Comparing Reinforcement Learning Methods for Real-Time Optimization of a Chemical Process

Authors:

Titus Quah, Derek Machalek, Kody M. Powell

Date Submitted: 2021-06-02

Keywords: reinforcement learning, process optimization, real-time optimization, Proximal Policy Optimization, artificial neural networks, Particle Swarm Optimization

Abstract:

One popular method for optimizing systems, referred to as ANN-PSO, uses an artificial neural network (ANN) to approximate the system and an optimization method like particle swarm optimization (PSO) to select inputs. However, with reinforcement learning developments, it is important to compare ANN-PSO to newer algorithms, like Proximal Policy Optimization (PPO). To investigate ANN-PSO's and PPO's performance and applicability, we compare their methodologies, apply them on steady-state economic optimization of a chemical process, and compare their results to a conventional first principles modeling with nonlinear programming (FP-NLP). Our results show that ANN-PSO and PPO achieve profits nearly as high as FP-NLP, but PPO achieves slightly higher profits compared to ANN-PSO. We also find PPO has the fastest computational times, 10 and 10,000 times faster than FP-NLP and ANN-PSO, respectively. However, PPO requires more training data than ANN-PSO to converge to an optimal policy. This case study suggests PPO has better performance as it achieves higher profits and faster online computational times. ANN-PSO shows better applicability with its capability to train on historical operational data and higher training efficiency.

Record Type: Published Article

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

Citation (overall record, always the latest version):	LAPSE:2021.0497
Citation (this specific file, latest version):	LAPSE:2021.0497-1
Citation (this specific file, this version):	LAPSE:2021.0497-1v1

DOI of Published Version: https://doi.org/10.3390/pr8111497

processes

MDPI

Article

Comparing Reinforcement Learning Methods for Real-Time Optimization of a Chemical Process

Titus Quah [1], Derek Machalek [1] and Kody M. Powell [1,2,*]

[1] Department of Chemical Engineering, University of Utah, 50 Central Campus Dr, Salt Lake City, UT 84112, USA; titus.quah@gmail.com (T.Q.); derek.machalek@utah.edu (D.M.)
[2] Department of Mechanical Engineering, University of Utah, 1495 E 100 S, Salt Lake City, UT 84112, USA
* Correspondence: kody.powell@utah.edu; Tel.: +1-801-581-3957

Received: 31 October 2020; Accepted: 17 November 2020; Published: 19 November 2020

Abstract: One popular method for optimizing systems, referred to as ANN-PSO, uses an artificial neural network (ANN) to approximate the system and an optimization method like particle swarm optimization (PSO) to select inputs. However, with reinforcement learning developments, it is important to compare ANN-PSO to newer algorithms, like Proximal Policy Optimization (PPO). To investigate ANN-PSO's and PPO's performance and applicability, we compare their methodologies, apply them on steady-state economic optimization of a chemical process, and compare their results to a conventional first principles modeling with nonlinear programming (FP-NLP). Our results show that ANN-PSO and PPO achieve profits nearly as high as FP-NLP, but PPO achieves slightly higher profits compared to ANN-PSO. We also find PPO has the fastest computational times, 10 and 10,000 times faster than FP-NLP and ANN-PSO, respectively. However, PPO requires more training data than ANN-PSO to converge to an optimal policy. This case study suggests PPO has better performance as it achieves higher profits and faster online computational times. ANN-PSO shows better applicability with its capability to train on historical operational data and higher training efficiency.

Keywords: particle swarm optimization; artificial neural networks; real-time optimization; Proximal Policy Optimization; process optimization; reinforcement learning

1. Introduction

1.1. Motivation

Many chemical processes, like coal combustion and bioreactors, are complex, and thus deriving appropriate models and optimizing outputs is difficult [1–3]. Machine learning has shown success in optimizing complex systems such as scheduling electricity prices to manage demand and maximize power grid performance [4–6]. This motivates exploration of other machine learning techniques like reinforcement learning (RL) on model-free optimization [7].

RL research has seen many breakthroughs in recent years with new algorithms capable of defeating most humans in difficult games [8–11]. These algorithms are not specifically designed to play games, but to learn and accomplish general tasks. A real-world example can be seen in OpenAI's algorithm which learned how to control a robotic hand to solve a Rubick's cube under disturbances [12].

1.2. Literature Review

One popular RL algorithm in process systems, ANN-PSO, trains an artificial neural network (ANN) to model the system and uses a global optimization method, such as particle swarm optimization (PSO), to select optimal inputs. ANN-PSO is an off-policy algorithm, meaning it can train using data previously gathered under normal operating conditions. In a real case study, ANN-PSO is shown to be effective in

minimizing NOx production and high-temperature alarms of a power plant using ANN-PSO to control 68 dampers in a closed-loop operation [2]. This demonstrates that ANN-PSO is effective in learning and controlling high-dimensional systems. There have been other power plant successes with yield maximization, efficiency maximization, and multiobjective optimization [13–16]. Bhattacharya et al. demonstrates that in a real validation test, ANN-PSO increases ϵ-polylysine production in marine bacteria by predicting optimal medium composition [17]. Khajeh et al. shows ANN-PSO achieves high extraction molybdenum extraction from water using silver nanoparticles by determining optimal volumes of the extractant and complex, pH, and adsorption time [18]. There have been other successes for optimization in pharmaceuticals, extraction, and HVAC [19–26]. While this shows ANN-PSO is widely used to optimize process inputs, newer algorithms may offer further improvements.

Actor–critic algorithms are a newer class of RL algorithms which train an actor ANN to select optimal actions given a set of observations and a critic ANN to estimate the value of the set of observations. Proximal policy optimization (PPO) is an on-policy stochastic actor–critic algorithm, on-policy meaning the algorithm must gather the training data, and stochastic meaning the actions taken are sampled from a distribution. Over a 90-day real case study, Filipe et al. demonstrated the efficacy of PPO on minimizing power usage and high level alarms of a wastewater treatment plant [27]. For combined heat and power systems economic dispatch, Zhou et al. found PPO performs as well as conventional methods, but handles more scenarios without recalculation, has better time flexibility, and requires less input information [28]. For an integrated electrical and heating system with wind energy, Zhang demonstrates PPO can optimize under power, demand, and spot electricity price uncertainties to minimize operational costs [29]. Actor–critic algorithms have also been used in optimizing costs for power systems, lot scheduling, hybrid vehicles, autonomous vehicles, brine injection, and drones [30–36].

1.3. Contributions

While process systems RL applications is an active area of research, there are few papers comparing the algorithms in their applicability and performance in process systems. To investigate if the widely used ANN-PSO can be replaced by newer actor–critic methods, this paper presents a novel comparison between two algorithms—ANN-PSO and PPO—by comparing the methods of the algorithms and evaluating these algorithms on a case study of a stochastic steady-state chemical optimization problem. To explore both algorithms' potential to achieve an optimal policy, ANN-PSO and PPO are also compared to two benchmark control strategies: nonlinear programming on a first principles model and maximum production. The algorithms' performance and applicability are compared based on implementation feasibility, system optimization, training efficiency, and online computational time. Furthermore, insight into the behavior of these algorithms is explored through parity plots of the agents' predicted and actual profits, and sensitivity analysis of the agents' actions. Below is a summary of the novel contributions of this work.

1. This work represents a first-of-its-kind comparison study between PPO and other methods for real-time optimization. Specifically, comparisons are made to maximum production operation (no optimization), optimization using an ML model (artificial neural network) and particle swarm optimization (ANN-PSO), and optimization using a first principles model and gradient-based non-linear programming.
2. Our results demonstrate that PPO increases profitability by 16% compared to no optimization. It also outperforms ANN-PSO by 0.6% and comes remarkably close to matching the performance of the FP-NLP method, getting within 99.9% of FP-NLP profits.
3. Though more time must be invested into training the system, PPO reduces online computational times, resulting in 10 and 10,000 times faster computation compared to FP-NLP and ANN-PSO, respectively .
4. ANN-PSO has higher training efficiency compared to PPO as ANN-PSO converges with $\approx 10^5$ training examples while PPO converges to an optimal policy with $\approx 10^6$ training examples.

5. Parity plots suggest ANN-PSO's lower profits are due to PSO exploiting errors in the ANN, causing ANN-PSO to consistently overpredict profit and select suboptimal actions.
6. Comparing PPO and ANN-PSO, PPO has better performance as shown by its higher profits and faster computational times. ANN-PSO has better applicability as shown by its higher training efficiency and capability to train on historical operational data.

The following are the sections and the material they cover. Section 2 introduces the optimization methods and their theory. Section 3 presents the case study of optimizing steady state operation of a continuously stirred tank reactor. Section 4 describes how the optimization methods are implemented and evaluated. Section 5 presents and analyzes the results from the case study. Section 6 summarizes our findings and suggests future work.

2. Materials and Methods

2.1. Markov Decision Process

An optimization problem is considered where an agent interacts with an environment which is assumed to be fully observable. This problem can be formulated as a Markov Decision Process (MDP) where the environment is described by a set of possible states $S \in \mathbb{R}^n$, possible actions $A \in \mathbb{R}^m$, a distribution of initial states $p(s_0)$, a reward distribution function $R(s_t, a_t)$ given state s_t and action a_t, a transitional probability $p(s_{t+1}|s_t, a_t)$, and a future reward discount factor γ. An episode begins with the environment sampling an initial state $s_0 \in S$ with probability $p(s_0)$. The agent then selects action $a \in A$, and receives the reward sampled from the environment $r_t \in R$ with probability $R(s, a)$. The environment then samples from $s_{t+1} \in S$ with probability $p(s_{t+1}|s_t, a_t)$. The agent selects another action and this process continues until the environment terminates. This process is shown in Figure 1.

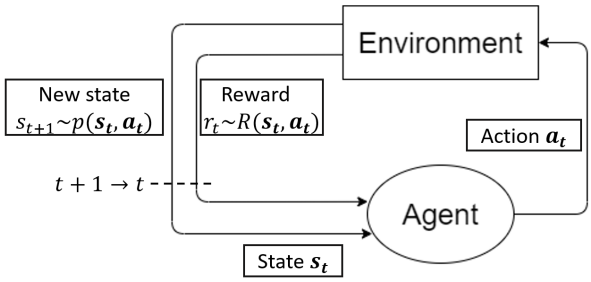


Figure 1. Markov decision process.

The two algorithms of interest—ANN-PSO and PPO—and a benchmark algorithm operate as the agent in the MDP process to select actions. All three algorithms are briefly discussed in Sections 2.2–2.4 and are summarized in Figure 2.

2.2. Artificial Neural Network with Particle Swarm Optimization (ANN-PSO)

For ANN-PSO, an artificial neural network is trained to approximate the environment reward function by mapping states and actions to an expected reward. This is done by minimizing the mean squared error between predicted and actual reward on collected state–action pair data and is shown in Figure 2 (ANN-PSO offline preparation section). Once the ANN accurately predicts rewards from states and actions, particle swarm optimization (PSO) is used on the ANN to find actions that maximize the expected reward for a given state and is shown in Figure 2 (ANN-PSO online computation section). ANN-PSO is an off-policy algorithm meaning the algorithm can learn from data collected from other policies. ANN-PSO's weaknesses include overfitting to noise in stochastic environments and being computationally expensive online since PSO performs many iterations to find optimal actions for each state. Below is a brief explanation of ANN-PSO which uses Mnih et al.'s q-learning formulation to train the ANN and Kennedy and Eberhart PSO algorithm [37,38] to search for optimal actions.

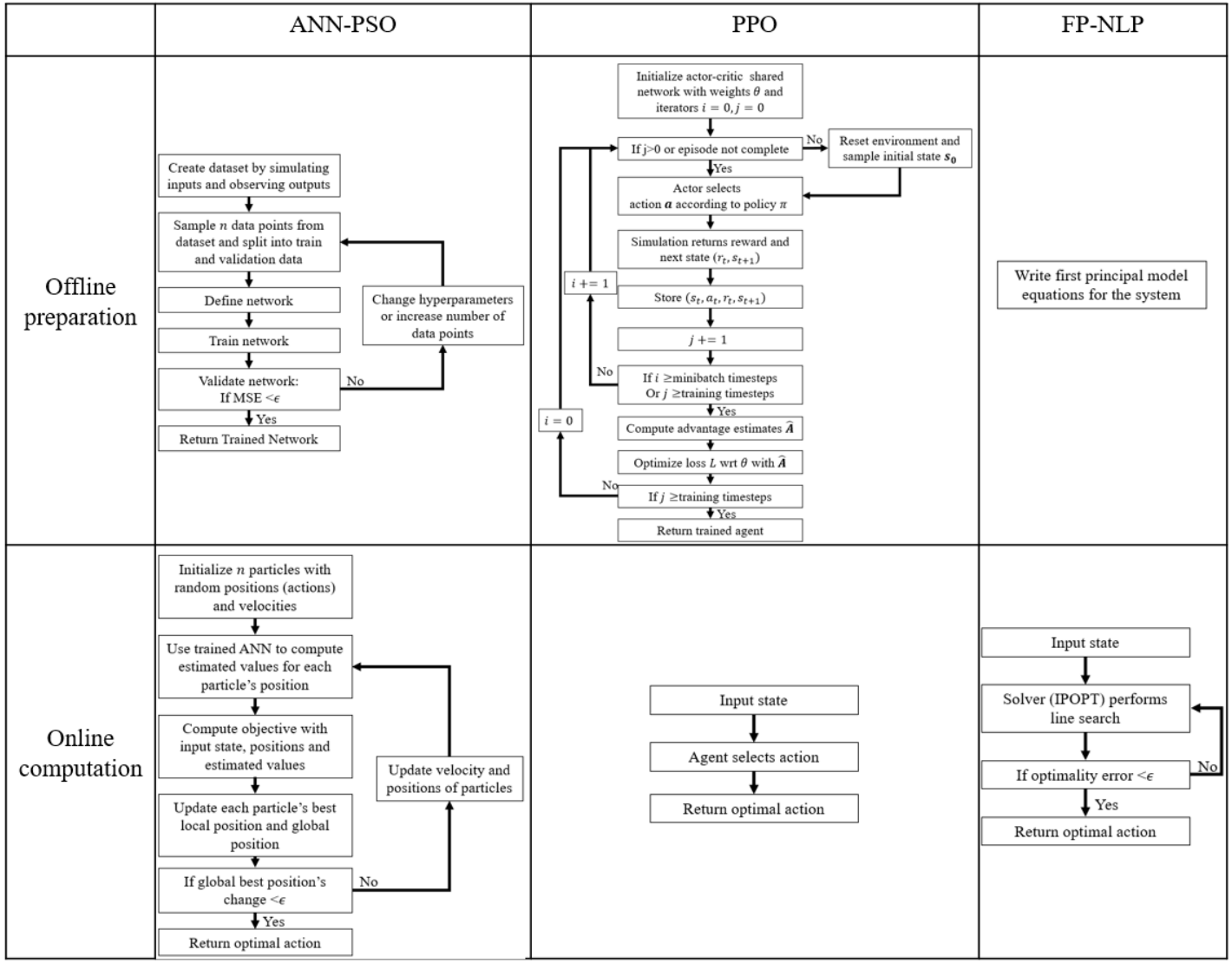


Figure 2. Algorithm flow chart for offline preparation and online operation of a steady state system ($\gamma = 0$).

To maximize the reward, a state–action value function, $Q^{\pi}(s,a)$, is defined in Equation (1). This equation states that, given an initial state, the policy's value of a state–action pair is the expected cumulative reward of policy π.

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] \tag{1}$$

π is a policy mapping states to actions.

An optimal state-action value function $Q^*(s,a)$ satisfies the Bellman equation which is shown in Equation (2).

$$Q^*(s_t,a_t) = \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})|s_t,a_t\right] \tag{2}$$

With Equation (2), Q^* can be estimated with a function approximator such as a linear function or a nonlinear function through iteration. In this paper, an ANN is used to estimate Q^*. Training is done by discretizing each dimension of the state S and action sets A into $\mathcal{N}$ spaces. Every combination of the discretized states and actions are passed to the system, which returns the reward. The neural network is then trained to estimate the reward given a state–action pair. For stochastic environments, the ANN may overfit to noise which causes a plant–model mismatch.

To find the maximum value state–action pair for the estimated Q^*, PSO is used. PSO is a derivative free optimizer which uses multiple particles, collectively called the swarm, to search the solution space to find an optimal action. The particles are initially given random positions and velocities and then are guided by each particle individual best and the swarm best positions to find the optimal solution of function f. The algorithm is shown in Figure 2 (ANN-PSO online computation section).

For ANN-PSO, the actions are the positions and the function to be maximized is the reward returned by the ANN. For every new state, PSO searches for optimal actions. This process is computationally expensive since there are many particles and iterations that must be performed for every new state encountered.

2.3. Proximal Policy Optimization (PPO)

While ANN-PSO finds the optimal action by learning values of every state–action pair and searching for the optimal action, Proximal Policy Optimization [39] learns an optimal policy to sample actions, given states. PPO is an example of an actor–critic algorithm where a policy network is trained to maximize the cumulative reward, while a critic is trained to determine how much better the new policy is and push the probabilities of those actions up or down. PPO is an on-policy algorithm meaning the algorithm learns from data collected based on the algorithm policy, and thus the algorithm requires interaction with the environment.

The PPO algorithm for training is shown in Figure 2 (PPO offline preparation section). While online, the mean action is selected from $\pi_\theta(a_t|s_t)$ and is shown in Figure 2 (PPO online computational section).

For PPO, the policy, denoted as $\pi(a|s)$, is an action probability distribution function, given a state. $\pi(a|s)$ is approximated using a neural network $\pi_\theta(a|s)$ with parameters θ optimized to maximize a utility function $U(\theta)$. $U(\theta)$ is shown in Equation (3) and represents the policy expected reward from time t to time $t + t_f$ starting from state s_t.

$$U(\theta) = \int_{\tau} r(\tau)p(\tau|\theta)d\tau \tag{3}$$

τ is all possible trajectories from state s_t, $r(\tau)$ is the discounted reward of τ, and $p(\tau|\theta)$ is the probability of τ given parameters θ. The probability distribution is typically a normal distribution.

Instead of directly maximizing Equations (3) and (4), a surrogate objective with the same gradient is maximized so constraints can be applied to limit update step sizes.

$$U^*(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta_{old})} \left[r(\tau) \frac{p(\tau|\theta)}{p(\tau|\theta_{old})} \right] \tag{4}$$

However, using $r(\tau)$ in $U(\theta)$ poses a problem as the agent increases probabilities of all trajectories with positive reward which leads to high variance and low sample efficiency. Thus, $r(\tau)$ is replaced with a generalized advantage estimation $\hat{A}_\tau$, which is shown in Equations (5a) and (5b).

$$\hat{A}_\tau = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{5a}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{5b}$$

λ is a factor for trade-off of bias vs. variance, and $V(s_t)$ is the expected episode cumulative discounted reward of state s_t given $\pi_\theta(a|s)$. $V(s_t)$ is approximated with a neural network $V_\theta(s_t)$ and, in this paper, shares parameters with $\pi_\theta(a|s)$.

$\hat{A}_\tau$ provides a baseline by comparing action a_t received reward to the previous policy estimated value of state s_t and pushes the agent to increase probabilities of trajectories that have higher rewards than the current policy. Combining Equations (4) and (5a) yields Equations (6a) and (6b):

$$U^*(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta_{old})} \left[\hat{A}_\tau \eta_\tau(\theta) \right] \tag{6a}$$

$$\eta_\tau(\theta) = \frac{p(\tau|\theta)}{p(\tau|\theta_{old})} \tag{6b}$$

To prevent the agent from making excessively large updates, the surrogate objective is clipped as shown in Equation (7).

$$L^{CLIP}(\theta) = \hat{\mathbb{E}} \left[\min \left(\eta_\tau(\theta)\hat{A}_\tau, \text{clip}(\eta_\tau(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_\tau \right) \right] \tag{7}$$

ϵ is a hyperparameter to limit the policy updates.

This limits how much the policy changes the probabilities of trajectories compared to the old policy since $\eta_\tau(\theta)$ denotes trajectory probability ratios between the new and old policies. As the policy and value function neural networks share parameters in this paper, the objective function incorporates the value function error. An entropy bonus is also added to encourage the agent to explore and avoid premature convergence to a suboptimal solution. The final objective is shown in Equations (8a) and (8b).

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}} \left[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{8a}$$

$$L^{VF} = (V_\theta(s_t) - V_t^{targ})^2 \tag{8b}$$

c_1 and c_2 are hyperparameters, $S[\pi_\theta](s_t)$ is an entropy bonus function, and V_t^{targ} is the observed cumulative discounted reward from time t to the end of the episode.

2.4. First Principles with Nonlinear Programming (FP-NLP)

Maximizing the reward by using nonlinear programming (NLP) on a first principles model (FP-NLP) is a reliable and well-accepted approach [40–42]. Thus, FP-NLP is used as a benchmark to evaluate the performance of the two previous methods. Using FP-NLP requires thorough understanding of the system governing equations, such as fundamental equations for energy and mass balances, to formulate the first principles model. This first principles model can then be formulated into

an optimization problem and solved using NLP algorithms such as an interior-point method for NLP (IPOPT). While this method does not require learning like ANN-PSO and PPO, the requirement for an accurate first principles model makes this method difficult to implement in systems where accurate models are difficult to develop. Furthermore, FP-NLP requires an iterative line search, as shown in Figure 2 (FP-NLP online computation section), which can lead to long computational times.

3. Case Study

These methods were evaluated by having them perform economic optimization of a steady state continuously stirred tank reactor (CSTR) model as shown in Figure 3. The values for the model constants are listed at the end of this section in Table 1. The CSTR is assumed to be well mixed. A mass balance with species A and B flowing in at rates of $\dot{v}_A$ and $\dot{v}_B$, respectively, is shown in Equation (9).

$$\dot{v}_A + \dot{v}_B = \dot{v} = c\sqrt{h} \tag{9}$$

c is a valve constant and h is the liquid height in the tank.

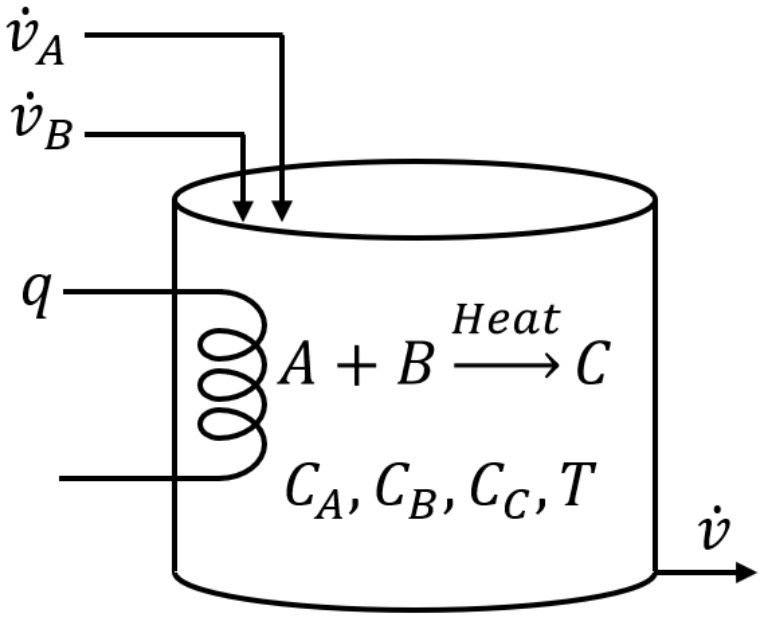


Figure 3. CSTR model.

Inside the CSTR, an elementary, liquid phase, endothermic, irreversible reaction occurs with Equation (10).

$$A + B \rightarrow C \tag{10}$$

The rate $-r_A$ is given by Equation (11).

$$-r_A = kC_AC_B \tag{11}$$

C_A and C_B are the concentrations of A and B in the reactor, and k is related to temperature with Arrhenius equation as shown in Equation (12).

$$k = \mathcal{A}e^{\frac{-E_a}{RT}} \tag{12}$$

$\mathcal{A}$ is the Arrhenius pre-exponential factor, E_a is the activation energy of the reaction, R is the universal gas constant, and T is the temperature of the reactor.

Table 1. Model simulation parameters.

Parameter	Value	Unit
c	1.2	$m^{2.5} \cdot min^{-1}$
h	8	m
$\mathcal{A}$	0.23	$m^3 \cdot kmol^{-1} \cdot min^{-1}$
E_a	5000	$kJ \cdot kmol^{-1}$
R	8.314	$kJ \cdot kmol^{-1} \cdot K{-}1$
ρ	780	$kg \cdot m^{-3}$
c_p	3.25	$kJ \cdot kg^{-1} \cdot K^{-1}$
$C_{A,0}$	15	$kmol \cdot m^{-3}$
$C_{B,0}$	15	$kmol \cdot m^{-3}$
T_0	500	K
T_{ub}	800	K
V	100.53	m^3
ΔH_{rxn}	156	$kJ \cdot kmol^{-1}$
$P_{A,low}$	2	$\$ \cdot m^{-3}$
$P_{A,high}$	20	$\$ \cdot m^{-3}$
$P_{B,low}$	5	$\$ \cdot m^{-3}$
$P_{B,high}$	50	$\$ \cdot m^{-3}$
$P_{C,low}$	2	$\$ \cdot kmol^{-1}$
$P_{C,high}$	20	$\$ \cdot kmol^{-1}$
$P_{q,low}$	0	$\$ \cdot kJ^{-1}$
$P_{q,high}$	2×10^{-5}	$\$ \cdot kJ^{-1}$
σ	0.1	
N	120	

As the system is at steady state, the mole and energy balance equations for the system are written as Equations (13)–(16).

$$\frac{dC_A}{dt} = 0 = (\dot{v} - \dot{v}_B)C_{A,0} - \dot{v}C_A - VA\exp\left(\frac{-E_a}{RT}\right)C_A C_B \tag{13}$$

$$\frac{dC_B}{dt} = 0 = \dot{v}_B C_{B,0} - \dot{v}C_B - VA\exp\left(\frac{-E_a}{RT}\right)C_A C_B \tag{14}$$

$$\frac{dC_C}{dt} = 0 = -\dot{v}C_C + VA\exp\left(\frac{-E_a}{RT}\right)C_A C_B \tag{15}$$

$$\frac{dE}{dt} = 0 = \dot{v}\rho c_p(T_0 - T) - V\Delta H_{rxn}A\exp\left(\frac{-E_a}{RT}\right)C_A C_B + q \tag{16}$$

A and B are assumed to have similar densities and heat capacities of ρ and c_p, respectively, which remain constant over the operating temperature range. A and B enter with concentrations $C_{A,0}$ and $C_{B,0}$, respectively, T_0 is the feed temperature of both species, V is the volume of the reactor, ΔH_{rxn} is the heat of reaction, and q is the heat input.

Solving for C_B using Equation (13) yields Equation (17).

$$C_B = \frac{(\dot{v} - \dot{v}_B)C_{A,0} - \dot{v}C_A}{VA\exp\left(\frac{-E_a}{RT}\right)C_A} \tag{17}$$

Plugging this back into Equation (14) yields Equations (18a)–(18d).

$$a_0 = \frac{-(\dot{v} - \dot{v}_B)C_{A,0}\dot{v}}{VA\exp\left(\frac{-E_a}{RT}\right)} \tag{18a}$$

$$a_1 = \dot{v}_B C_{B,0} + \frac{\dot{v}^2}{VA\exp\left(\frac{-E_a}{RT}\right)} - (\dot{v} - \dot{v}_B)C_{A,0} \tag{18b}$$

$$a_2 = \dot{v} \tag{18c}$$

$$C_A = \frac{-a_1 + \sqrt{a_1^2 - 4a_0a_2}}{2a_2} \tag{18d}$$

Thus, C_C and q can be written as Equations (19) and (20).

$$C_C = \frac{VA\exp\left(\frac{-E_a}{RT}\right)C_AC_B}{\dot{v}} \tag{19}$$

$$q = V\Delta H_{rxn}A\exp\left(\frac{-E_a}{RT}\right)C_AC_B - \dot{v}\rho C_p(T_0 - T) \tag{20}$$

The cost per time (CPT) function is defined as Equation (21).

$$CPT = (\dot{v} - \dot{v}_B)P_A + \dot{v}_BP_B + qP_q - \dot{v}C_CP_C \tag{21}$$

P_i is the price of commodity i and is sampled from a uniform distribution ranging from $P_{i,low}$ to $P_{i,high}$.

The agents can manipulate $\dot{v}_B$ within the range of $[0, \dot{v}]$ and the temperature set point (Temp SP) within $[T_0, T_{ub}]$ where T_{ub} is the upper temperature limit for steady state operation of the CSTR. The goal is to minimize the cost so the optimization problem can be formulated as Equation (22a) subject to Equations (17), (18a)–(18d), (19), (20), (21), (22b) and (22c).

$$\min_{\dot{v}_B, T} \quad CPT \tag{22a}$$

$$\text{s.t.} \quad \text{Equations. } (17), (18a) - (18d), (19), (20), (21)$$

$$T_0 \leq T \leq T_{ub} \tag{22b}$$

$$0 \leq \dot{v}_B \leq \dot{v} \tag{22c}$$

For the MDP formulation, the possible states are all combinations of prices, and the possible actions are all combinations of $\dot{v}_B$ and T. $p(s_0)$ is a uniform distribution over the ranges of all four prices, and the reward function is $-CPT$ in Equation (21).

To make the model more realistic, feed concentrations and temperatures were stochastic and sampled from a normal distribution with a given mean and a relative standard deviation of σ. Actions were also perturbed by sampling from a normal distribution with a mean of the chosen action and a relative standard deviation of σ. For each set of prices, the agent selects one action and the simulation is repeated N times. The average reward is then calculated and returned.

4. Evaluation

Simulations were run on an Intel(R) Core(TM) i7-6500U CPU with 2.50 GHz clock rate and 8 GB of RAM.

4.1. ANN-PSO Implementation

The neural network was trained to map prices with actions to profits. The prices, P_A, P_B,P_C, and P_q, and actions $\dot{v}_B$ and T were each discretized into $\mathcal{N}$ spaces. $\mathcal{N}$ ranges from 3 to 14 to test the performance

of ANN-PSO given various amounts of training data. All combinations of the prices and actions ($\mathcal{N}^6$) are fed through the reactor simulation and the profit is returned. The ANNs were configured and trained using Keras [43] with an Adam optimizer. The network trained on the minimum-maximum scaled data and k-fold validation was used to evaluate the networks. The network architecture was optimized by varying number of layers, activation functions, regularization coefficients, and learning rate. The network trained with $N = 13$ discretization points performed best and was used for algorithm comparisons. PSO was done using Pyswarms [44]. The hyperparameters can be found in Table A1.

4.2. PPO Implementation

PPO training was done using Stable Baselines, which is an RL python package [45]. The PPO agents were trained for $14^6 \approx 7.5 \times 10^6$ time steps and agents were saved periodically to investigate performance improvement over training time. The agents were trained in a normalized vectorized environment and the network used was 2 fully connected layers of 64 units followed by a long short-term memory layer of 256 units. This was then split into the value and policy functions. Hyperparameters, such as the entropy coefficient, value function coefficient, activation functions, and number of epochs were varied to optimize the agent performance. The final network hyperparameters as well as the particle swarm hyperparameters are listed in Table A2. For evaluation, action means from the action probability distributions were used.

4.3. Benchmark Algorithms

For FP-NLP, the case model study was coded into GEKKO, an optimization suite [46]. IPOPT was then used as the NLP solver to minimize the cost. Another strategy evaluated was maximizing the production of product C regardless of the prices. The actions for maximizing production were $T = T_{ub}$ and $\dot{v}_B = \frac{\dot{v}}{2}$.

4.4. Testing and Metrics

To evaluate the agents' overall performance in profitability, sample efficiency and computational time, ANN-PSO agents, PPO agents, FP-NLP, and maximum production chose actions for the same 1000 random price combinations. The random prices are not necessarily examples the agents have seen before, but they are within the range of prices the agents have encountered. The received profits from the stochastic system, predicted profits, and computational times were saved and compared between algorithms. To examine the agent actions and their effects on profits, a sensitivity analysis was performed by holding the prices of B and C at their average values of 27.5 \$/m^3 and 11 \$/kmol, respectively. Prices for A and q were discretized into 500 points and ranged from 2 to 20 \$/m^3 and 0 to 2×10^{-5} \$/kJ, respectively. The agents' selected actions for each combination of A and q and the simulation returned profit were recorded. For the sensitivity analysis, the simulation was made deterministic by setting all random variables to their mean.

5. Results

5.1. PPO Achieves Higher Profits, but ANN-PSO Has Better Training Efficiency

The methods' average profits for the 1000 random price evaluation are shown in Figure 4. FP-NLP has the highest average profit at \$139.28/min and is followed by PPO best profit of \$139.20/min. ANN-PSO best profit is \$138.37/min and max production has a mean profit of \$120.23/min. This shows both methods achieve a higher profit than the maximum production strategy with ANN-PSO and PPO increase the average profit by 15% and 16%, respectively. PPO is also shown to perform as well as FP-NLP with PPO achieving 99.9% the profit of FP-NLP.

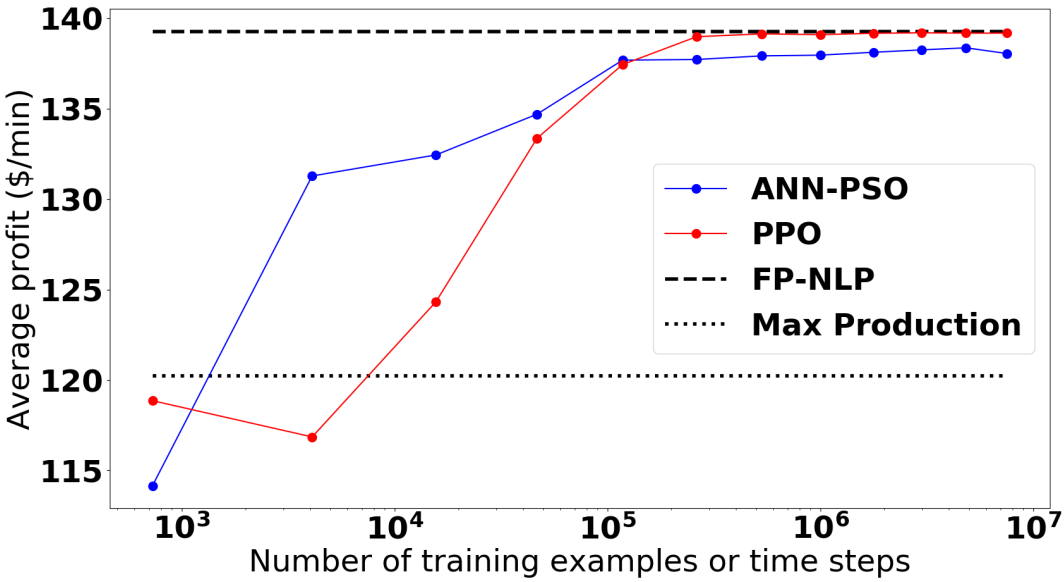


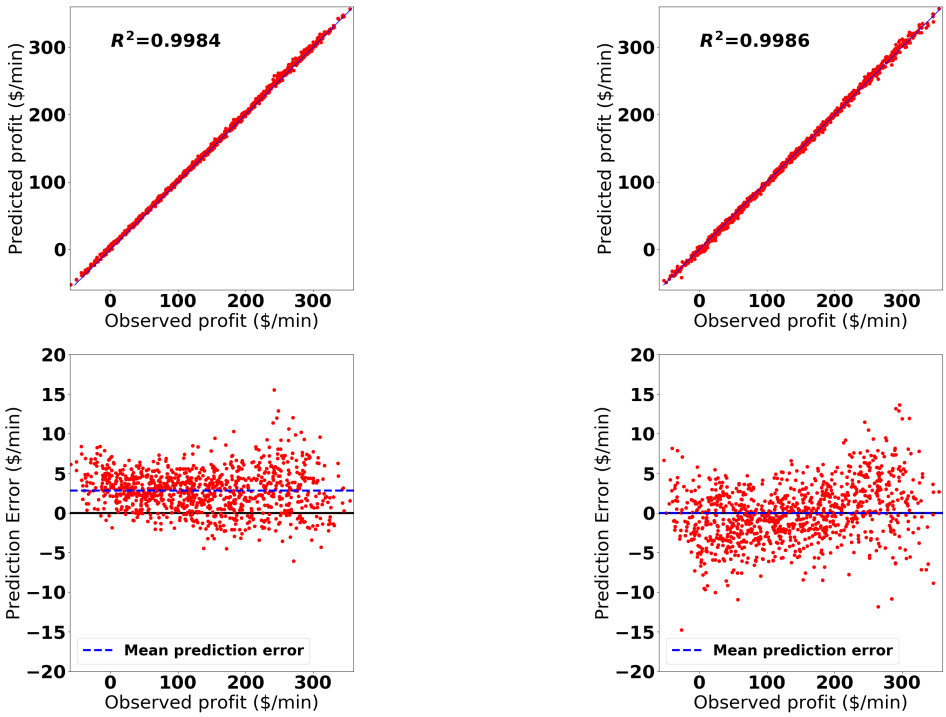
Figure 4. Methods' learning curves on continuously stirred tank reactor (CSTR) optimization.

Figure 4 also shows ANN-PSO has better training efficiency than PPO, as ANN-PSO profit overtakes the max production benchmark with $\approx 4.1 \times 10^3$ data points and plateaus with $\approx 1.2 \times 10^5$ data points as compared to PPO, which overtakes the max production benchmark with roughly four times the amount of data at $\approx 1.6 \times 10^4$ data points and plateaus at $\approx 2.6 \times 10^5$ data points. PPO's lower training efficiency is due to needing to learn both the policy and the value of the state, while ANN-PSO only needs to learn the value of the state–action pairs. Furthermore, ANN-PSO exploration is determined by the fed data while PPO explores based on the stochastic policy. This allows for easier implementation of ANN-PSO since normal operation data—such as when PID controllers or expert personnel controls the process—can be used as training data.

More training examples likely would not improve performance of either method as ANN-PSO and PPO's average profit starts oscillating near 10^7 training examples.

While ANN-PSO training efficiency is appealing, PPO still converges to policy with higher profits. This can be explained by examining the neural networks' prediction accuracy, shown by parity and residual plots in Figure 5. Figure 5a shows that ANN-PSO neural network accurately predicts the agent profits, as seen by the high R^2 value of 0.998. However, if the residual plot is examined, the prediction error has a mean of almost \$3/min, indicating the prediction network is overestimating the profit. This is due to the PSO algorithm exploiting small errors in the neural network that arise from overfitting to stochastic data. Figure 5b shows that PPO value network has a similarly high R^2 value. The residual plot shows that PPO's profit prediction errors are evenly distributed around 0 with a mean of -0.01. PPO does not overpredict the profit due to the update clipping, stochastic policy, and online training. The update clipping both prevents the critic from overfitting to noise and the actor from exploiting the critic prediction errors. The stochastic policy helps to further prevent critic error exploitation, and the online training allows the PPO agent to explore actions with high predicted profit and update the critic with the observed reward.

Figure 6 shows the computational times of the algorithms to return the optimal action. ANN-PSO has the longest average computational time of 41 s. PPO and FP-NLP have relatively fast computational times of 0.002 s and 0.02 s, respectively. This highlights the fast computational time of PPO since the computation is one forward pass through the policy neural network. Both FP-NLP and ANN-PSO require iterations to find the optimal action, which results in longer computational times.



(**a**) (**b**)

Figure 5. Parity plots (Row 1) and residual plots (Row 2) for ANN-PSO and Proximal Policy Optimization (PPO) value functions. (**a**) ANN-PSO action-value function; (**b**) PPO value function.

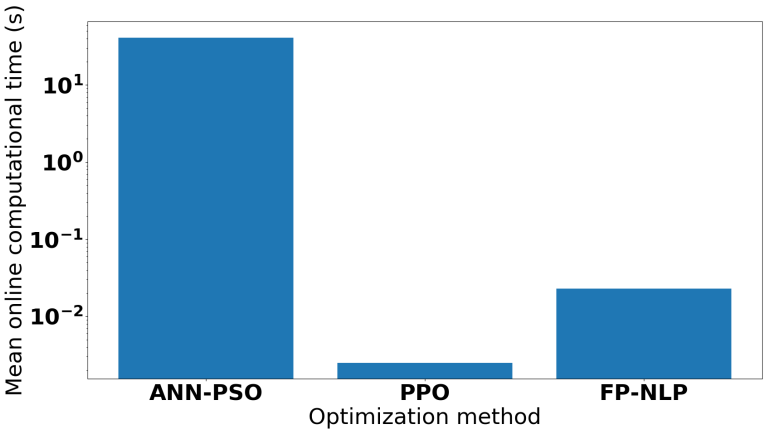


Figure 6. Online average computational times for ANN-PSO, PPO, and FP-NLP.

5.2. Sensitivity Analysis

The results of the sensitivity analysis are shown in Figure 7. For these pricing cases, the temperature set point selection is the main variable affecting price, while the flow of B can be held nearly constant. All three methods have similar profit contour plots, but ANN-PSO has a more jagged profile and overall darker contour indicating the agent had a lower profit compared to the other methods. This is caused by PSO exploiting the errors in the ANN, which is shown clearly in the ANN-PSO flow of B actions. While FP-NLP flow of B ranges from 1.63 to 1.68 m^3/min, ANN-PSO ranges from 1.6 to 1.73 m^3/min. Overall, the ANN-PSO contour plot appears random showing how PSO finds apparent optimal actions in areas the ANN overpredicts the reward. The erroneous flow of B may contribute to the suboptimal temperature set point selection as ANN-PSO typically selects lower temperatures compared to FP-NLP when P_q is between 0.25 to 0.75 \$/kJ. The temperature profile is also jagged, indicating the PSO is again exploiting the ANN model overprediction.

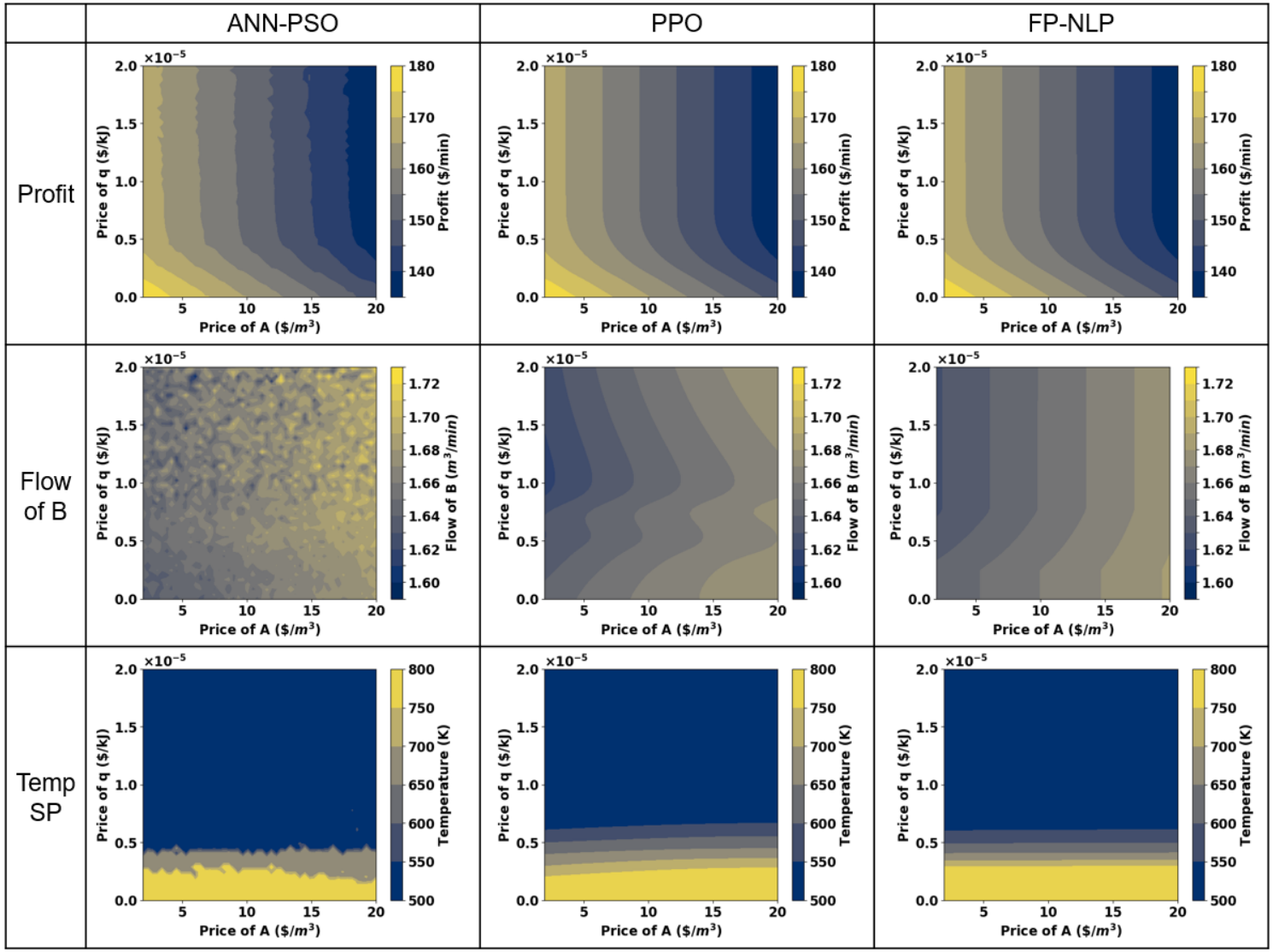


Figure 7. Contour plots for methods' profits, flows of B, and temperature set points over prices of A and q.

PPO on the other hand has smoother actions and the profiles appear to be very similar to FP-NLP actions. The main reason for the smooth action contour plots is because instead of optimizing a complicated ANN, PPO is mapping the state to actions with a continuous function. The main difference between FP-NLP and PPO actions is in the flow of B, but the difference between the actions is negligible. PPO is choosing actions based on the training with the stochastic environment, and FP-NLP is choosing actions based on a perfect deterministic environment. This causes PPO's actions to be slightly suboptimal in the tested deterministic environment compared to FP-NLP.

6. Conclusions

In this study, we compared the performance and applicability of two RL algorithms—ANN-PSO and PPO—by exploring their methods and applying them on stochastic steady-state economic optimization of a CSTR with FP-NLP as a benchmark algorithm [4–6]. We evaluate the RL algorithms' performance with their profitability and online computational times, and their applicability with their data requirements and training efficiencies. On the case study, PPO shows better performance as its average profits are higher compared to ANN-PSO and its online computational times are faster. ANN-PSO shows better applicability as the algorithm can train with normal operational data and converges to a policy with less training data. Both algorithms perform similarly to FP-NLP with ANN-PSO and PPO achieving 99.3% and 99.9% of FP-NLP's average profit, respectively. PPO has the fastest online computational time, 10 and 10,000 times faster than FP-NLP and ANN-PSO, respectively. While these results are based on this case study, they may be useful guidelines for more complex applications.

Investigation of the ANN-PSO residual plots reveal PSO exploits overpredictions in the ANN, resulting in suboptimal actions. PPO does not have this issue as it has update clipping and a stochastic policy. The ANN exploitation is also seen in a similar algorithm named deep deterministic policy gradient. Some methods to combat this issue are to clip the artificial neural network and perturb actions to prevent ANN error exploitation [47].

As ANN-PSO has high training efficiency, researching methods to limit value overprediction to improve performance, such as training two ANNs and taking the minimum predicted profit of the two, would be beneficial. Other off-policy algorithms should also be investigated like twin-delayed deep deterministic policy gradients and V-trace to observe their performance on a chemical system after training on operational data. Other improvements to ANN-PSO include decreasing the computational time. Our previous work shows replacing PSO with an actor ANN can significantly reduce the online computational time and find an optimal policy [48]. As PPO and other on-policy algorithms requires interaction with an environment, they should be explored in their ability to augment FP-NLP. Another research topic is exploring how these algorithms perform with more complex systems or when the system changes such as multiple reactions, reversible reactions, or heat exchanger fouling.

Author Contributions: T.Q.: Conceptualization, Methodology, Software, Investigation, and writing—original draft. D.M.: Supervision and Writing—review and editing. K.M.P.: Supervision, Funding acquisition, and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Acknowledgments: This research is funded by the University of Utah.

Conflicts of Interest: All the authors declare no conflicts of interest.

Nomenclature

Acronyms

ANN	Artificial neural network
ANN-PSO	Artificial neural network with particle swarm optimization
CSTR	Continuously stirred tank reactor
FP-NLP	First principles with nonlinear programming
MDP	Markov decision process
NLP	Nonlinear programming
PPO	Proximal policy optimization
PSO	Particle swarm optimization
Temp SP	Temperature set point

MDP symbols

γ	Factor to discount future rewards
$\mathbb{R}$	Real numbers
A	Set of possible actions in the environment
a_t	Action at time t
m	Dimension of action space A
n	Dimension of state space S
$p(s_t)$	Probability distribution for state s_t
$R(s_t, a_t)$	Probability distribution of rewards given s_t and a_t
r_t	Reward received at time t
S	Set of possible states in the environment
s_t	State at time t
t	Time after environment episode starts

ANN-PSO symbols

$\mathbb{E}[x]$	Expected value function of x
$\mathcal{N}$	Number of times state and action sets are discretized for training data generation
π	Function which maps states to actions
$Q^{\pi}(s, a)$	Value of a state-action pair given policy π

PPO symbols

ϵ	Coefficient to limit policy updates in L^{CLIP}
$\eta_{\tau}(\theta)$	Trajectory τ probability ratio between new and old policies
$\hat{A}_{\tau}$	Generalized estimated advantage of parameters θ over θ_{old}
λ	Factor for bias vs. variance trade-off
τ	Trajectory starting from state s_t
θ	Vector of policy and value neural networks parameters
c_1	Coefficient to weight value function loss in $L^{CLIP+VF+S}(\theta)$
c_2	Coefficient to weight entropy bonus in $L^{CLIP+VF+S}(\theta)$
$L^{CLIP+VF+S}(\theta)$	Final objective function to maximize L^{CLIP}, minimize $V_{\theta}(s_t)$ mean, and encourage exploration
$L^{CLIP}(\theta)$	Clipped objective function to prevent excessively large updates
$L^{VF}(\theta)$	Mean squared error between $V_{\theta}(s_t)$ and V_t^{targ}
$r(\tau)$	Discounted reward of trajectory τ
$S[\pi_{\theta}](s_t)$	Entropy bonus function for exploration encouragement
t_f	Time length of trajectories τ
$U(\theta)$	Policy expected cumulative reward from time t to time $t + t_f$ starting from state s_t given parameters θ
$V_{\theta}(s_t)$	Expected episode cumulative discounted reward given policy with parameters θ
V_t^{targ}	Observed cumulative reward from time t to the end of the episode

Case study symbols

ΔH_{rxn}	Heat of reaction

$\dot{v}$	Total Flow Rate
$\dot{v}_i$	Flow rate of species i
$\mathcal{A}$	Arrhenius constant
ρ	Density of all species
σ	Relative standard deviation for stochastic parameters
c	Valve constant
C_i	Concentration of species i
c_p	Specific heat capacity of all species
$C_{i,0}$	Feed concentration of species i
CPT	Cost per time
E_a	Reaction activation energy
h	Height of CSTR
k	Reaction constant
N	Number of times simulation is repeated for one set of prices and actions
P_i	Price of commodity i
$P_{i,high}$	High price of commodity i
$P_{i,low}$	Low price of commodity i
q	Heat input rate
R	Universal gas constant
r_A	Reaction rate
T	Temperature of CSTR contents
T_0	Feed temperature
T_{ub}	Upper limit on CSTR temperature
V	Volume of CSTR

Appendix A. Hyperparameter Tables

Table A1. ANN-PSO hyperparameters.

Hyperparameter	Value
Adam learning rate	1×10^{-4}
Adam exponential decay rate for the first-moment estimates	0.9
Adam exponential decay rate for the second-moment estimates	0.999
Adam small number to prevent any division by zero	10^{-7}
ANN layers	1
ANN nodes	64
ANN activation function	ReLU
ANN L1 norm regularization coefficient	10^{-4}
ANN Loss function	Mean squared error
PSO cognitive parameter	1×10^{-3}
PSO social parameter	1×10^{-3}
PSO inertia parameter	0.9
PSO swarm size	100
PSO lower velocity bound	1×10^{-8}
PSO upper velocity bound	1
PSO iterations	100

Table A2. PPO hyperparameters.

Hyperparameter	Value
Policy Value Shared Network layers	2 Fully connected layers + 1 LSTM layer
Policy Value Shared Network structure	[64,64,128]
γ	0
n_steps (t_f)	100
ent_coef (c_2)	0.01
learning_rate	2.5×10^{-3}
vf_coef (c_1)	0.5
max_grad_norm	0.5
lam (λ)	0.95
nminibatches	4
noptepochs	4
cliprange (ϵ)	0.2

References

1. Dotoli, M.; Fay, A.; Miskowicz, M.; Seatzu, C. A survey on advanced control approaches in factory automation. *IFAC-PapersOnLine* **2015**, *28*, 394–399. [CrossRef]
2. Tuttle, J.F.; Vesel, R.; Alagarsamy, S.; Blackburn, L.D.; Powell, K. 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.* **2019**, *93*, 104167. [CrossRef]
3. Petsagkourakis, P.; Sandoval, I.O.; Bradford, E.; Zhang, D.; del Rio-Chanona, E.A. Reinforcement learning for batch bioprocess optimization. *Comput. Chem. Eng.* **2020**, *133*, 106649. [CrossRef]
4. Sheha, M.; Powell, K. Using Real-Time Electricity Prices to Leverage Electrical Energy Storage and Flexible Loads in a Smart Grid Environment Utilizing Machine Learning Techniques. *Processes* **2019**, *7*, 870. [CrossRef]
5. Sheha, M.; Mohammadi, K.; Powell, K. Solving the Duck Curve in a Smart Grid Environment Using a Non-Cooperative Game Theory and Dynamic Pricing Profiles. *Energy Convers. Manag.* **2020**, *220*, 113102. [CrossRef]
6. Sheha, M.; Mohammadi, K.; Powell, K. Techno-economic analysis of the impact of dynamic electricity prices on solar penetration in a smart grid environment with distributed energy storage. *Appl. Energy* **2021**, *282*, 116168. [CrossRef]
7. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
8. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef]
9. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef]
10. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with large scale deep reinforcement learning. *arXiv* **2019**, arXiv:1912.06680.
11. Argyros, I.K. *Undergraduate Research at Cameron University on Iterative Procedures in Banach and Other Spaces*; Nova Science Publishers: Hauppauge, NY, USA, 2019.
12. Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell, G.; Ribas, R.; et al. Solving Rubik's Cube with a Robot Hand. unpublished. Available online: http://xxx.lanl.gov/abs/1910.07113 (accessed on 20 March 2020)
13. Buyukada, M. Co-combustion of peanut hull and coal blends: Artificial neural networks modeling, particle swarm optimization and Monte Carlo simulation. *Bioresour. Technol.* **2016**, *216*, 280–286. [CrossRef]
14. Blackburn, L.D.; Tuttle, J.F.; Powell, K.M. Real-time optimization of multi-cell industrial evaporative cooling towers using machine learning and particle swarm optimization. *J. Clean. Prod.* **2020**, *271*, 122175. [CrossRef]
15. Naserbegi, A.; Aghaie, M. Multi-objective optimization of hybrid nuclear power plant coupled with multiple effect distillation using gravitational search algorithm based on artificial neural network. *Therm. Sci. Eng. Prog.* **2020**, *19*, 100645. [CrossRef]

16. Head, J.D.; Lee, K.Y. Using artificial neural networks to implement real-time optimized multi-objective power plant control in a multi-agent system. *IFAC Proc. Vol.* **2012**, *8*, 126–131. [CrossRef]
17. Bhattacharya, S.; Dineshkumar, R.; Dhanarajan, G.; Sen, R.; Mishra, S. Improvement of ϵ-polylysine production by marine bacterium Bacillus licheniformis using artificial neural network modeling and particle swarm optimization technique. *Biochem. Eng. J.* **2017**, *126*, 8–15. [CrossRef]
18. Khajeh, M.; Dastafkan, K. Removal of molybdenum using silver nanoparticles from water samples: Particle swarm optimization-artificial neural network. *J. Ind. Eng. Chem.* **2014**, *20*, 3014–3018. [CrossRef]
19. Dhanarajan, G.; Mandal, M.; Sen, R. A combined artificial neural network modeling-particle swarm optimization strategy for improved production of marine bacterial lipopeptide from food waste. *Biochem. Eng. J.* **2014**, *84*, 59–65. [CrossRef]
20. Ghaedi, M.; Ghaedi, A.M.; Ansari, A.; Mohammadi, F.; Vafaei, A. Artificial neural network and particle swarm optimization for removal of methyl orange by gold nanoparticles loaded on activated carbon and Tamarisk. *Spectrochim. Acta Part A Mol. Biomol. Spectrosc.* **2014**, *132*, 639–654. [CrossRef]
21. Khajeh, M.; Kaykhaii, M.; Hashemi, S.H.; Shakeri, M. Particle swarm optimization-artificial neural network modeling and optimization of leachable zinc from flour samples by miniaturized homogenous liquid-liquid microextraction. *J. Food Compos. Anal.* **2014**, *33*, 32–38. [CrossRef]
22. Nezhadali, A.; Shadmehri, R.; Rajabzadeh, F.; Sadeghzadeh, S. Selective determination of closantel by artificial neural network- genetic algorithm optimized molecularly imprinted polypyrrole using UV–visible spectrophotometry. *Spectrochim. Acta Part A Mol. Biomol. Spectrosc.* **2020**, *243*, 118779. [CrossRef]
23. Abdullah, S.; Chandra Pradhan, R.; Pradhan, D.; Mishra, S. Modeling and optimization of pectinase-assisted low-temperature extraction of cashew apple juice using artificial neural network coupled with genetic algorithm. *Food Chem.* **2020**, 127862. [CrossRef]
24. Bagheri-Esfeh, H.; Safikhani, H.; Motahar, S. Multi-objective optimization of cooling and heating loads in residential buildings integrated with phase change materials using the artificial neural network and genetic algorithm. *J. Energy Storage* **2020**, *32*, 101772. [CrossRef]
25. Ilbeigi, M.; Ghomeishi, M.; Dehghanbanadaki, A. Prediction and optimization of energy consumption in an office building using artificial neural network and a genetic algorithm. *Sustain. Cities Soc.* **2020**, *61*, 102325. [CrossRef]
26. Solís-Pérez, J.E.; Gómez-Aguilar, J.F.; Hernández, J.A.; Escobar-Jiménez, R.F.; Viera-Martin, E.; Conde-Gutiérrez, R.A.; Cruz-Jacobo, U. Global optimization algorithms applied to solve a multi-variable inverse artificial neural network to improve the performance of an absorption heat transformer with energy recycling. *Appl. Soft Comput. J.* **2019**, *85*, 105801. [CrossRef]
27. Filipe, J.; Bessa, R.J.; Reis, M.; Alves, R.; Póvoa, P. Data-driven predictive energy optimization in a wastewater pumping station. *Appl. Energy* **2019**, *252*, 113423. [CrossRef]
28. Zhou, S.; Hu, Z.; Gu, W.; Jiang, M.; Chen, M.; Hong, Q.; Booth, C. Combined heat and power system intelligent economic dispatch: A deep reinforcement learning approach. *Int. J. Electr. Power Energy Syst.* **2020**, *120*, 106016. [CrossRef]
29. Zhang, B.; Hu, W.; Cao, D.; Huang, Q.; Chen, Z.; Blaabjerg, F. Deep reinforcement learning-based approach for optimizing energy conversion in integrated electrical and heating system with renewable energy. *Energy Convers. Manag.* **2019**, *202*, 112199. [CrossRef]
30. Rummukainen, H.; Nurminen, J.K. Practical reinforcement learning—Experiences in lot scheduling application. *IFAC-PapersOnLine* **2019**, *52*, 1415–1420. [CrossRef]
31. Hofstetter, J.; Bauer, H.; Li, W.; Wachtmeister, G. Energy and Emission Management of Hybrid Electric Vehicles using Reinforcement Learning. *IFAC-PapersOnLine* **2019**, *52*, 19–24. [CrossRef]
32. Philipsen, M.P.; Moeslund, T.B. Intelligent injection curing of bacon. *Procedia Manuf.* **2019**, *38*, 148–155. [CrossRef]
33. Xiong, W.; Lu, Z.; Li, B.; Wu, Z.; Hang, B.; Wu, J.; Xuan, X. A self-adaptive approach to service deployment under mobile edge computing for autonomous driving. *Eng. Appl. Artif. Intell.* **2019**, *81*, 397–407. [CrossRef]
34. Pi, C.H.; Hu, K.C.; Cheng, S.; Wu, I.C. Low-level autonomous control and tracking of quadrotor using reinforcement learning. *Control. Eng. Pract.* **2020**, *95*, 104222. [CrossRef]
35. Machalek, D.; Quah, T.; Powell, K.M. Dynamic Economic Optimization of a Continuously Stirred Tank Reactor Using Reinforcement Learning. In Proceedings of the 2020 American Control Conference (ACC), Denver, CO, USA, 1–3 July 2020; pp. 2955–2960.

36. Hubbs, C.D.; Li, C.; Sahinidis, N.V.; Grossmann, I.E.; Wassick, J.M. A deep reinforcement learning approach for chemical production scheduling. *Comput. Chem. Eng.* **2020**, *141*, 106982. [CrossRef]
37. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M.A. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602.
38. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
39. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
40. Grossmann, I.E.; Apap, R.M.; Calfa, B.A.; García-Herreros, P.; Zhang, Q. Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty. *Comput. Chem. Eng.* **2016**, *91*, 3–14. [CrossRef]
41. Biegler, L.T.; Yang, X.; Fischer, G.A. Advances in sensitivity-based nonlinear model predictive control and dynamic real-time optimization. *J. Process. Control* **2015**, *30*, 104–116. [CrossRef]
42. Diehl, M.; Bock, H.G.; Schlöder, J.P.; Findeisen, R.; Nagy, Z.; Allgöwer, F. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *J. Process. Control* **2002**, *12*, 577–585. [CrossRef]
43. Chollet, F.; Rahman, F.; Lee, T.; Marmiesse, G.; Zabluda, O.; Santana, E.; McColgan, T.; Snelgrove, X.; Branchaud-Charron, F.; Oliver, M.; et al. Keras. 2015. Available online: https://keras.io (accessed on 26 March 2020).
44. Miranda, L. Pyswarms. 2017. Available online: https://github.com/ljvmiranda921/pyswarms (accessed on 13 April 2020).
45. Hill, A.; Raffin, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; Traore, R.; Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; et al. Stable Baselines. 2018. Available online: https://github.com/hill-a/stable-baselines (accessed on 26 March 2020).
46. Beal, L.; Hill, D.; Martin, R.; Hedengren, J. GEKKO Optimization Suite. *Processes* **2018**, *6*, 106. [CrossRef]
47. Fujimoto, S.; Van Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. *arXiv* **2018**, arXiv:1802.09477.
48. Powell, K.M.; Machalek, D.; Quah, T. Real-Time Optimization using Reinforcement Learning. *Comput. Chem. Eng.* **2020**, *143*, 107077. [CrossRef]