models import Sequential

def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.learning_rate = 0.001

def deep_learning_model(self):
        model = Sequential()
        model.add(Dense(32, input_dim=self.state_size, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(24, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dense(self.action_size, activation='linear', kernel_initializer='he_uniform'))

        model.summary()
        model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
return model
```

Fig. 4. Python codes and a corresponding architecture of the suggested deep-learning algorithm using Keras library.

tempt to take random actions, the agent can reach for a global optimal policy at a certain point.

4.3.2. Reward function to evaluate the environment

As the input data, solvent flowrate corresponding to the action-value determined from the agent and reshaped feed stream data to the certain step in the certain episode are fed into the LLE column developed in Aspen Plus™ via the interface. After running the LLE simulation, the API concentration is given back through the interface and a reward is calculated depending on results from the simulation run. In this research, a simple reward function is defined only to figure out the enhancement of the process performance (Eqs. (6) and (7)). The setpoint *SP* of control system is the average of all API concentration values from open loop operation in MC sampling method $API^{MC}_{Simulation}$ (i.e., the summation of API concentration values divided by the total number of simulation runs *NS*) (Eq. (6)). In case that the API concentration from the LLE column as the environment in the DQN model $API^{RL}_{Environment}$ is less than the setpoint or the LLE column operated by the action fails to be converged (i.e., if the control system is converged, *SC* is equal to zero, otherwise, one), the negative integer is rewarded, otherwise, the positive integer is rewarded (Eq. (7)).

$$SP = \sum API^{MC}_{Simulation}/NS \tag{6}$$

$$R = \begin{cases} -1 & if\ (1 - SC) \times API^{RL}_{Environment} < SP \\ 1 & otherwise \end{cases} \tag{7}$$

All rewards from the episode (i.e., one reward per one step) are added and result in the score, and the first step in the next episode starts implementing the same procedure. For example, in case that the proposed control system is converged and discharges greater API concentration values than the setpoint for every step in the episode, the total of 20 score is generated, otherwise, negative 20 score is obtained. If the expected score in the last *m* episodes is greater than the fixed mean score and each score in the last *m* episodes is greater than the fixed score, training stops.

To summarize, the following key input factors are identified in the implementation of the framework. (1) Defining a proper inputs space reflecting the case study application and generating large random samples, *N*, to perform MC simulations. (2) Building the loss function based on the target Q-value from the Q-value update algorithm and using the gradient decent method as required by the model-free and off-policy DQN model. (3) Developing DL networks to find good approximate Q-value functions in both the behavior policy and the target policy.

5. Results and discussion

5.1. Solvent selection for the case study

Segment parameters of the API and several solvents, which were disclosed by the previous works, are used in the case study to determine a proper solvent from the perspective of solvent selection (Chen & Song, 2004; Molla et al., 2019). A total of 5 solvents, each of which is iso-butanol, 1-butanol, 1,4-dioxane, methyl isobutyl ketone, and methyl tert-butyl ether, are employed to operate the LLE column designed by Aspen Plus™. Each mass flowrate of the feed stream and the solvent stream is assumed as 100 kg per hour and mass fractions of water and API in the feed stream are fixed as 0.8 and 0.2, respectively. Table 3 represents the results of the mass fractions of components in the extract stream and raffinate stream. Based on these results, methyl isobutyl ketone (MIBK) is selected among the solvent candidates to be further used in this case study.

5.2. Monte-Carlo sampling to generate large samples

Once the LLE column is designed together with an appropriate solvent, we focus on data generation for training of the deep RL algorithm. A number of datasets from MC sampling method must be generated to implement the proposed DQN since it is cumbersome to collect empirical big data from existing processes in a production facility. Feasible datasets of segment parameters influencing

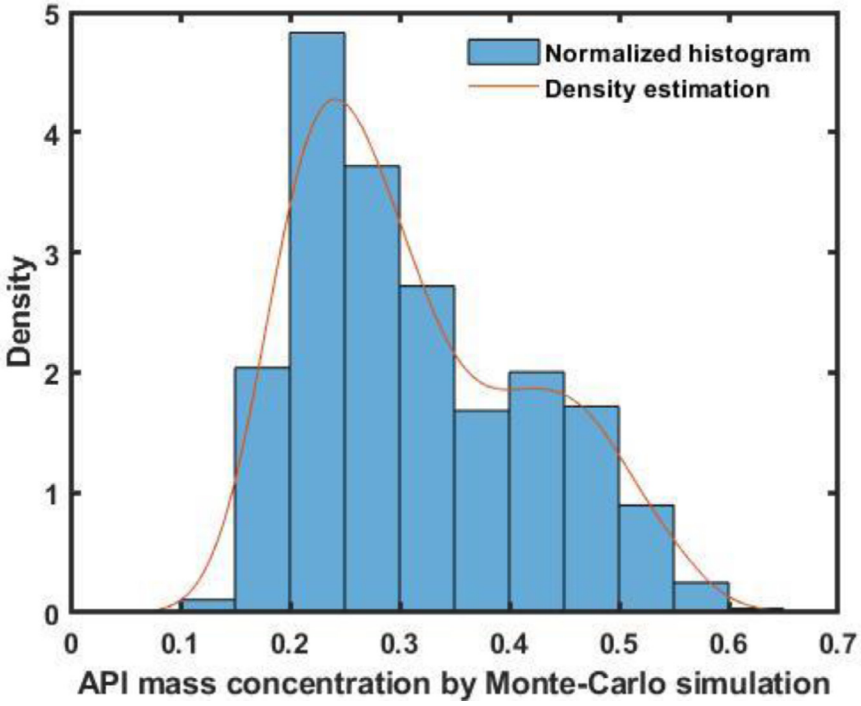


Fig. 5. API concentration distribution by Monte-Carlo simulation using Latin hypercube sampling.

Table 3
Results of mass fractions from the solvent selection using Aspen Plus™; MIBK: methyl isobutyl ketone and MTBE: methyl tert-butyl ether.

Extract stream (mass fraction)			
	Water	API	Solvent
iso-butanol	0.1264	0.1462	0.7273
1-butanol	0.1634	0.1431	0.6935
1,4-dioxane	0.0173	0.9829	0
MIBK	0.0456	0.1592	0.7952
MTBE	0.0612	0.1579	0.7810
Raffinate stream (mass fraction)			
	Water	API	Solvent
iso-butanol	0.9917	0	0.0083
1-butanol	0.9486	0	0.0514
1,4-dioxane	0.4434	0	0.5566
MIBK	0.9990	0	0.0010
MTBE	0.9857	0	0.0143

solubilities, feed flowrate, and feed mass ratio are produced by the LHS method.

To describe the input space for the sampling, the following assumptions are made. 1) All distribution functions for the identified inputs are a normally distributed. 2) Average values of segment parameters are equal to the regressed values from the literature (Molla et al., 2019) and their standard deviation is 0.01. 3) The average and the standard deviation of feed flowrate are 100 and 15. 4) The average and the standard deviation of feed mass ratio 0.2 and 0.03. The input space can easily updated and modified according to the needs and conditions specific to each application or a case study. 40,000 sample datasets are produced by the LHS method and transmitted to the LLE column developed by Aspen Plus™ through the interface. MC simulations per each solvent flowrate candidate are accomplished and the API concentration distribution is achieved (Fig. 5), showing that the average and the standard deviation of the API concentration are 0.3145 and 0.1045 corresponding to open loop operation without a control system. The shape of Fig. 5 shows a positively-skewed (also known as right skewed) curve so that high API concentrations can be maintained only in small probability density. It describes that during MC simulations, some input samples inappropriate for open loop operation appear in the given design specification of the LLE column. This leads to a skewed distribution of the output of API in the top product column. The expected API concentration from open loop operation is used for the setpoint in the DQN-based control system.

5.3. Deep-Q-network and evaluation

The LLE column in Aspen Plus™ utilizes feed stream sample datasets from the LHS and a solvent flowrate corresponding to an action determined by the agent, and results in the API concentration in the extract stream (top product of the LLE column). The behavior policy is trained per every single step in an episode and the target policy is updated at the end of the last step (i.e., prior to starting the next episode). It is of great importance to the agent to start training its policies in a way to use the experience replay memory, which indicates that experiences should be considerably accumulated to a certain extent.

Fig. 6 depicts variations of the total score in an increase of the number of episodes. Increase in the score implies that the control performance is improving and achieving its target, which is the desired outcome. As assuming that a single episode consists of 20 steps in the previous section, the maximum positive score of one episode is 20 (e.g., the total score of 10 explains that the controlled API concentration values by the DQN model in 15 times of 20 steps are greater than the setpoint, otherwise, in 5 times of 20 steps, either those are less than the setpoint or the LLE column does not achieve any convergence). Training procedure of the DQN-based control system stops when it satisfies following conditions: 1) the average score of the latest 30 episodes is greater than 15 and 2) each score of each episode is greater than 10.

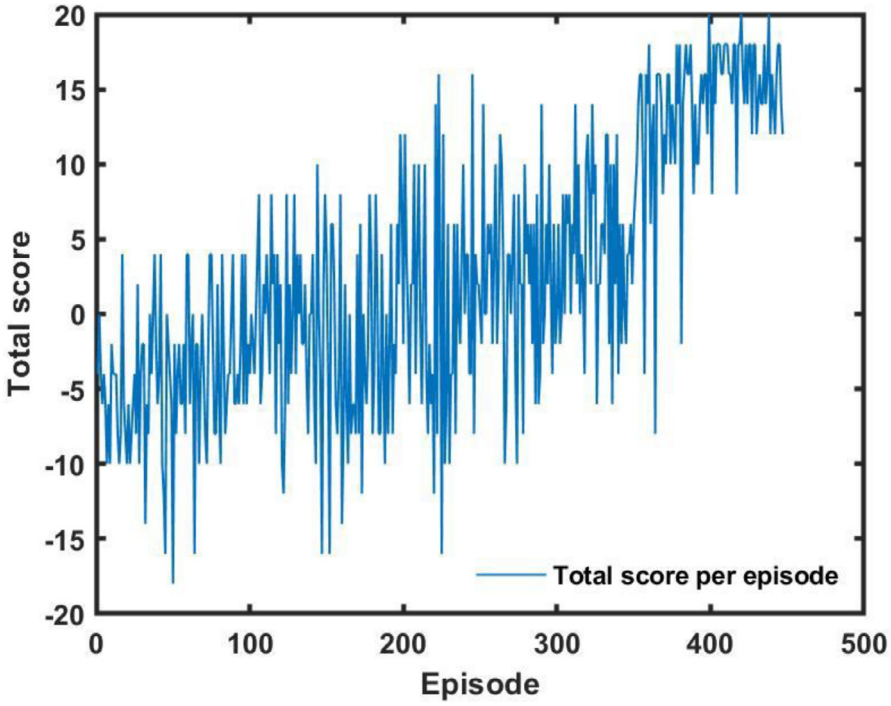


Fig. 6. Score fluctuation according to an increase of episodes in the middle of the DQN-based control system training.

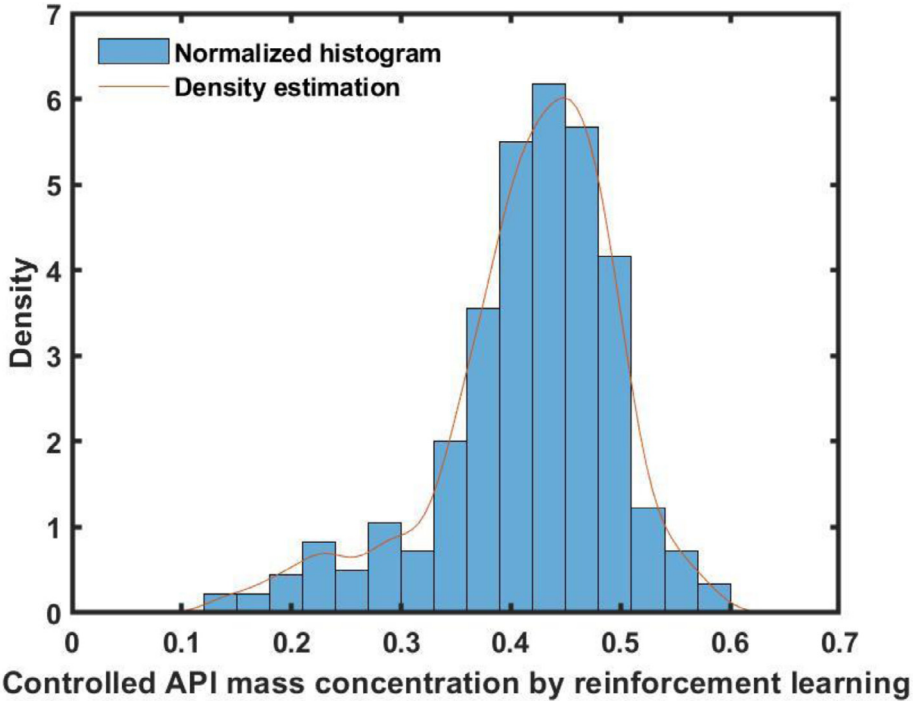


Fig. 7. Controlled API concentration distribution by the optimal control system based on the DQN model.

Large variations with negative scores take place at the beginning of the episode scale because solvent flowrates are randomly selected until the first 50 episodes to prepare for the training: experiences during the first 50 episodes are stored in the experience replay memory then learning Q-values in the policies of the agent starts. A number of API concentration values less than the setpoint can be occurred in this preparation stage similar to open loop operation with random parameters.

The reason negative scores are often allocated after starting training is related to the ε-greedy policy used for the DQN, which manages trade-off between exploitation and exploration in a probabilistic way. The ε-greedy keeps on generating random actions at a certain probability without depending on the policy in the agent. In this study, the more the trained episodes are accumulated, the smaller that random action probability gets since the ε value is set to gradually decrease as the episode scale progresses (i.e., ex-

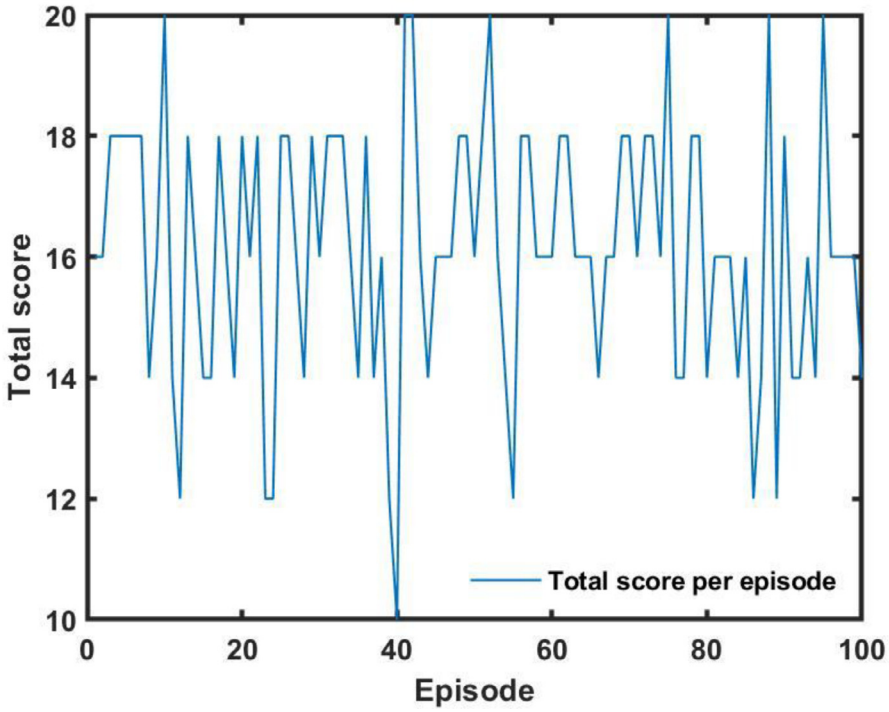


Fig. 8. Score results of the DQN-based control system using test datasets.

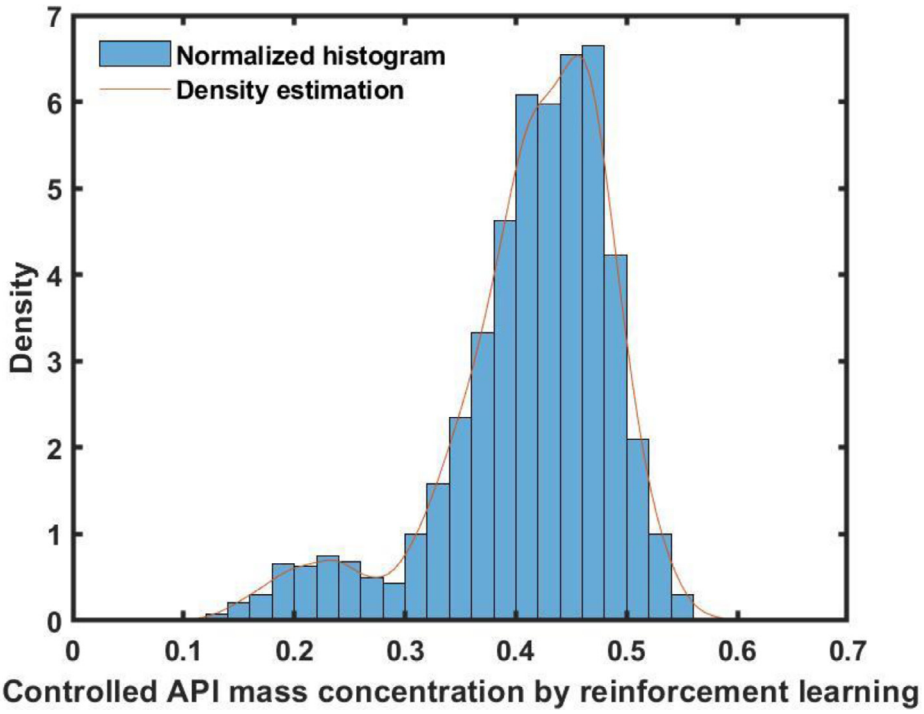


Fig. 9. DQN-based API concentration distribution using test datasets.

ploration decreases in favor of exploitation) (Table 2). The general flow of the total score is significantly increased at around the 400th episode, and the high scores are continuously maintained at the end of training (Fig. 6).

The distribution of the API concentration values controlled by the DQN-based control system is illustrated in Fig. 7. The average and the standard deviation of the controlled API concentration values are now 0.41 and 0.08, respectively. There exists the range of small API concentration values. However, most of API concentration values are distributed near the average, which is approximately increased by 32 % compared to the average of the API concentration distribution from MC simulation under open loop conditions. The standard deviation of the controlled API concentration values is reduced by around 23 % compared to the previous standard deviation from MC simulations under open loop conditions. These outcomes demonstrate the feasibility of developing and training RL algorithm using simulation data and samples from MC sampling method to operate in a stable manner an LLE column.

The trained control system is evaluated by using test data. New sample datasets of 100 episodes (i.e., equal to 2,000 steps) are arbitrarily extracted from sample datasets unused in the training datasets. The results explain that about 16 positive score is on average (Fig. 8). This specifies that in the total of 20 operations, 18 times of API concentration values in the extract stream are greater than the setpoint, otherwise, only 2 times of those are less than the setpoint or fail to converge. The distribution of API concentration values from the trained control system using test datasets is presented in Fig. 9, which says that the average and the standard deviation are 0.41 and 0.077. The entire shape of the distribution in Fig. 9 is similar to that in Fig. 7. Indeed, results from using test datasets validate performance of the optimal control system under different process operation conditions.

We believe that the proposed framework and its implementation indicate the feasibility of emerging AI concepts (especially deep RL algorithms) for optimizing operation in downstream processing and other process engineering applications. Looking ahead, further validation and testing of the proposed methods are needed using pilot and production facility platforms as well as benchmarking against other established industry operation practice to better identify and match industrial needs. In particular, further research is required to identify the specific needs and constraints of industrial applications, assess computational costs and improve numerical efficiency, and enhance the accuracy of modelling among others and its validation. In that regard, hybrid-modelling concept using both model-based and data-driven techniques is worth considering especially where there is lack of mechanistic models

6. Conclusions

We have presented a control framework based on the DQN algorithm and applied the developed model for the downstream separation in biopharmaceutical processes. Major challenges related to the lack of historical data have been solved by employing MC sampling method. The model-free and off-policy DQN algorithm has been designed and trained successfully to control a separation process. An interface based on component object model has been integrated in the proposed model to provide a vital link between different software, each of which is MATLAB for the sampling method, Aspen Plus™ for simulation runs, and Python for the main algorithm of the DQN. A prototype software is produced and shared on Mendeley Data to further facilitate the application of the DQN algorithm.

A case study of the LLE column of downstream separation in biopharmaceutical process has been accomplished. The LHS method has generated a number of suitable sample datasets considering disturbances and MC simulation runs under open loop conditions have been conducted to determine the setpoint of the DQN-based control system in the LLE column. Results from the case study state that the developed control model has improved the API concentration distribution in terms of the system robustness, showing an increase of the average by 32 % and a decrease of the standard deviation by 23 % compared to the results from open loop operation. Future works focus on industrial/pilot scale validation and refinement of the proposed control framework, and extend the application to a virtual plant encompassing other processes from fermentation to ultrafiltration, extraction column, and crystallization processes.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Soonho Hwangbo: Data curation, Formal analysis, Conceptualization, Methodology, Resources, Software, Validation, Visualization, Writing - original draft, Writing - review & editing. **Gürkan Sin:** Conceptualization, Funding acquisition, Investigation, Project administration, Writing - review & editing.

Acknowledgment

We would like to thank the Danish Council for Independent Research (DFF) for financing the project under the grant ID: DFF-6111600077B.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.compchemeng.2020.106910.

References

Ahn, K.U., Park, C.S., 2020. Application of deep Q-networks for model-free optimal control balancing between different HVAC systems. Sci. Technol. Built Environ. 26, 61–74.
Al-Jabery, K., Xu, Z., Yu, W., Wunsch, D.C., Xiong, J., Shi, Y., 2016. Demand-side management of domestic electric water heaters using approximate dynamic programming. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 36, 775–788.
Arif, A., Babar, M., Ahamed, T.I., Al-Ammar, E., Nguyen, P., Kamphuis, I.R., Malik, N., 2016. Online scheduling of plug-in vehicles in dynamic pricing schemes. Sustain. Energy Grids Netw. 7, 25–36.
Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A., 2017. Deep reinforcement learning: a brief survey. IEEE Signal Process. Mag. 34, 26–38.
Bellman, R., 1966. Dynamic programming. Science 153, 34–37.
Berlink, H., Kagan, N., Costa, A.H.R., 2015. Intelligent decision-making for smart home energy management. J. Intell. Robot. Syst. 80, 331–354.
Bock, S., Goppold, J., & Weiß, M. (2018). An improvement of the convergence proof of the ADAM-Optimizer. arXiv preprint arXiv:1804.10587.
Buyel, J., Twyman, R., Fischer, R., 2015. Extraction and downstream processing of plant-derived recombinant proteins. Biotechnol. Adv. 33, 902–913.
Chen, C.-C., Song, Y., 2004. Solubility modeling with a nonrandom two-liquid segment activity coefficient model. Ind. Eng. Chem. Res. 43, 8354–8362.
Cheng, Z., Zhao, Q., Wang, F., Jiang, Y., Xia, L., Ding, J., 2016. Satisfaction based Q-learning for integrated lighting and blind control. Energy Build. 127, 43–55.
De Somer, O., Soares, A., Vanthournout, K., Spiessens, F., Kuijpers, T., Vossen, K., 2017. Using reinforcement learning for demand response of domestic hot water buffers: A real-life demonstration. In: Proceedings of the IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), pp. 1–7 IEEE.
Djamaluddin, I., Mitani, Y., Esaki, T., 2011. Evaluation of ground movement and damage to structures from Chinese coal mining using a new GIS coupling model. Int. J. Rock Mech. Min. Sci. 48, 380–393.
Formentin, S., De Filippi, P., Corno, M., Tanelli, M., Savaresi, S.M., 2011. Data–driven design of braking control systems. IEEE Trans. Control Syst. Technol. 21, 186–193.
Gulli, A., & Pal, S. (2017). Deep learning with Keras: Packt Publishing Ltd.
He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034.
Heinzle, E., Biwer, A.P., Cooney, C.L., 2007. Development of Sustainable Bioprocesses: Modeling and Assessment. John Wiley & Sons.
Hjalmarsson, H., 2002. Iterative feedback tuning—an overview. Int. J. Adapt. Control signal process. 16, 373–395.
Hou, Z.-S., Wang, Z., 2013. From model-based control to data-driven control: Survey, classification and perspective. Inf. Sci. 235, 3–35.
Hou, Z., Jin, S., 2010. A novel data-driven control approach for a class of discrete–time nonlinear systems. IEEE Trans. Control Syst. Technol. 19, 1549–1558.
Hou, Z., Jin, S., 2013. Model Free Adaptive Control: Theory and Applications. CRC Press.
Houk, J.C., Davis, J.L., Beiser, D.G., 1995. Models of Information Processing in the Basal Ganglia. MIT Press.
Howard, R. A. (1960). Dynamic programming and markov processes.
Hwangbo, S., & Sin, G. (2019). Integration of Monte-Carlo sampling method and deep-Q-learning network via COM-based interface between MATLAB/Python and Aspen Plus. In. Mendeley: 10.17632/rz2zj86yzp.1.
Irwin, G. W., Irwin, G. W., Warwick, K., & Hunt, K. J. (1995). Neural network applications in control: Iet.
Jiménez-González, C., Woodley, J.M., 2010. Bioprocesses: modeling needs for process evaluation and sustainability assessment. Comput. Chem. Eng. 34, 1009–1017.
Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. J. Artif. Intell. Res. 4, 237–285.

Konidaris, G., Osentoski, S., & Thomas, P. (2011). Value function approximation in reinforcement learning using the Fourier basis. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.
Lee, J.H., 2014. From robust model predictive control to stochastic optimal control and approximate dynamic programming: A perspective gained from a personal journey. Comput. Chem. Eng. 70, 114–121.
Lee, J.H., Lee, J.M., 2006. Approximate dynamic programming based approach to process control and scheduling. Comput. Chem. Eng. 30, 1603–1618.
Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909.
Liu, J., Peng, C., Dang, Q., Apps, M., Jiang, H., 2002. A component object model strategy for reusing ecosystem models. Comput. Electron. Agric. 35, 17–33.
McKay, M.D., Beckman, R.J., Conover, W.J., 1979. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21, 239–245.
McPartland, T.J., Patil, R.A., Malone, M.F., Roberts, S.C., 2012. Liquid–liquid extraction for recovery of paclitaxel from plant cell culture: solvent evaluation and use of extractants for partitioning and selectivity. Biotechnol. Progr. 28, 990–997.
Mišković, L., Karimi, A., Bonvin, D., Gevers, M., 2007. Correlation-based tuning of decoupling multivariable controllers. Automatica 43, 1481–1494.
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., 2015. Human-level control through deep reinforcement learning. Nature 518, 529.
Mocanu, E., Nguyen, P.H., Kling, W.L., Gibescu, M., 2016. Unsupervised energy prediction in a smart grid context using reinforcement cross-building transfer learning. Energy Build. 116, 646–655.
Molla, G.S., Freitag, M.F., Stocks, S.M., Nielsen, K.T., Sin, G.r., 2019. Solubility prediction of different forms of pharmaceuticals in single and mixed solvents using symmetric electrolyte nonrandom two-liquid segment activity coefficient model. Ind. Eng. Chem. Res. 58, 4267–4276.
Peters, J., Schaal, S., 2008. Natural actor-critic. Neurocomputing 71, 1180–1190.
Petsagkourakis, P., Sandoval, I.O., Bradford, E., Zhang, D., del Rio-Chanona, E.A., 2020. Reinforcement learning for batch bioprocess optimization. Comput. Chem. Eng. 133 106649.
Powell, W.B., 2009. What you should know about approximate dynamic programming. Naval Res. Logist. (NRL) 56, 239–249.
Qi, X., Wu, G., Boriboonsomsin, K., Barth, M.J., Gonder, J., 2016. Data-driven reinforcement learning–based real-time energy management system for plug-in hybrid electric vehicles. Transp. Res. Rec. 2572, 1–8.
Radac, M.-B., Precup, R.-E., 2019. Data-Driven model-free tracking reinforcement learning control with VRFT-based adaptive actor-critic. Appl Sci. 9, 1807.
Raju, L., Sankar, S., Milton, R., 2015. Distributed optimization of solar micro-grid using multi agent reinforcement learning. Procedia Comput. Sci. 46, 231–239.
Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.
Rocchetta, R., Bellani, L., Compare, M., Zio, E., Patelli, E., 2019. A reinforcement learning framework for optimal operation and maintenance of power grids. Appl. Energy 241, 291–301.
Seader, J. D., Henley, E. J., & Roper, D. K. (1998). Separation process principles.
Sekizaki, S., Hayashida, T., Nishizaki, I., 2015. An intelligent home energy management system with classifier system. In: Proceedings of the IEEE 8th International Workshop on Computational Intelligence and Applications (IWCIA), pp. 9–14 IEEE.
Selvi, D., Piga, D., Bemporad, A., 2018. Towards direct data-driven model-free design of optimal controllers. In: Proceedings of the European Control Conference (ECC), pp. 2836–2841 IEEE.
Shin, J., Badgwell, T.A., Liu, K.-H., Lee, J.H., 2019. Reinforcement Learning–Overview of recent progress and implications for process control. Comput. Chem. Eng. 127, 282–294.
Sin, G., Gernaey, K.V., Lantz, A.E., 2009. Good modeling practice for PAT applications: propagation of input uncertainty and sensitivity analysis. Biotechnol. Progr. 25, 1043–1053.
Singh, S., Jaakkola, T., Littman, M.L., Szepesvári, C., 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. Mach. Learn. 38, 287–308.
Spall, J.C., Cristion, J.A., 1998. Model-free control of nonlinear stochastic systems with discrete-time measurements. IEEE trans. Autom. Control 43, 1198–1210.
Strube, J., Grote, F., Josch, J.P., Ditz, R., 2011. Process development and design of downstream processes. Chemie Ingenieur Technik 83, 1044–1065.
Sun, B., Luh, P.B., Jia, Q.-S., Yan, B., 2015. Event-based optimization within the Lagrangian relaxation framework for energy savings in HVAC systems. IEEE Trans. Autom. Sci. Eng. 12, 1396–1406.
Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction: MIT press.
Sutton, R.S., Maei, H.R., Szepesvári, C., 2009. A convergent $ o (n) $ temporal-difference algorithm for off-policy learning with linear function approximation. Adv. Neural inf. Process. Syst. 1609–1616.
Tan, Z., Zhang, X., Xie, B., Wang, D., Liu, B., Yu, T., 2018. Fast learning optimiser for real-time optimal energy management of a grid-connected microgrid. IET Gener. Transm. Distrib. 12, 2977–2987.
Tanaskovic, M., Fagiano, L., Novara, C., Morari, M., 2017. Data-driven control of nonlinear systems: an on-line direct approach. Automatica 75, 1–10.
Taylor, M.E., Whiteson, S., Stone, P., 2006. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1321–1328 ACM.
Wang, Y., Velswamy, K., Huang, B., 2017. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. Processes 5, 46.
Watkins, C.J.C.H. (1989). Learning from delayed rewards.
Weatherley, L.R., 2013. Engineering Processes for Bioseparations. Elsevier.
Wiering, M., Van Otterlo, M., 2012. Reinforcement learning. Adapt. Learn. Optim. 12, 3.
Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8, 229–256.
Yang, L., Nagy, Z., Goffin, P., Schlueter, A., 2015. Reinforcement learning for optimal control of low exergy buildings. Appl. Energy 156, 577–586.