Computers and Chemical Engineering 143 (2020) 107077

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Real-time optimization using reinforcement learning

By Kody M. Powell[a,b,*], Derek Machalek[a], Titus Quah[a]

[a] *Department of Chemical Engineering, University of Utah, 50 S. Central Campus Drive, Salt Lake City, UT 84112, USA*
[b] *Department of Mechanical Engineering, University of Utah, 1495 E 100 S, Salt Lake City, UT 84112, USA*

ARTICLE INFO

Article history:
Received 28 May 2020
Revised 28 July 2020
Accepted 23 August 2020
Available online 25 August 2020

ABSTRACT

This work introduces a novel methodology for real-time optimization (RTO) of process systems using reinforcement learning (RL), where optimal decisions in response to external stimuli become embedded into a neural network. This is in contrast to the conventional RTO methodology, where a process model is solved repeatedly for optimality. This reinforcement learning real-time optimization methodology (RL-RTO) utilizes an actor-critic architecture similar to that being used in dynamic control research. However, the methodology presented here is purely for steady-state optimization, which is a novel feature of this work. This work also presents a novel, hybrid training methodology, where a gradient-based optimization solver is used for the training the value network (or critic) and a meta-heuristic optimization algorithm (particle swarm optimization or PSO) is used for training the policy network (or actor). Using this novel training algorithm, the neural networks representing the RL application can be updated in real-time or by using a batch-online training methodology. This technique allows for a solver to utilize the entire data set and attempt to find a global optimum, rather than by taking smaller, incremental update steps after each new data point is collected. As the process system runs and more data becomes available, the critic and the actor networks can be updated in sequence so that the RL-RTO application continually updates itself and gets closer to approaching true optimality. A process system (a chemical reactor) is used as a demonstration case study and also to compare the performance of RL-RTO to a conventional RTO methodology, which uses a near-perfect first principles model of the system, combined with a nonlinear programming (NLP) optimization technique. Each of these methods is compared to a brute force operational methodology in which the system's product throughput is maximized. The RL-RTO application demonstrates promise, as it improves the reactor's annual profit by 9.6%. By comparison, the first principles plus NLP method improves the profit by 17.2%. These RL-RTO results, while promising, indicate that there is still more development needed for RL-RTO to be a viable competitor to the conventional methods.

1. Introduction

As computing technology advances, reinforcement learning (RL) techniques are rapidly finding new applications. RL applications can be used to train artificial intelligence (AI) agents to choose the best actions, given their current state in relation to their environment. Prominent examples of RL applications include training AI agents to successfully navigate video games by processing individual pixels in order to make assessments of the agent's state within its environment so that the agent can "learn" the best way to respond in every scenario. With the rise in deep learning, where multi-layer neural networks leverage unprecedented computing power, these methods have proven to meet and even exceed human-level performance at some of the aforementioned tasks (Mnih et al., 2015). While video games are manmade and controlled environments, developments in RL techniques are finding their way into the physical world. They are being used to help robots learn how to interact with their environment for the completion of certain tasks in manufacturing applications (Li et al., Jun. 2019, Tsurumine et al., Feb. 2019), how to help autonomous vehicles learn how to drive in energy efficient ways (Qi et al., Feb. 2019), and even how to make traffic systems run more smoothly with autonomous vehicles (You et al., Apr. 2019). Generally speaking, the development of deep learning techniques has allowed for a rapid expansion of the field of RL to new applications. One of the predominant reasons for this is that many engineering systems require the solution space to be continuous. The onset of deep neural networks allow for the RL agent to retain much more information about the system and have it explore nearly infinite combinations of state and action spaces, rather than only a small subset of possible actions as in the purely discrete case (Lillicrap et al., 2016).

* Corresponding author.
E-mail address: kody.powell@utah.edu (B.K.M. Powell).

https://doi.org/10.1016/j.compchemeng.2020.107077
0098-1354/© 2020 Elsevier Ltd. All rights reserved.

Nomenclature

Variable	Value/Range	Units	Description
Decision Variables			
Q_B	1-8	m^3/min	**Flow of Reactant B** Perturbed every 2 hours w/ uniform distribution during critic training. Optimal values determined by actor network after training
Q	0-12	GJ/min	**Reactor Heat Input** Perturbed randomly every 2 hours w/ uniform distribution during critic training. Optimal values determined by actor network after training
Measured Disturbance Variables			
$\$_A$	2-20	$\$/m^3$	**Price of Component A** Perturbed randomly every 4 hours w/ uniform distribution
$\$_B$	5-50	$\$/m^3$	**Price of Component B** Perturbed randomly every 4 hours w/ uniform distribution
$\$_C$	2-20	\$/kmol	**Price of Component C** Perturbed randomly every 4 hours w/ uniform distribution
$\$_q$	0-20	\$/GJ	**Price of Heat** Perturbed randomly every 4 hours w/ uniform distribution
Unmeasured Disturbances			
$C_{A,0}$	15 ± 0.5	$kmol/m^3$	**Component A Inlet Concentration** Perturbed randomly every 5 minutes w/ Gaussian distribution using variance shown
$C_{B,0}$	15 ± 0.5	$kmol/m^3$	**Component B Inlet Concentration** Perturbed randomly every 5 minutes w/ Gaussian distribution using variance shown
T_A	500 ± 40	K	**Reactant A Inlet Temperature** Perturbed randomly every 5 minutes w/ Gaussian distribution using variance shown
T_B	500 ± 40	K	**Reactant B Inlet Temperature** Perturbed randomly every 5 minutes w/ Gaussian distribution using variance shown
Other manipulated and intermediate variables			
H	8 (nominal)	m	**Reactor Level** Controlled variable with fixed set point
Q_A	1-8	m^3/min	**Flow of Reactant A** Manipulated variable used in tank level control
Q_{out}	9.05 (nominal)	m^3/min	**Reactor Outlet Flow**
R_{rxn}	Variable	$kmol/(min{*}m^3)$	**Reaction Rate Per Unit Volume**
State Variables			
C_A	Variable	$kmol/m^3$	**Reactant A Concentration in Reactor**
C_B	Variable	$kmol/m^3$	**Reactant B Concentration in Reactor**
C_C	Variable	$kmol/m^3$	**Reactant C Concentration in Reactor**
T	Variable	K	**Reactor Temperature**
V	Variable	m^3	**Volume of Liquid in Reactor**
Model Parameters			
C	3.2	$m^3/(m^{0.5}{*}min)$	**Reactor Valve Constant**
c_p	3.25	$kJ/kg{*}K$	**Heat Capacity of All Liquids**
D	4	m	**Reactor Diameter**
E_A	9500	kJ/kmol	**Reaction Activation Energy**
ΔH_{rxn}	156,000	kJ/kmol	**Heat of Reaction**
k_0	0.23	$m^3/(kmol{*}min)$	**Reaction Rate Law Pre-Exponential Constant**
ρ	780	kg/m^3	**Density of All Liquids**
R	8.314	$kJ/(kmol{*}K)$	**Ideal Gas Constant**

In the fields of process systems engineering (PSE) and energy systems engineering, researchers are gradually discovering new ways to apply RL to *potentially* replace more conventional methods of controlling or optimizing systems. Shin et al. gave an overview of recent progress in reinforcement learning in the process systems space for process control. In this work, they discuss some of the advantages and disadvantages of RL as compared to model predictive control (MPC) and recent progress with deep RL methods and how they can be applied in this space. Compared to MPC, RL methods have strengths and weaknesses. Strengths include requiring only a single evaluation of the function representing the RL agent to generate prescribed control actions, the ability to simultaneously explore and exploit the function, and an adaptive function that learns through trial and error. RL methods, however, also possess many weaknesses that, in their current state of development, make them less ideal for use in real control systems. These weaknesses include no consideration of closed-loop stability of the system, no ability to constrain process states, and no consideration of the trajectory through the state space to the end goal (Shin et al., Aug. 2019). Another major challenge in utilizing RL-based control methodologies is that their results are difficult or impossible for a human to completely comprehend. While that is the basic premise of machine learning (i.e., that a computer will learn and understand the results so that humans do not need to), ultimately, humans will need to make the final decision on control inputs in a real plant. Qin and Chiang argue that, until results from RL applications pass the test of interpretability, they will not find widespread adoption in the process industries (Qin and Chiang, Jul. 2019).

1.1. Regulatory control

In process and energy systems, one application of RL is for regulatory control, where an RL agent replaces a more conventional control algorithm. For example, Wang et al. proposed replacing conventional feedback control with a deep RL methodology. Their method uses an actor-critic architecture with artificial neural networks (ANNs) acting as the critic, or value function (whose role is to predict the expected reward based on inputs), as well as the actor, or policy function (whose role is to determine the optimal inputs, based on the state/environment of the agent). They found that the RL methodology compared favorably in performance to traditional PI control as well as a linear quadratic regulator (LQR) (Wang et al., Jan. 2018). Pandian and Noel developed a partially supervised reinforcement learning algorithm to control a bioreactor (Pandian and Noel, Sep. 2018). Syafiie et al. successfully demonstrated model-free control using RL for neutralization processes. This methodology employed Q-learning and used an actual neutralization experiment to demonstrate the effectiveness of RL for automatic control (Syafiie et al., Sep. 2007). Syafiie et al. also applied a similar methodology for successful control of the oxidation process in wastewater treatment plants (Syafiie et al., Jan. 2011). Ma et al. applied deep reinforcement learning with a deep deterministic policy gradient (DDPG) to a polymerization process for continuous control (Ma et al., Mar. 2019). Shah and Gopal developed an RL methodology to replace model predictive control (MPC) with model-free predictive control (MFPC). This methodology uses Q-learning combined with fuzzy inference systems to allow the controller to specify the next single control move. One key benefit of this methodology is that it does not require the development of a predictive model of the plant, as the controller's parameters are instead determined by direct interaction with the plant (Shah and Gopal, Jan. 2016). Morinelly and Ydstie developed a dual MPC methodology with reinforcement learning and demonstrated that RL converges to an optimal policy for a discrete time linear plant (Morinelly and Ydstie, Jan. 2016). Görges compared MPC to RL for regulatory control for problems that are linear, time-invariant, and have a quadratic cost function and determined that RL methods, while still needing much development, have potential for adaptive MPC with guaranteed stability and feasibility (Görges, Jul. 2017).

1.2. Optimization

Beyond regulatory control, RL has also found its way into optimization applications, where the RL agent's role is to maximize system performance, rather than merely maintain set points. Jia et al. developed an RL-based control scheme for optimal control of buildings in an effort to reduce energy consumption. Their application included experience replay and expert policy to help guide the RL application to learn more quickly from existing best practices (Jia et al., Feb. 2019). Similarly, Valladares et al. applied RL to minimize energy usage for a building system and incorporated indoor air quality into their formulation. Their results show 10% lower CO_2 levels and 4-5% less energy consumption than the building's existing control system (Valladares et al., May 2019). RL-based methods have also been used to control renewable generation systems, integrated with building systems, showing a 10% improvement over rules-based methods (Yang et al., Oct. 2015).

In the power generation industry, RL has been used to minimize NO_X and maximize efficiency of coal-fired power generation plants (Cheng et al., Sep. 2018). Conventionally, this process has historically been optimized by a combination of supervised learning (ANN) models, that are iterated on by separate optimizer, using algorithms such as particle swarm optimization, to find the points of minimum NO_X, maximum efficiency, or some combination of the two (Safdarnejad et al., May 2019, Tuttle et al., Dec. 2019, Tuttle et al., 2020). As RL is capable of dealing with process dynamics, it has some features that may be able to help power plants better smooth load transitions, while still pushing towards optimal transient operation, which is an issue that many thermal power plants find themselves dealing with, given the growing prevalence of intermittent renewables (Tuttle and Powell, May 2019, Safdarnejad et al., May 2019). At the grid-level scale, RL has also been applied to the economic dispatch problem, where grid operators will seek to find the best combination of power plants to use to meet demand, while also managing the environmental impact of the fleet, collectively (Bora et al., Jan. 2019). These, and similar applications, better help grid operators know which units to commit to power generation and better enable the inclusion of renewable energy resources on the grid (Zhou et al., Dec. 2018), (Wang et al., Jul. 2019) and to optimally control and maintain the electric grid in general (Rocchetta et al., May 2019). In addition to generation dispatch problems, RL methods are also being explored for their utility in demand response problems, where the demand side of the grid can utilize their operational flexibility to help better regulate the electric grid (Vázquez-Canteli and Nagy, Feb. 2019).

1.3. Contributions of this work

While there are a limited number of systems-level optimization applications for RL being explored, few of them are focused on process industries. Real-time optimization (RTO) in particular is a prevalent technology in the process industries and is gaining attention in power generation and energy systems (Rashid et al., Jun. 2018, Rashid et al., May 2019, Powell et al., 2016, Diehl et al., Jun. 2002, Pedersen and Hoffman, Jun. 1997, Blackburn et al., 2019, Blackburn et al., 2020). There is little information in the literature on using RL for economic RTO and particularly for the process industries. In process systems, the research in using RL to enhance operation is focused on dynamic, regulatory control issues, as noted from the focus on MPC in the literature review above. In contrast to MPC, the focus of RTO is to guide a complex system from an economic point of view to respond in optimal ways to externalities, such as the economic environment (e.g., commodity prices) or ambient conditions. Although dynamic RTO (D-RTO) is a subset of RTO that considers the trajectory of the system in the economics (Machalek et al., 2020), the vast majority of RTO applications use a steady-state model. Although steady-state RTO deviates somewhat from the underlying principles that led to the creation of reinforcement learning, such as its foundations in Markov Processes and dynamic programming (Szepesvári, 2010), the methodologies available now for RL, such as deep neural networks and actor-critic methods are equally applicable in the space of steady-state optimization. While this methodology deviates substantially in that it does not consider process or system dynamics, there is great potential in exploring RL for RTO applications.

The major contributions of this work include the following:

- This work outlines a methodology for solving real-time optimization problems using reinforcement learning, which is novel and a fundamentally different from any conventional RTO methodology. Rather than iterating on a process model to find optimal solutions in real time, as in the conventional RTO application, this reinforcement learning-based RTO (RL-RTO) methodology trains a deep neural network to output optimal solutions, when given the environmental variables describing the context of the system in a single function pass. This also means that, unlike conventional RTO, RL-RTO does not require a first principles model of the plant, but instead learns from contextual data (environmental conditions and plant inputs) the optimal way to respond, which becomes embedded within the application's policy network.

- A novel, hybrid training methodology is used, where a gradient-based solver is used to train the value network for best fit in value prediction and a meta-heuristic algorithm (particle swarm optimization) is used to train a policy network for optimal output.
- As opposed to updating the value and policy models at each time step with small, gradient-based steps, this work presents a batch-online training methodology. This allows the two networks to use a large collection of data so that it can seek a global optimum, particularly for the policy network, which leverages the global PSO solver.
- To shed more light on the types of outputs that the RL network produces, output plots across a range of inputs are produced and compared to the same plots produced from a conventional nonlinear programming-based method. This work demonstrates that, while not identical, these output plots for the two different methods are similar in shape, indicating that RL captures most of the behavior of a nonlinear programming (NLP) method that uses a near perfect steady-state model of the system.
- The results of RL-RTO for a year of operation over the full range of environmental conditions (in this case commodity prices) are compared to an NLP-based method and also a maximum-throughput operating condition. Results indicate that RL-RTO improves over the maximum-throughput operation by 9.6%, although not to the extent of the NLP method (17.2%).
- One advantage of RL-RTO is that, unlike conventional RTO, which utilizes a first principles model coupled with a nonlinear programming solver, it requires only a single function evaluation of the policy network. This results in reducing computation time by 87.7% compared to the NLP-based method. It must be noted that, while online computation is much faster, there is still significant offline computation in training the RL agent.
- One disadvantage of RL-RTO vs. conventional RTO methods is that process constraints cannot be enforced directly. However, this work presents a methodology for enabling constraints on decision variables by adding a penalty term to the reward function during training of the policy network.

2. Methods

2.1. Overview

The overall goal of developing an RL-based control or optimization strategy is to train a function to receive information about the agent's current state or environment and have that function output optimal control moves based on the current state or environment. Depending on the current state (S_t) and the predicted reward (R_t), the RL agent will determine the appropriate action to take (A_t). The policy function (π) would output the appropriate action for the agent to take, given the current state.

$$A_t = \pi(S_t, \theta_\pi) \tag{1}$$

where the function π is assumed to contain fitting parameters, denoted θ_π, which contain information that is "learned" over time by the policy function.

After calculating the action, A_t, these control moves would then be applied to the actual system in its environment, resulting in the system transitioning to the next state (S_{t+1}) and producing the corresponding reward (R_{t+1}). The agent's policy function would take this new state information and use it to produce the next action, and so on.

While the description above describes the operation of an RL agent in practice, it does not answer the question of how to construct the policy function (π). There are several methodologies for developing a policy for the system. These are categorized and discussed in more detail in (Shin et al., Aug. 2019). These methods include: (1) Model-Based, (2) Value-Based, (3) Policy-Gradient, and (4) Actor-Critic methods. This work focuses on the Actor-Critic methodology, which leverages a value function that predicts the reward when given a certain action within the context of a particular state or environment. This value function (υ) predicts the reward (R_{t+1}), when given state (S_t) variables and when taking a certain set of actions (A_t), and is also dependent on a set of fitting parameters (θ_υ) wherein the machine-learned behavior is embeded.

$$R_{t+1} = \upsilon(S_t, A_t, \theta_\upsilon) \tag{2}$$

The value function (also referred to as the "critic"), once trained, can be used as a process approximator, over which the policy function (π) can be trained. The basic methodology for online training is shown in Fig. 2. As the reward is measured from the actual system, this information is fed to the value function, or critic. Over time, as more data is collected, the value function is trained. Ideally, it becomes an accurate predictor of the reward when given specific environmental and action data as inputs. The trained value function is then used in an iterative manner as the update algorithm or solver iterates to find the optimal parameters for the policy function (π) that result in producing optimal actions as a function of the current state (see Eq. 1).

2.2. Actor/Critic RL methodology

Taking the basic structure proposed in Figs. 1 and 2, but specifically inserting artificial neural networks to be the value (υ) and policy (π) functions, yields the deep neural network structure, shown in Fig. 3. This structure follows the actor/critic methodology where the value network (or critic) is a neural network designed to predict the reward based on environmental, or state variables, and decision variables. The policy network (or actor) has the role of determining what the decision variables will be as a function of the environment or state of the agent. Ultimately, the policy network will take in the environmental variables and determine the appropriate actions to take. It will feed these recommended actions directly to the plant. However, before it can do so, it must rely on the value network during the training phase. The policy network must utilize the value network to produce an expected reward so that it can learn how to maximize that reward.

The remainder of this section elucidates a novel methodology for breaking down the deep actor/critic neural network into two shallow neural networks and conducting batch-online training. It first demonstrates the training of the value network (or critic function) alone using random or historic data on the environmental conditions (state) and the decision variables. Once the value network has been trained, it is connected to the policy network, which is then trained by using the value network to produce the expected reward, as the training solver iterates to find the policy network parameters (θ_π) that maximize this reward.

2.2.1. Value network training

To train the value network (υ), or critic function, a smaller, sub-network is first extracted from the larger actor/critic deep neural network (Fig. 4). This sub-network *is* the value network and, by itself, is a shallow, three-layer perceptron, artificial neural network. This network is set up to have both environmental (or state) variables, as well as the plant's decision variables, entered directly as inputs. It is trained by comparing the predicted reward produced by the network with the actual measured reward from the plant's historical data. This training optimization problem is a regression problem, with the objective of minimizing the error between the

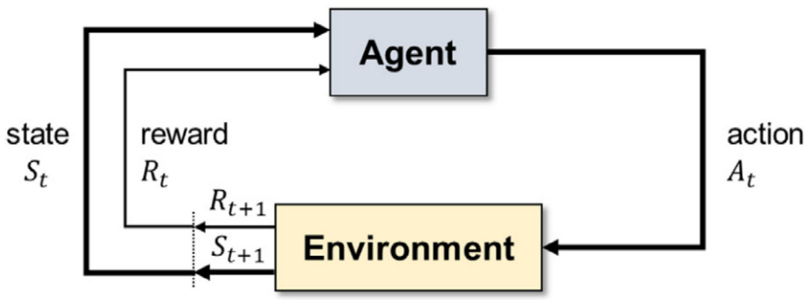


Fig. 1. The basic setup for a reinforcement learning configuration, available from (Shin et al., Aug. 2019).

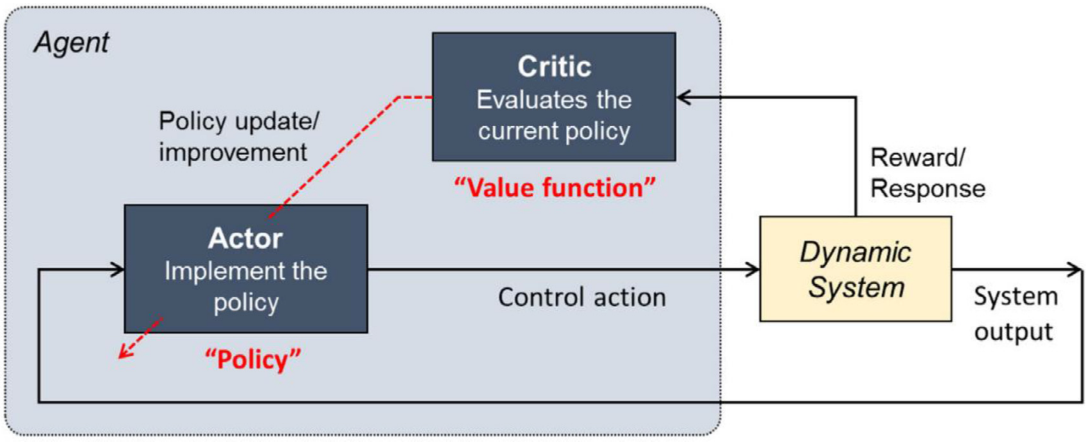


Fig. 2. The Actor-Critic RL design architecture, available from (Shin et al., Aug. 2019).

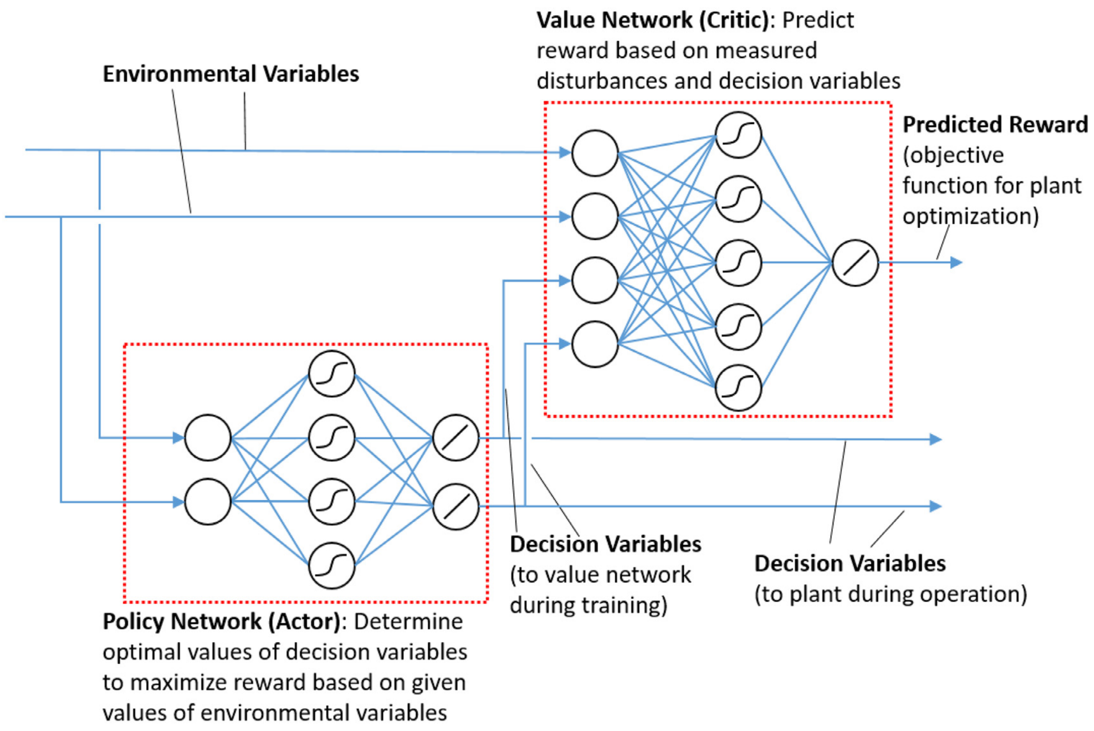

Environmental Variables
Value Network (Critic): Predict reward based on measured disturbances and decision variables
Predicted Reward (objective function for plant optimization)
Decision Variables (to value network during training)
Decision Variables (to plant during operation)
Policy Network (Actor): Determine optimal values of decision variables to maximize reward based on given values of environmental variables

Fig. 3. The deep neural network structure used for reinforcement learning. The model uses a critic function to predict the reward based on input variables (both measured disturbances and decision variables) and an actor function, which will ultimately be trained to determine the optimal values of the decision variables when given a specific set of measured disturbances.

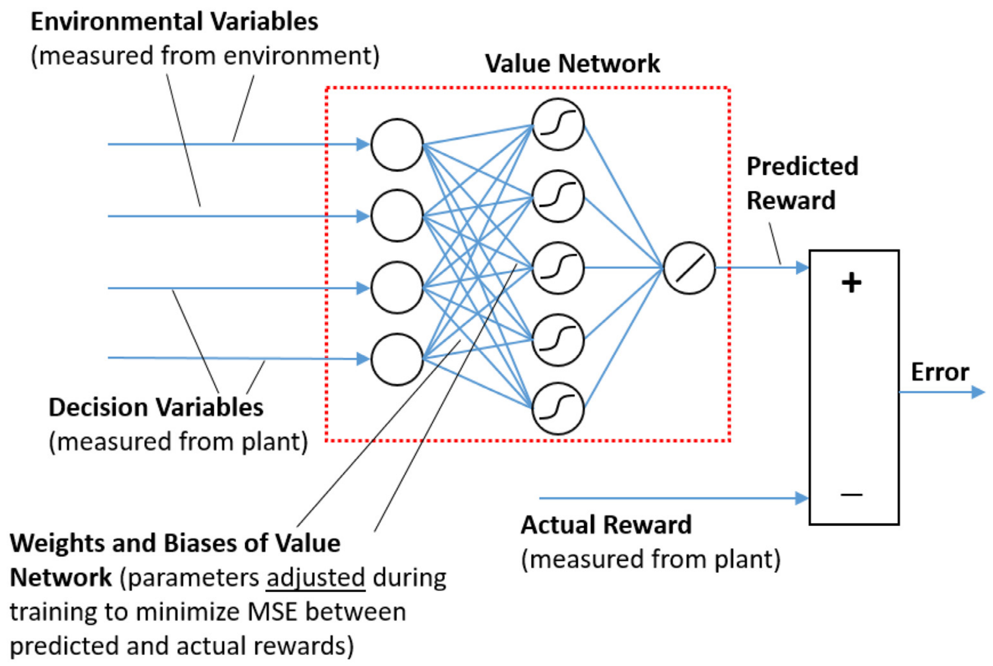

Environmental Variables
(measured from environment)
Value Network
Predicted Reward
+
Error
−
Decision Variables
(measured from plant)
Weights and Biases of Value Network (parameters adjusted during training to minimize MSE between predicted and actual rewards)
Actual Reward
(measured from plant)

Fig. 4. A diagram showing the critic function during training. The critic function is a feed-forward neural network, designed to accurately predict the reward function. The training problem is a regression problem, where the error between the actual reward and the predicted reward is minimized.

value network's output and the measured reward by changing the parameters (biases and weights, represented by θ_υ) of the value network. For a continuous system, such as a process or energy system, conventional back-propagation routines using gradient-based solvers work well for this parameterization problem.

As the training of the policy network will rely heavily on having an accurate value network, it is critical that the value network be adequately trained. The results from the value network must be validated with testing data (i.e., data not used during the model fitting itself) and/or use a cross-validation routine to ensure that the model remains highly accurate even when encountering new, previously unseen, data.

2.2.2. Policy network training

After the value sub-network is trained, it is connected with the larger deep neural network in order to train the policy sub-network. During this phase, the parameters of the value sub-network (θ_υ) are held constant. Historical, measured values of the environmental variables are fed to the network. These are connected to both the value sub-network and the policy sub-network. The policy sub-network's role is to produce optimal decision variables in a manner that results in the maximum reward. However, it must rely on the value sub-network to predict this reward. Therefore, the value network must be in the loop so that the policy network's training function can maximize the expected reward that it predicts.

Unlike the value function's training routine with the objective to minimize error, the policy function's objective is to maximize the expected reward. The training function will take in measured environmental data and iterate on the policy network's parameters (θ_π) so that the policy network produces decision variables that result in maximum reward, as predicted by the value network.

As this is not a conventional regression problem, but rather a reward maximization problem, many conventional back-propagation solvers do not work well for training the policy network. In this work, a non-gradient-based optimization method, particle swarm optimization (PSO) actually proved to be more effective at solving this particular problem. The standard, gradient-based solvers tended to converge rather quickly at local minima, giving solutions that were unsatisfactory and generally resulted in trained policy networks that were ineffective at determining optimal operational policies.

2.2.3. Constraints

One of the key limitations of RL-RTO is that there is no proven methodology for dealing with constraints on input or intermediate variables. As the decision variables are determined solely from the policy network as a function of environmental variables, constraints can be built into this function using penalty methods. Specifically, during training, the reward function can be modified from being one of pure performance optimization (e.g., to maximize profit) to one that adds a penalty term for deviating from normal plant operation (e.g., maximize profit minus a tuned penalty term for having the plant deviate from desired or feasible behavior). An example of enforcing simple input constraints will be given in the case study highlighted in the results section.

2.2.4. Extracting the policy network

As can be observed from Fig. 5, when given the environmental variables as inputs, the policy network directly outputs the optimal decision variables. These can be fed directly to the plant. Depending on the setup of the application, the policy network may ultimately be all that is needed for closed-loop operation of the plant. It can therefore be extracted as a stand-alone function, as depicted in Fig. 6.

However, for continuous, online learning of both the value and policy networks, the best configuration would be to retain the entire actor/critic network, so that the training routines can continuously update these networks with more data as recorded from the plant. This would allow the value network to continuously adjust to incorporate new plant data and account for things such as plant drift. As mentioned previously, the efficacy of the policy network relies on highly accurate reward predictions from the value network, so if the plant is subject to change over time, maintaining and continuously updating these two networks is crucial to the ongoing success of the RL-RTO application.

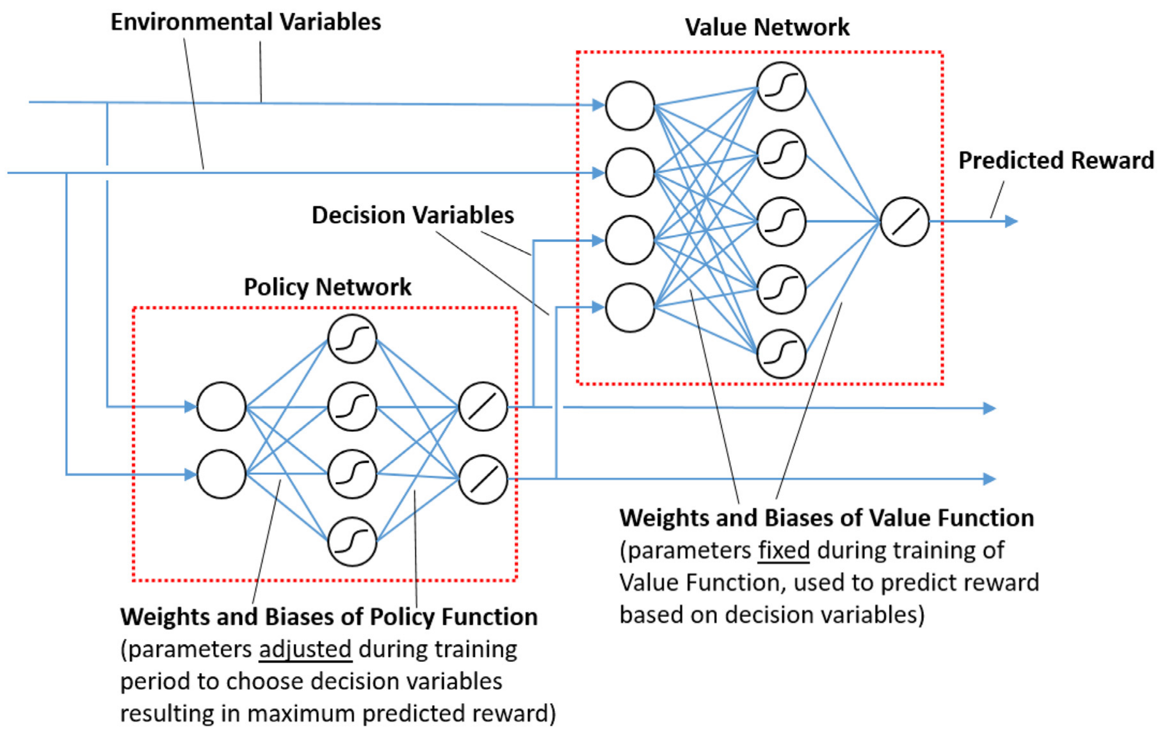

Environmental Variables
Value Network
Predicted Reward
Decision Variables
Policy Network
Weights and Biases of Value Function
(parameters fixed during training of Value Function, used to predict reward based on decision variables)
Weights and Biases of Policy Function
(parameters adjusted during training period to choose decision variables resulting in maximum predicted reward)

Fig. 5. During training of the actor function, the actor network's parameters (weights and biases) are varied to maximize the predicted reward. During this phase, the critic network's parameters are held constant as it is only used to predict the reward for each set of input variables.

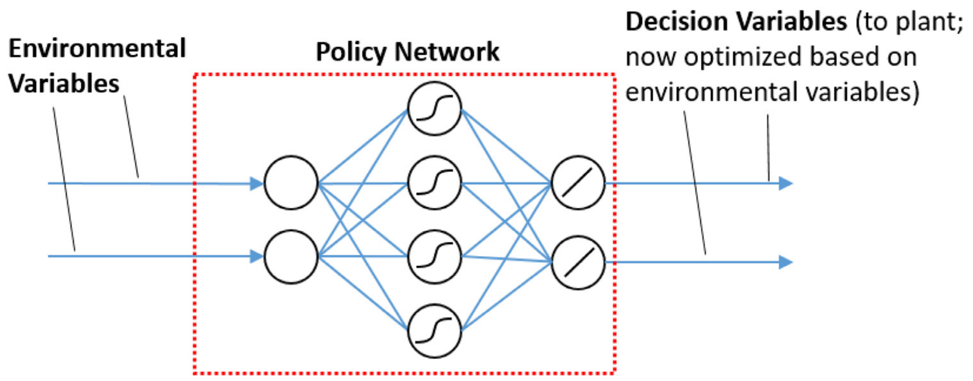

Environmental Variables
Policy Network
Decision Variables (to plant; now optimized based on environmental variables)

Fig. 6. After its training, the actor network can be extracted and put into live operation in the plant. It has "learned" to determine the optimal decision variables in a wide variety of contexts (i.e., with many combinations of measured disturbance variables).

3. Results: A process systems case study

In order to demonstrate the results of the RL-RTO framework outlined above, this methodology is applied to a process system as a demonstration and case study. The process system is the chemical reactor shown in Fig. 7. This plant consists of a continuous stirred-tank reactor, an in-reactor heater, and a control system. The reaction is the irreversible, 2nd order reaction:

$$A + B \rightarrow C \tag{3}$$

The reactor is modeled as a variable volume, non-isothermal, well-mixed reactor. Eqs. 4–8 describe the balance equations used in the simulation. All variables and their descriptions are provided in the Appendix. The reactor is equipped with flow controls to regulate the volumetric flow of components A and B (Q_A and Q_B). Q_B is used to control the level of the reactor in a cascade control arrangement, leaving Q_A as a degree of freedom for the RTO application. The heat delivered to the reactor (q) is also a degree of freedom, allowing the RTO application to use heat or not, depending on the cost.

Volume Balance

$$\frac{dV}{dt} = Q_A + Q_B - Q_{out} \tag{4}$$

Mole Balance on Component A

$$\frac{d(C_A V)}{dt} = Q_A C_{A,0} - Q_{out} C_A - R_{rxn} V \tag{5}$$

Mole Balance on Component B

$$\frac{d(C_B V)}{dt} = Q_B C_{B,0} - Q_{out} C_B - R_{rxn} V \tag{6}$$

Mole Balance on Component C

$$\frac{d(C_C V)}{dt} = -Q_{out} C_C + R_{rxn} V \tag{7}$$

Energy Balance

$$\rho c_p \frac{d(VT)}{dt} = \rho c_p Q_A T_A + \rho c_p Q_B T_B - \rho c_p Q_{out} T - R_{rxn} V \Delta H_{rxn} + q \tag{8}$$

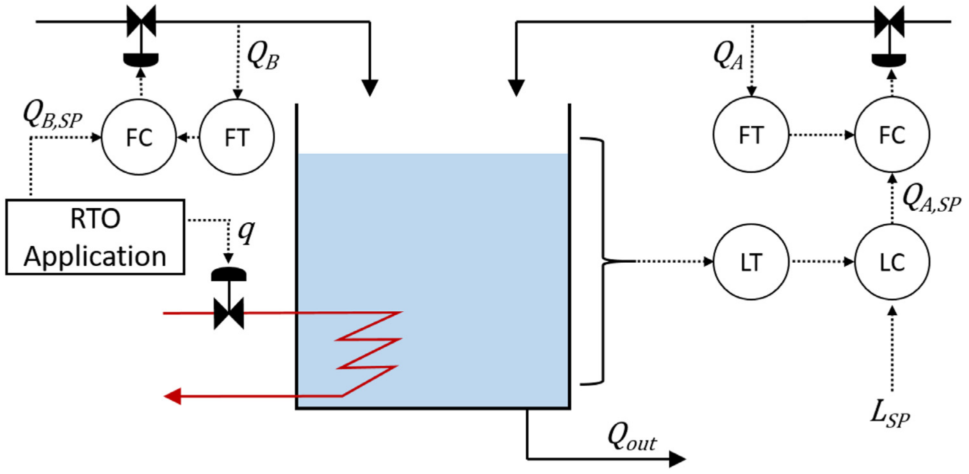


Fig. 7. A diagram of the plant.

Reaction Rate

$$R_{rxn} = k_0 e^{-E_A/RT} C_A C_B \tag{9}$$

Out Flow

$$Q_{out} = c_v \sqrt{h} \tag{10}$$

As the reactor is variable volume, the V term is not constant and is kept inside the differential on the left-hand side of each balance equation. Eq. 4 represents the volume balance. Eqs 5–7 are component balances on each component (A, B, and C, respectively). Eq. 8 is the energy balance, which assumes constant enthalpies and densities of each component and a reference temperature of 0 K for the enthalpy of each component. The reaction is modeled with a 2nd order reaction rate, following the Arrhenius equation to incorporate the exponential dependency of the reaction rate with respect to temperature (Eq. 9). The tank itself also displays nonlinear behavior with the outlet flow rate (Q_{out}) being proportional to the square root of the tank height (Eq. 10).

3.1. Optimization problem formulation

The basic objective of the optimization problem for this reactor is to maximize the real-time profit, subject to changing environmental conditions. In this case, the commodity prices for components A, B, and C change, in addition to the energy price for the reactor. The objective function is shown in Eq. 11. The decision variables in the vector x are shown in Eq. 12 with upper and lower bounds in Eq. 13. While there are seven decision variables, applying the five balance equations at steady state Eqs. 14–(18) consumes five degrees of freedom, leaving only two degrees of freedom remaining. Note that, in this case study, the reactor height has a constant set point, eliminating height/volume as a decision variable.

$$\max_{x} \quad \$_C Q_{out} C_C - \$_A Q_A - \$_B Q_B - \$_q q \tag{11}$$

s.t.

$$x = \begin{bmatrix} Q_A \\ Q_B \\ C_A \\ C_B \\ C_C \\ T \\ q \end{bmatrix} \tag{12}$$

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 300 \\ 0 \end{bmatrix} \le \begin{bmatrix} Q_A \\ Q_B \\ C_A \\ C_B \\ C_C \\ T \\ q \end{bmatrix} \le \begin{bmatrix} 8 \\ 8 \\ 20 \\ 20 \\ 10 \\ 1000 \\ 12 \end{bmatrix} \tag{13}$$

$$0 = Q_A + Q_B - Q_{out} \tag{14}$$

$$0 = Q_A C_{A,0} - Q_{out} C_A - R_{rxn} V \tag{15}$$

$$0 = Q_B C_{B,0} - Q_{out} C_B - R_{rxn} V \tag{16}$$

$$0 = -Q_{out} C_C + R_{rxn} V \tag{17}$$

$$0 = \rho c_p Q_A T_A + \rho c_p Q_B T_B - \rho c_p Q_{out} T - R_{rxn} V \Delta H_{rxn} + q \tag{18}$$

The optimization problem shown in Eqs. 11–18 is solved using two different methodologies: 1- the reinforcement learning methodology developed previously and 2- by using the steady-state first-principles model directly and formulating the optimization problem formulating it as a nonlinear program (NLP) using a conventional NLP solver. The results of this steady-state optimization problem are then applied to the dynamic system described in Eqs. 4–10. When environmental variables change, the problem is re-solved and the new, updated, optimal inputs are implemented, qualifying this as a real-time optimization (RTO) application. It should be noted that, while there is a difference in the relative magnitudes of the input and output variables, no variable scaling was employed in any of the optimization problems outlined in this work. The authors observed no issues with this, although it could be topic of future consideration.

3.2. Reinforcement learning model training

3.2.1. Data collection

To develop the RL-RTO application, the RL agent must first be trained. This is done by running the dynamic model of the plant over a wide range of different environmental conditions. The commodity prices ($\$_A$, $\$_B$, $\$_C$, and $\$_q$) are used as measured disturbances, meaning that they are known inputs fed to the RL agent. Other environmental conditions (inlet temperatures and inlet concentrations) are treated as unmeasured disturbances and are not fed in as inputs. To collect data, the dynamic simulation is run for a year of operation, with the commodity prices changing every 4 hours with a uniform distribution and the unmeasured disturbances changing every 5 minutes with a Gaussian distribution (ranges and magnitudes available in the Appendix). The simulation is conducted in MATLAB/Simulink using a variable step ordinary differential equation solver (ODE23s).

3.2.2. Training the value network

Once the data is collected, the value network fitting is conducted using the Levenberg-Marquardt algorithm to minimize the mean-squared error (MSE) between the value network model and the data representing measured profit from the plant simulation. A 20-hidden-node, three-layer perceptron is used to represent the value network function. This network uses hyperbolic tangent activation functions in the hidden layer nodes and linear activation functions in the output nodes. The data is randomly divided as follows: 70% for training, 15% for testing, and 15% for validation. The Levenberg-Marquardt training algorithm is used to minimize the mean squared error between the target data and the network's output data. Randomly initialized weights are used, but, once a viable value network has been identified, it is recommended that this network be used for future updates once new data is acquired.

The results of the value network training and cross-validation are plotted in Fig. 8. As the figure demonstrates, the model is highly accurate in comparison with the training, testing, and validation data. This means that, within the bounds of the data set used for these three functions, the model is an excellent predictor of real profit when given environmental conditions (commodity prices in this case) and the decision variables (Q_B and q). However, it is important to note that, when put in closed-loop, the RL agent will continuously recommend new operating conditions. When a condition is encountered that is not well-represented in the model, the RL-RTO application's performance at that particular operating condition may be subpar. This is due to plant/model mismatch. While the RL-RTO may identify an optimum based on the process model, this optimum may be based on an inaccurate model. For this reason, it is imperative to implement online learning for the agent, wherein both the value and policy networks are periodically updated to include the most recent observations from the actual plant. While this is not done in this simulation-based work, it would be critical for a real system, which can experience drift over time.

3.2.3. Training the policy network

The training of the value network is a relatively standard neural network fitting problem. Thus, conventional, gradient-based training algorithms, such as Levenberg-Marquardt can be used in this supervised learning problem. By contrast, the policy network's objective is one of value maximization. Conventional supervised learning solvers for regression problems do not work well with this particular problem. It is thus solved in this work using particle swarm optimization (PSO). The basic formulation of this training problem is as follows:

$$\max_{\theta_\pi} \quad \sum_{i=1}^{n} [\upsilon(S_i, A_i, \theta_\upsilon) + \lambda(A_i)] \tag{19}$$

$$\text{s.t.} \quad A_i = \pi(S_i, \theta_\pi) \tag{20}$$

$$\lambda = \begin{cases} -1000 \; if \quad A_i < LB_A \\ -1000 \; if \quad A_i > UB_A \\ 0 \quad otherwise \end{cases} \tag{21}$$

The policy network in this study is a three-layer perceptron with 12 nodes in the hidden layer and the hyperbolic tangent activation function in the hidden layer. The output layer uses a linear activation function. Eqs. 19–21 are solved using a particle swarm optimization in MATLAB (particleswarm function) with 500 particles (see Appendix for a list of all parameters used). Randomly initialized weights are used, but, once a viable policy network has been identified, it is recommended that this network be used for future updates once new data is acquired.

As Eq. 19 shows, the reward predictions produced from the previously trained value network (υ) are built into the objective for training of the policy network (π). As Eqs. 19 and 20 demonstrate, the evaluation of the objective function is dependent on the actions produced from the policy network during training. During each iteration of the networks by the PSO solver, the actions are first produced from the policy network and then fed to the value network for the computation of the objective function. Constraints on the actions are enforced by adding penalty terms to the objective function when the actions are either below their lower bounds (LB_A) or above their upper bounds (UB_A), as denoted in Eq. 21. As neural networks tend to diverge outside of the region on which they are trained, adding this penalty term to the objective keeps the policy network from having weights that result in infeasible actions.

Because RL applications require an ample amount of data for adequate training, any available data must be utilized to maximum advantage. For this reason, the value network and the policy network both utilize the same input data set for training. Because the policy network training does not have an output or target data set, the output data is not re-used. As previously explained, however, the objective functions and methodologies for these two training problems are vastly different. In a real-world application, it is assumed that plant operational data would be at a premium and therefore advantageous to re-use the data for training. In real-time operation, however, an RL-RTO application will naturally encounter new scenarios and data that were not included in its training data set. This is replicated in this study by using all new input data when testing the application's viability in the subsection of this work entitled: Real-Time Optimization.

4. Results

4.1. RL-RTO vs. NLP-Based RTO: Sensitivity plots

As RL applications use "black box" methods, it can be difficult to understand mathematically what occurs inside of these complex functions (i.e., the problem of interpretability). However, using the function itself, output data can be collected over a wide range of inputs so that one can qualitatively evaluate and even validate the decisions that the RL agent has made. While the agent is trained (using the methodology described above) over a wide range of conditions, a snapshot is presented in Fig. 9a. This plot is generated by holding the prices of reactants A and B ($\$_A$ and $\$_B$) constant, while varying the prices of product C and heat ($\$_C$ and $\$_q$, respectively). For each combination of $\$_C$ and $\$_q$, the policy network is evaluated, producing the optimal value of heat as determined by the RL agent (denoted as q^* in the plot).

Qualitatively speaking, the plot shows that the RL agent's behavior is generally correct (i.e., it makes moves that improve profitability). As the price for heat ($\$_q$) increases, the agent recommends inputting less heat. When it decreases to very low values, the RL agent recommends maximizing the use of it as a low cost commodity. The inverse is true for product C. When the price of C ($\$_C$) is low, the reactor would be less profitable, so it cuts back on q to save on cost. However, when the price of C is high, the RL agent recommends still using heat, even when it is expensive, deeming that the profit it can make is worth the cost of the heat input.

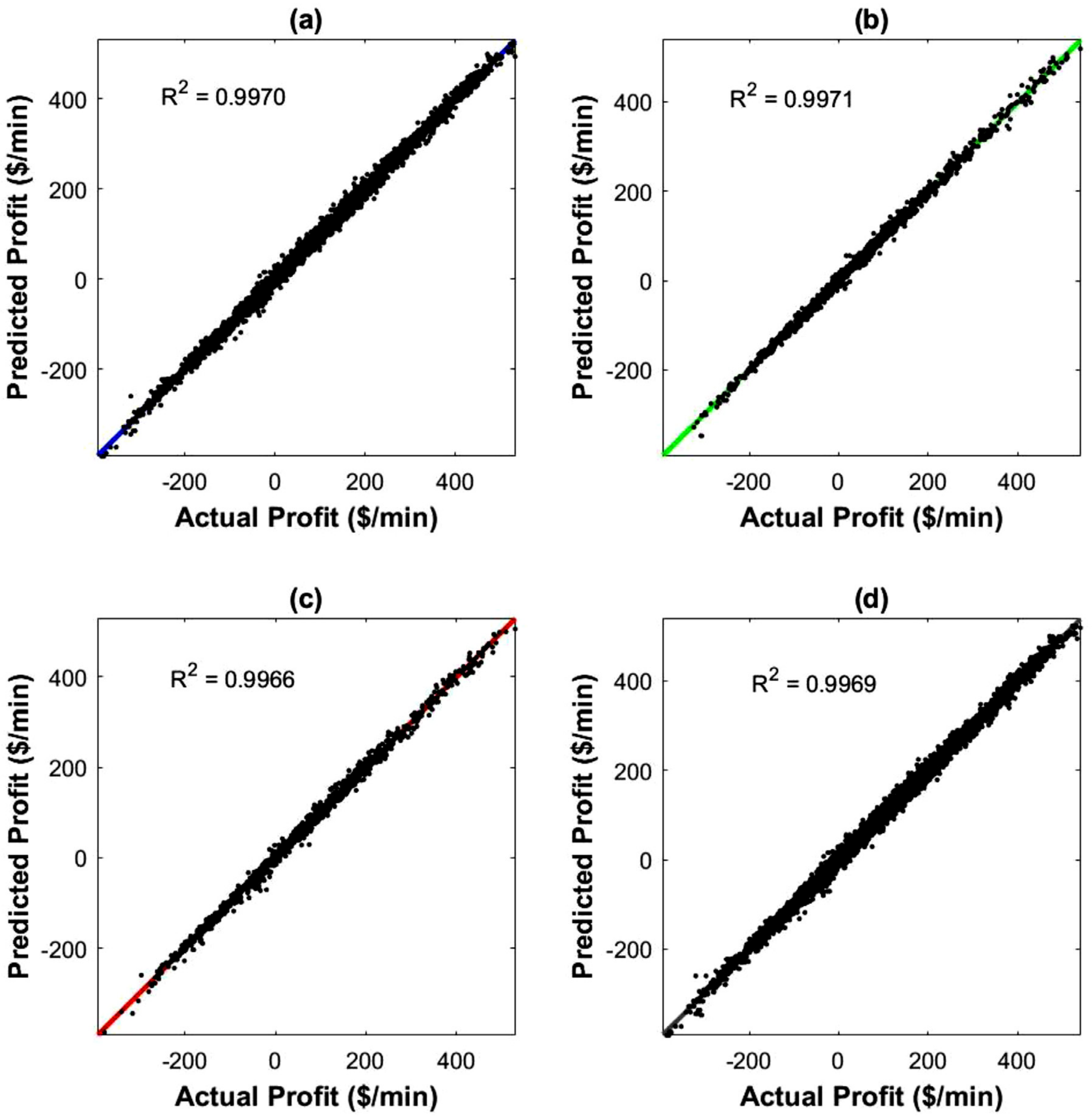


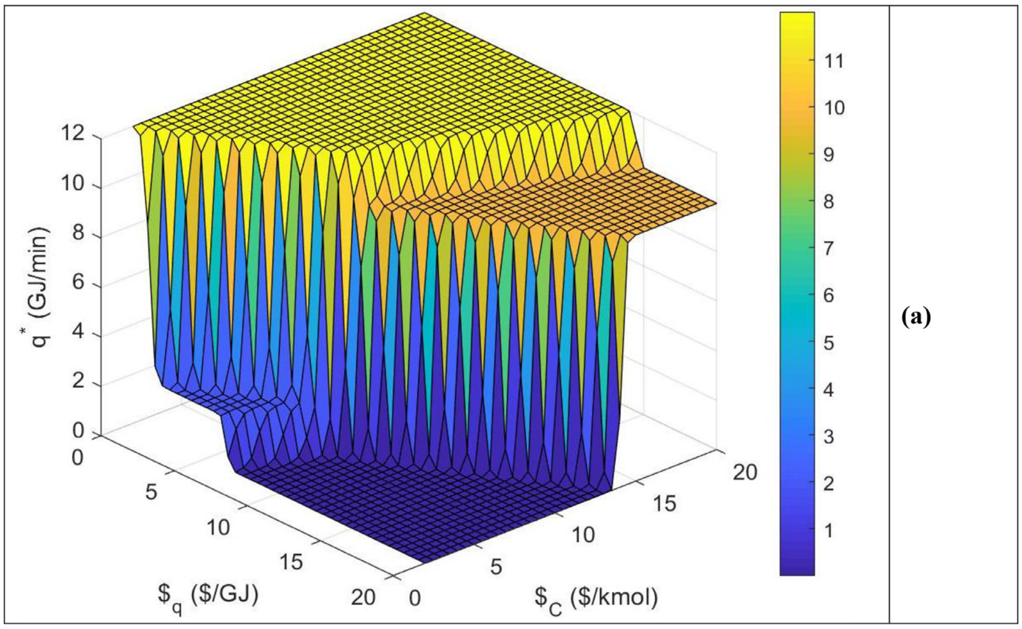
Fig. 8. Results of training for the critic function with the data sub-divided into (a) training data, (b) validation data, and (c) testing data. Subplot (d) shows all data consolidated onto a single plot.

This behavior is compared to the output from solving the NLP directly via Eqs. 11–18 (Fig. 9b). Similar trends are observed generally, with the NLP solution recommending more heat in more profitable conditions (low cost of heat and/or high selling price for product) and less heat in less profitable conditions (high cost of heat and/or low selling price for product).

The comparison of Fig. 9a (RL-RTO) to Fig. 9b (NLP-based RTO) shows that the two applications produce similar outcomes in terms of specifying how much heat to input into the reactor. This is a positive indication of the potential viability of RL-RTO, as the NLP-based RTO application uses a near-perfect steady-state model of the reactor system and its results can be relied on as being highly accurate and effective in optimization.

Another snapshot into the RL agent's behavior is given in Fig. 10a. This plot shows the optimal flow of reactant B (Q_B^*) as a function of the cost of reactant B ($\$_B$) and the selling price of product C ($\$_C$), while holding other commodity prices constant. As the figure shows, the RL agent again produces results that are qualitatively correct. As the cost of reactant B increases, the RL-RTO agent recommends using less of reactant B in order to reduce costs. Similarly, as the selling price of product C increases, the RL-RTO agent recommends using more of reactant B so that more of product C can be produced. In the bottom corner of the plot, where the selling price of product C is low and the purchase price of reactant B is high, the RL-RTO agent recommends nearly minimizing the use of reactant B, which can be qualitatively confirmed as the correct decision to make.

An interesting result shown from the RL-RTO sensitivity results is that the policy network from the RL-RTO agent tends to produce results that are tiered and mimic that of a discrete action space, despite the available action space actually being continuous. This tiered behavior is likely the result of the sigmoidal shape of the activation functions in the policy network. This results in the RL-RTO application not taking full advantage of the flexibility of the real system, resulting in sub-optimal performance. The RL-RTO agent also has results that display some abnormal behavior in the top corner of the plot, where the selling price of product C is high, and the purchase price of reactant B is low. The results tend to shoot



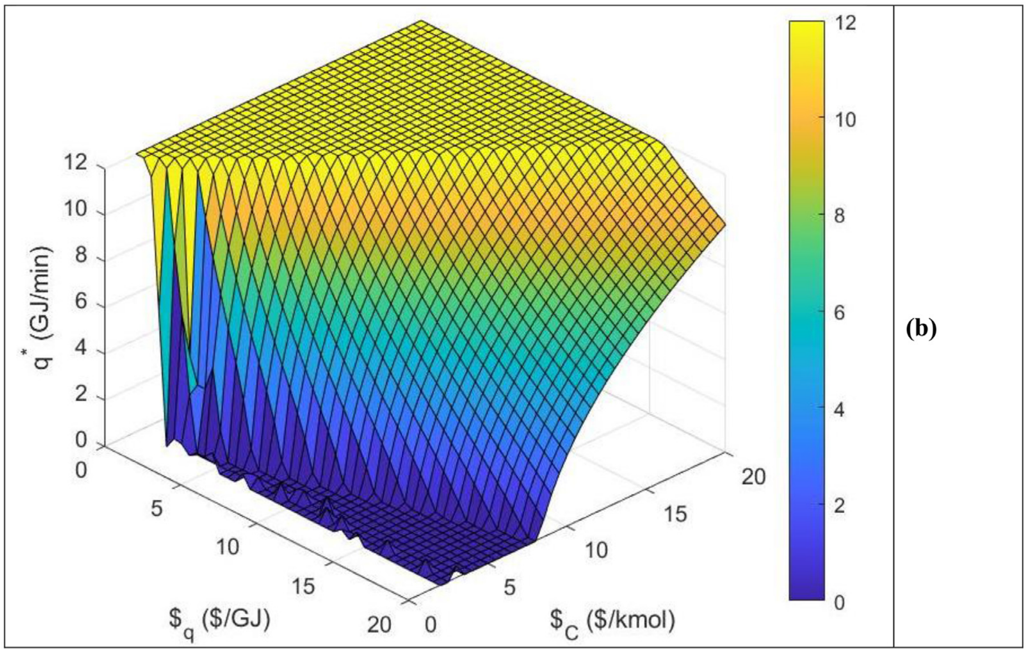


Fig. 9. Plots of the optimal value of heat (q^*) over varying price of heat ($\$_q$) and price of product C ($\$_C$) using (a) the RL-RTO method and (b) a direct NLP method. In both cases, the price of reactant A ($\$_A$) and reactant B ($\$_B$) are held constant at 11 $/kmol and 27.5 $/kmol, respectively.

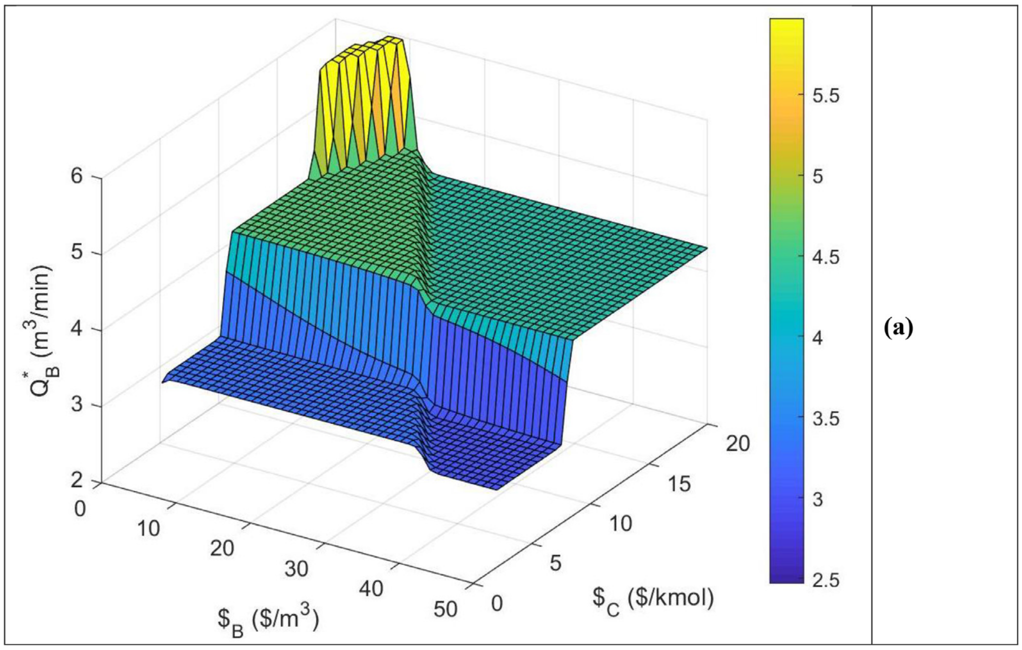
up to abnormally high values of Q_B, indicating a region where the policy network does not extrapolate well and may require more data in this region for training.

By contrast, the NL-based RTO (with results shown in Fig. 10b) produces results for Q_B^* that are much smoother. Again, since the NL-based RTO utilizes a near perfect, steady-state model of the plant, these results can be considered to be highly accurate (although it is important to note that the model is non-convex, so there are some minor areas of deviation from smooth behavior). Unlike the RL-RTO application, the NLP-based RTO takes much better advantage of the continuous action space of the true system and does not exhibit the tiered behavior of the RL-RTO application.

Overall, however, the results for RL-RTO are still promising in that overall behavioral trends are similar to an NLP-based RTO application using a near perfect model. It is clear that RL methods will require much more development before they are competitive with more conventional RTO methods. A performance comparison under a wide range of environmental/economic conditions between the two methods is given in the next section.

4.2. Real-Time optimization

The sensitivity plots from Fig. 9 and Fig. 10 give some insight into how the RL-RTO application responds under specified environmental/economic conditions. They also show how RL-RTO compares to a traditional NLP-based method. However, judging the true effectiveness of the application requires a performance evaluation over a long period of time in order to see how it handles a wide range of external conditions. To perform this evaluation, the RL-RTO application is put in closed-loop with the original dynamic



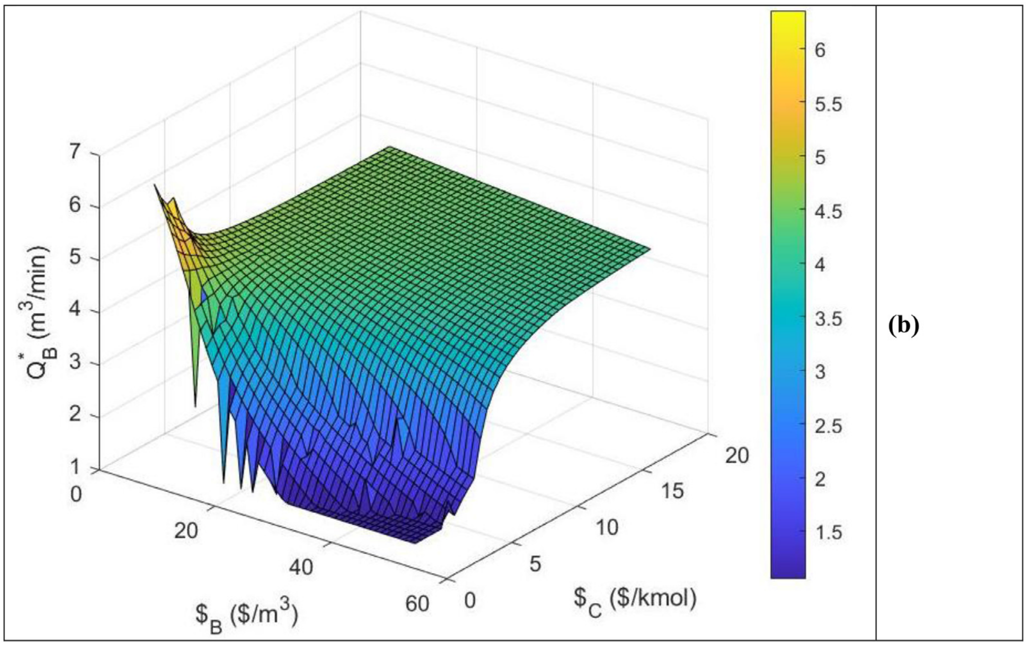


Fig. 10. (a) A plot of the optimal flow rate of reactant B (Q^*_B) over varying price of reactant B ($\$_B$) and price of product C ($\$_C$) using (a) the RL-RTO method and (b) a direct NLP method. In both cases, the price of reactant A ($\$_A$) and the price of heat ($\$_q$) are held constant at 11 \$/kmol and 27.5 \$/kmol, respectively.

model Eqs. 4–(10) acting as the "plant" and the disturbance variables changing at random as specified in the Nomenclature table (see Appendix). As the measured disturbances ($\$_A$, $\$_B$, $\$_C$, and $\$_q$) change, the RL-RTO updates the plant inputs accordingly, based on the recommended values (q, and Q_B) obtained from evaluation of the policy network with those inputs. In this closed-loop simulation, the measured disturbances change every four hours, which triggers the RL-RTO to run and input its recommended operating conditions to the plant. This simulation is run for a year in order to summarize the results and compare to the NLP-based RTO application. The simulation is also run with a completely new set of random inputs as disturbances (measured and unmeasured) than were used during the training phase.

Fig. 11 shows a sampling of 24 hours out of the year-long simulation results. As the figure shows, in the first few hours, when all commodity prices are relatively low, the RL-RTO application keeps Q_B and q steady, although the predicted profit does fluctuate. However, at hour 11 in the plot, the price of reactant B increases and the cost of product C decreases, creating an unfavorable environment for profitability of the plant. The RL-RTO application responds accordingly with a decrease in Q_B as well as in q, as the application tried to minimize the losses. At hour 15 in the plot, when the price of product C recovers and conditions become favorable again, Q_B and q ramp back up. At hour 23, the price of heat reaches near its maximum and the rate of heat input (q) decreases to 0.

Fig. 11 is only a small snapshot of the RL-RTO's behavior, provided to help readers gain at least qualitative insight into the RL agent's decision-making. These same general trends are observed throughout the entire simulated year, with the RL-RTO generally

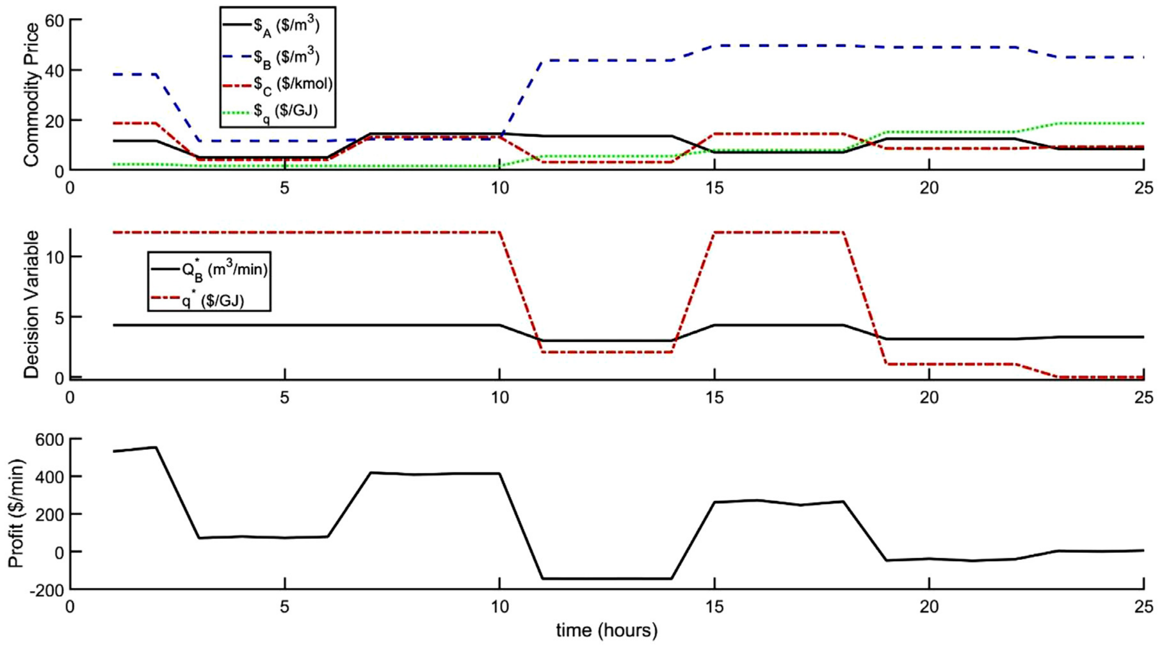


Fig. 11. The results of the RL-RTO over a sample of 24 hours. The top plot shows the commodity prices, which change at random in the simulation. The middle plot shows the resulting decisions as a result of the policy network evaluation, and the bottom plot shows the projected profit, as a result of the value network evaluation. The RL-RTO is run at a frequency of every 4 hours.

Table 1
A summary of the annual performance results of the RL-RTO application compared to other modes of operation (NLP-based RTO, maximum throughput operation, and random inputs).

	Random Inputs (training)	Maximum Throughput	NLP-based RTO	Reinforcement Learning RTO
Annual Profit	$38,410,000	$87,420,000	$102,460,000	$95,850,000
% Improvement (relative to Max Throughput)	-56.06%	N/A	17.20%	9.64%

making operational decisions that make sense to a human observer.

In order to effectively compare the two methodologies, the NLP-Based RTO is also evaluated for a period of an entire year under the same conditions. These two scenarios are compared to an additional operating scenario: maximum throughput. This operating scenario feeds reactants A and B to the reactor in equal quantities and also uses the maximum allowable amount of heat. This results in the reactor generating the maximum possible molar flow rate of product C and is a likely operating scenario in the absence of an active real-time optimization strategy.

The results of this annual study are shown in Table 1. As the table shows, the NLP-based RTO application demonstrates superior performance, with an annual profit 17.2% greater than the maximum throughput operating strategy. This is to be expected as the NLP-based RTO application uses a near-perfect model. Its optimization model is based off of an exact steady-state model of the simulated plant, only it does not account for the variability in unmeasured disturbances. Also, because the RTO strategy is a steady-state optimization approach, the optimization model does not account for the dynamics of the simulated plant. In comparison, the RL-RTO application is not as effective, resulting in only a 9.6% increase in profitability, relative to the maximum throughput case. This, however, is still a promising result as it results in a substantial economic gain for the plant. Particularly for processes where accurate first-principles modeling is not feasible due to process uncertainties or lack of the skilled manpower required to model the process, RL-RTO could be a potential option, although much more development needs to be done first.

While the scenario using random inputs would certainly not be considered a viable operating strategy for the plant, it is still instructive to compare RL-RTO to this scenario. The RL-RTO operating strategy results in more than double the profitability of this scenario. Economics aside, the fact that RL-RTO can take a naïve operating strategy (which is what it does prior to any knowledge of the system, during the training phase) and transform it into a strategy that results in dramatically better performance demonstrates its effectiveness. While the maximum throughput strategy may seem obvious to a human operator, it is not obvious to an artificial intelligence agent, which must only use learned responses between inputs and outputs before it can formulate an adequate operating strategy. The fact that RL-RTO can do this while starting from a completely uninformed beginning demonstrates its potential for RTO applications, again, while still needing much more development.

One area where RL-RTO is far superior to the NLP-based RTO application is in computation time. A comparison of the computation time and average number of function evaluations is contained in Table 2. The NLP-based method requires evaluating the objective and constraint functions many times for each implementation of an optimal solution (every four hours). This is required as the solver obtains information about the system's gradients and iterates to find the point of maximum profitability. In the year-long study, the average number of function evaluations per RTO solution is 664.3. This results in a simulation time (including the dynamic simulated plant model) of 243.3 s.

By comparison, the RL-RTO application only requires a single function pass through the policy network to obtain what it deems to be the optimal operating strategy, resulting in a complete simulation time of only 31.2 s. This is because RL functions much differently than a conventional optimization method. In a conventional method, the solver must iterate repeatedly until it satisfies

Table 2
A comparison of computation times when running the year-long optimization and simulation study. Simulation times include running the dynamic simulated plant model.

	NLP-based RTO	Reinforcement Learning RTO
Total computation time for simulation	243.3 s	31.2 s
Average number of function evaluations per solution	664.3	1

the conditions of optimality. In the RL-RTO methodology presented in this paper, the "optimal" behavior is instead built into the policy network itself. After training, the policy network only requires data about the environmental conditions of the plant and returns the recommended decision variables. Unlike a supervised learning application, which are typically used to predict based on a regression, the neural network representing the policy function is used to specify the inputs directly. It must be noted that a substantial amount of computation does go into training the RL-RTO agent. In this demonstration, the total training time is 604.1 seconds. This can occur offline, however, and could be carried out in a way that is not detrimental to the optimization of the actual process. For instance, the training could be done on a parallel server and would not need to be done in real time. The trained network could then be shared with the online optimization server for real-time computation.

5. Conclusions and future work

This work outlines a novel methodology for doing real-time optimization (RTO) for process systems using reinforcement learning (RL). This application, denoted as RL-RTO in this work, provides readers with a step-by-step procedure for developing an RL application to optimize a process system. While there remains much work to be done before applications such as this are viable for RTO of real process systems, this work has some key takeaways that may benefit future research and development efforts.

Among the key contributions of this work is the development of a hybrid training methodology. This training methodology uses a gradient-based solver to train the value network (a.k.a., the critic) and a particle swarm optimization (PSO) algorithm to train the policy network (a.k.a., the actor). The necessity of using a derivative free optimization method like PSO to train the policy network is due to the fact that the objective function for this optimization problem is fundamentally different than that in more conventional regression neural network problems, where the objective is to minimize the sum of squared error or a similar metric. In the training of the policy network, the objective is to maximize the expected profit. This is achieved by changing the weights and biases of the policy network such that the network produces the proper values of the decision variables in response to measurements of the environmental conditions.

This hybrid learning method includes a batch fitting and optimization technique for the value network, where all the training data is processed simultaneously. This is in contrast with many other RL algorithms, which only make marginal updates with a single optimization step after each new data point is collected. The batch methodology presented here, coupled with the PSO solver, allows the RL-RTO algorithm to consider all the data simultaneously and approach a global best solution, rather than getting stuck in the nearest local minimum. This allows the optimizer to determine the optimal weights while considering the full range of data. This methodology could also be readily converted to an online method, where with each new data set, the model could be updated, but while still using as much historical data as necessary. As the policy network presumably improves with more experiential data, the value network could also be updated accordingly. This batch-online training methodology would allow a facility to shortcut much of the requisite exploration required in RL applications. Rather, it could leverage past data to provide the initial fit for a model, then begin doing more proactive exploration from that point onward.

Another novel feature of this work is the ability to maintain constraints using RL. The methodology presented allows for upper and lower bounds on constraints to be enforced by using a penalty method, which adds a numerical penalty to the objective function when training the policy network to prevent it from prescribing process inputs that would fall outside of these bounds. This feature would be critical in applying RL-RTO to a real process system as it would keep the application from going to unsafe or infeasible operating conditions.

To evaluate the performance of RL-RTO versus more standard methods, it is applied to a simulated process. The RL-RTO demonstrates promising performance, with profit increased by 9.64% compared to the max throughput operating methodology over the course of a one-year simulation, which covers a wide range of different environmental/economic conditions. While this result is promising, there is still much work to be done. For instance, a more conventional RTO method using a first principles model and nonlinear programming produces an increase in profit of 17.20%. However, this method is under idealized circumstances and uses a near-perfect model, which would be difficult to replicate with a real process.

RL-RTO represents a model-free methodology to optimizing process systems. Unlike the NLP-based method, RL-RTO would not require subject matter expertise and could be almost purely data driven. It could be applied to complex processes, where the physics are not readily understood or to processes in facilities where there are insufficient resources to hire subject matter experts capable of developing the complex and accurate models typically required for the NLP-based method.

Future work must be completed to make RL-RTO a viable methodology for real-time optimization. One of the critical first steps is to explore different neural network model structures and different types of activation functions in the policy network. As Fig. 9a and Fig. 10a demonstrate, the activation functions used in this study (sigmoidal), result in some pseudo-discreteness in their output. These plots show that optimal solutions are given in tiers, whereas the closer-to-true optimal solutions result in much smoother transitions (see Fig. 9b and Fig. 10b).

Another critical area of development is in improving the robustness of finding a policy network that results in near-optimal behavior. Because the network parameters are randomly initialized and fitting the neural network itself is a highly non-convex optimization problem, attempts to find a suitable policy network frequently fail and may require multiple iterations before an effective policy network is determined.

Another challenge is that the purely data-driven approach of this algorithm will naturally leave some gaps where the system has not yet explored a particular operating regime and may result in poor operation. While learning from this data is certainly a function of RL, future work should certainly include methods to make the applications more robust and avoid dangerous or grossly inefficient operating conditions. Hybrid methods, which leverage physics-based models may also be considered.

While RL-RTO presents a fundamentally different method for optimization, where the optimal behavior under different stimuli is built into the policy network itself, it is certainly not the only machine-learning based optimization method available. For instance, one could use a machine-learning-based regression model of a plant as the process model and then iterate on that model with an optimization solver at each time step of operation to determine optimality for the plant. Thus, RL-RTO must be compared to these different types of ML-based optimization methods to determine which might be superior and how RL-RTO can be improved in order to be more competitive. Regardless, the RL-RTO application presented here has presented both a novel use of RL and a novel methodology for doing RTO for a process system. This topic appears promising and worthy of further investigation.

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

By Kody M. Powell: Conceptualization, Methodology, Software, Investigation, Writing - original draft, Supervision. **Derek Machalek:** Conceptualization, Methodology, Investigation, Writing - review & editing, Visualization. **Titus Quah:** Methodology, Investigation, Resources, Writing - review & editing, Validation.

Acknowledgements

The authors would like to thank the University of Utah for funding this research.

Appendix

Particle swarm optimization parameters

Function Tolerance	1.000×10^{-6}
Inertia Range	0.100 – 1.100
Initial Swarm Span	2000
Minimum Neighbors Fraction	0.25
Self Adjustment Weight	1.490
Social Adjustment Weight	1.490
Swarm Size	500

References

Blackburn, L., Young, A., Rogers, W.P., Hedengren, J.D., Powell, K.M., 2019. Dynamic optimization of a district energy system with storage using a novel mixed-integer quadratic programming algorithm. Optim. Eng. 20 (2), 575–603.

Blackburn, L.D., Tuttle, J.F., Powell, K.M., 2020. Real-time optimization of multi-cell industrial evaporative cooling towers using machine learning and particle swarm optimization. J. Clean. Prod. 271, 122175. doi:10.1016/j.jclepro.2020.122175.

Bora, T.C., Mariani, V.C., Coelho, L.dos S., Jan. 2019. Multi-objective optimization of the environmental-economic dispatch with reinforcement learning based on non-dominated sorting genetic algorithm. Appl. Therm. Eng. 146, 688–700. doi:10.1016/J.APPLTHERMALENG.2018.10.020.

Cheng, Y., Huang, Y., Pang, B., Zhang, W., Sep. 2018. ThermalNet: a deep reinforcement learning-based combustion optimization system for coal-fired boiler. Eng. Appl. Artif. Intell. 74, 303–311. doi:10.1016/J.ENGAPPAI.2018.07.003.

Diehl, M., Bock, H.G., Schlöder, J.P., Findeisen, R., Nagy, Z., Allgöwer, F., Jun. 2002. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. J. Process Control 12 (4), 577–585. doi:10.1016/S0959-1524(01)00023-3.

Görges, D., Jul. 2017. Relations between model predictive control and reinforcement learning. IFAC-PapersOnLine 50 (1), 4920–4928. doi:10.1016/J.IFACOL.2017.08.747.

Jia, R., Jin, M., Sun, K., Hong, T., Spanos, C., Feb. 2019. Advanced building control via deep reinforcement learning. Energy Procedia 158, 6158–6163. doi:10.1016/J.EGYPRO.2019.01.494.

Li, F., Jiang, Q., Zhang, S., Wei, M., Song, R., Jun. 2019. Robot skill acquisition in assembly process using deep reinforcement learning. Neurocomputing 345, 92–102. doi:10.1016/J.NEUCOM.2019.01.087.

Lillicrap, T.P., et al., 2016. Continuous control with deep reinforcement learning. International Conference on Learning Representations OnlineAvailable http://arxiv.org/abs/1509.02971 .

Ma, Y., Zhu, W., Benton, M.G., Romagnoli, J., Mar. 2019. Continuous control of a polymerization system with deep reinforcement learning. J. Process Control 75, 40–47. doi:10.1016/J.JPROCONT.2018.11.004.

Machalek, D., Quah, T., Powell, K.M., 2020. Dynamic economic optimization of a continuously stirred tank reactor using reinforcement learning. In: 2020 American Control Conference (ACC). Denver, CO, USA, pp. 2955–2960. doi:10.23919/ACC45564.2020.9147706.

Mnih, V., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533. doi:10.1038/nature14236.

Morinelly, J.E., Ydstie, B.E., Jan. 2016. Dual MPC with reinforcement learning. IFAC-PapersOnLine 49 (7), 266–271. doi:10.1016/J.IFACOL.2016.07.276.

Pandian, B.J., Noel, M.M., Sep. 2018. Control of a bioreactor using a new partially supervised reinforcement learning algorithm. J. Process Control 69, 16–29. doi:10.1016/J.JPROCONT.2018.07.013.

Pedersen, C.C., Hoffman, T.W., Jun. 1997. The road to advanced process control: from DDC to real-time optimization and beyond. IFAC Proc. Vol. 30 (9), 251–279. doi:10.1016/S1474-6670(17)43168-5.

Powell, K.M., Kim, J.S., Cole, W.J., Kapoor, K., Mojica, J.L., Hedengren, J.D., Edgar, T.F., 2016. Thermal energy storage to minimize cost and improve efficiency of a polygeneration district energy system in a real-time electricity market. Energy 113, 52–63.

Qi, X., Luo, Y., Wu, G., Boriboonsomsin, K., Barth, M., Feb. 2019. Deep reinforcement learning enabled self-learning control for energy efficient driving. Transp. Res. Part C Emerg. Technol. 99, 67–81. doi:10.1016/J.TRC.2018.12.018.

Qin, S.J., Chiang, L.H., Jul. 2019. Advances and opportunities in machine learning for process data analytics. Comput. Chem. Eng. 126, 465–473. doi:10.1016/J.COMPCHEMENG.2019.04.003.

Rashid, K., Sheha, M.N., Powell, K.M., Jun. 2018. Real-time optimization of a solar-natural gas hybrid power plant to enhance solar power utilization. In: 2018 Annual American Control Conference (ACC), pp. 3002–3007. doi:10.23919/ACC.2018.8431220.

Rashid, K., Safdarnejad, S.M., Powell, K.M., May 2019. Process intensification of solar thermal power using hybridization, flexible heat integration, and real-time optimization. Chem. Eng. Process. - Process Intensif. 139, 155–171. doi:10.1016/j.cep.2019.04.004.

Rocchetta, R., Bellani, L., Compare, M., Zio, E., Patelli, E., May 2019. A reinforcement learning framework for optimal operation and maintenance of power grids. Appl. Energy 241, 291–301. doi:10.1016/J.APENERGY.2019.03.027.

Safdarnejad, S.M., Tuttle, J.F., Powell, K.M., May 2019. Dynamic modeling and optimization of a coal-fired utility boiler to forecast and minimize NOx and CO emissions simultaneously. Comput. Chem. Eng. 62–79. doi:10.1016/j.compchemeng.2019.02.001.

Safdarnejad, S.M., Tuttle, J.F., Powell, K.M., May 2019. Development of a roadmap for dynamic process intensification by using a dynamic, data-driven optimization approach. Chem. Eng. Process. - Process Intensif. doi:10.1016/j.cep.2019.04.002.

Shah, H., Gopal, M., Jan. 2016. Model-free predictive control of nonlinear processes based on reinforcement learning. IFAC-PapersOnLine 49 (1), 89–94. doi:10.1016/J.IFACOL.2016.03.034.

Shin, J., Badgwell, T.A., Liu, K.-H., Lee, J.H., Aug. 2019. Reinforcement learning – overview of recent progress and implications for process control. Comput. Chem. Eng. 127, 282–294. doi:10.1016/J.COMPCHEMENG.2019.05.029.

Syafiie, S., Tadeo, F., Martinez, E., Alvarez, T., Jan. 2011. Model-free control based on reinforcement learning for a wastewater treatment problem. Appl. Soft Comput. 11 (1), 73–82. doi:10.1016/J.ASOC.2009.10.018.

Syafiie, S., Tadeo, F., Martinez, E., Sep. 2007. Model-free learning control of neutralization processes using reinforcement learning. Eng. Appl. Artif. Intell. 20 (6), 767–782. doi:10.1016/J.ENGAPPAI.2006.10.009.

Szepesvári, C., 2010. Algorithms for Reinforcement Learning. Morgan & Claypool.

Tsurumine, Y., Cui, Y., Uchibe, E., Matsubara, T., Feb. 2019. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. Rob. Auton. Syst. 112, 72–83. doi:10.1016/J.ROBOT.2018.11.004.

Tuttle, J.F., Powell, K.M., May 2019. Analysis of a thermal generator's participation in the western energy imbalance market and the resulting effects on overall performance and emissions. Electr. J. doi:10.1016/j.tej.2019.05.010.

Tuttle, J.F., Blackburn, L.D., Powell, K.M., 2020. On-line classification of coal combustion quality using nonlinear SVM for improved neural network NOx emission rate prediction. Comput. Chem. Eng. 141, 106990. doi:10.1016/j.compchemeng.2020.106990.

Tuttle, J.F., Vesel, R., Alagarsamy, S., Blackburn, L.D., Powell, K., Dec. 2019. Sustainable NOx emission reduction at a coal-fired power station through the use of online neural network modeling and particle swarm optimization. Control Eng. Pract. 93, 104167. doi:10.1016/J.CONENGPRAC.2019.104167.

Vázquez-Canteli, J.R., Nagy, Z., Feb. 2019. Reinforcement learning for demand response: a review of algorithms and modeling techniques. Appl. Energy 235, 1072–1089. doi:10.1016/J.APENERGY.2018.11.002.

Valladares, W., et al., May 2019. Energy optimization associated with thermal comfort and indoor air control via a deep reinforcement learning algorithm. Build. Environ. 155, 105–117. doi:10.1016/J.BUILDENV.2019.03.038.

Wang, Y., Velswamy, K., Huang, B., Jan. 2018. A novel approach to feedback control with deep reinforcement learning. IFAC-PapersOnLine 51 (18), 31–36. doi:10.1016/J.IFACOL.2018.09.241.

Wang, B., Zhou, M., Xin, B., Zhao, X., Watada, J., Jul. 2019. Analysis of operation cost and wind curtailment using multi-objective unit commitment with battery energy storage. Energy 178, 101–114. doi:10.1016/J.ENERGY.2019.04.108.

Yang, L., Nagy, Z., Goffin, P., Schlueter, A., Oct. 2015. Reinforcement learning for optimal control of low exergy buildings. Appl. Energy 156, 577–586. doi:10.1016/J.APENERGY.2015.07.050.

You, C., Lu, J., Filev, D., Tsiotras, P., Apr. 2019. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Rob. Auton. Syst. 114, 1–18. doi:10.1016/J.ROBOT.2019.01.003.

Zhou, M., Wang, B., Li, T., Watada, J., Dec. 2018. A data-driven approach for multi-objective unit commitment under hybrid uncertainties. Energy 164, 722–733. doi:10.1016/J.ENERGY.2018.09.008.