Process Control using Deep Reinforcement Learning [⋆]

Steven Spielberg Pon Kumar [*] **Bhushan Gopaluni** [**] **Philip Loewen** [***]

[*] *University of British Columbia, Vancouver, Canada (email: spiel@mail.ubc.ca)*
[**] *University of British Columbia, Vancouver, Canada (e-mail: bhushan.gopaluni@ubc.ca)*
[***] *University of British Columbia, Vancouver, Canada (e-mail: loew@math.ubc.ca)*

Abstract: The conventional and optimization based controllers have been used in process industries for more than two decades. The application of such controllers on complex systems could be computationally demanding and may require estimation of hidden states. They also require constant tuning, development of a mathematical model (first principle or empirical), design of control law which are tedious. Moreover, they are not adaptive in nature. On the other hand, in the recent years, there has been significant progress in the fields of computer vision and natural language processing that followed the success of deep learning. Human level control has been attained in games and physical tasks by combining deep learning with reinforcement learning. They were also able to learn the complex go game which has states more than number of atoms in the universe. Self-Driving cars, machine translation, speech recognition etc started to gain advantage of these powerful models. The approach to all of them involved problem formulation as a learning problem. Inspired by these applications, in this work we have posed process control problem as a learning problem to build controllers to address the limitations existing in current controllers.

Keywords: Artificial intelligence, Neural Networks, Process Control, Adaptive Control, Parallel Computations

1. INTRODUCTION

One of the dreams of the field of process control is to enable controllers to understand the plant dynamics fully and then act optimally to control it. It could be challenging because online learning is relatively easy for human beings because of the evolution compared to machines.

The current approaches for control are either classic control approach or optimization based approach. These approaches do not take intelligent decisions but obey certain rules that was depicted to the controller by a control expert. The conventional approaches require extensive knowledge from an expert with relevant domain knowledge to transfer the knowledge to the controller via control law and other mathematical derivation. On the other hand, optimization based controllers like Model Predictive controllers look ahead in the future and take action considering future errors but suffer from the fact that the optimization step takes time to return optimal control input, especially for complex high dimensional systems. The detailed drawback of the current approaches for process control is discussed in the limitation section below.

There are many limitations with the current approaches to industrial process control. The classical controller requires very careful analysis of the process dynamics. It requires a model of the process either derived from first principles or empirical. If the system under consideration is complex then the first principle model requires advanced domain knowledge to build the model. In certain cases, the conventional approaches involves derivation of control law that meets desired criteria (either to control the plant to track the set point or operate in a way to be economical). It involves tuning of controllers with respect to the system to be controlled. Model maintenance is very difficult or rarely achieved. On the other hand, optimization based controllers look ahead into the future and take future errors into account. However, they suffer from the fact that computing optimal control actions via optimization tends to take time especially for non-linear high dimensional complex system. They cannot handle uncertainty about the true system dynamics and noise in observation data. One more burden of optimization based controller like MPC is the prediction of hidden states, since often those states are not readily available. MPC also demands to tune prediction horizon, control horizon, weights given to each output etc. As a prerequisite, MPC requires an accurate model of the system to start with. Also over the course of time the controller will not possess an updated model, hence the performance will start to deteriorate.

Having given the above limitations can we build a learning controller that does not require 1) intensive tuning 2)

[⋆] We would like to thank Mitacs Globalink for their financial support

plant dynamics 3)model maintenance, and 4) takes instant control actions? This work aims to develop such next generation intelligent controllers.

It could be challenging to learn the entire trajectory of a plant's output along with the set point. Deploying these learning controllers in production could be risky, hence it should be made sure that the controller has been already trained offline before implementing it in the real plant. The action space and state space in process control application are continuous, hence function approximators are used with reinforcement learning to learn the continuous state space and action space. Since it has been found that deep neural networks serve as an effective function approximators and recently have found great success in image [1], speech [2] and language understanding [3], deep neural networks are used as common function approximators with reinforcement learning. However, to speed up the training of deep neural networks, it requires training them in the Graphical Processing Unit (GPU) to parallelize the computation. This could be tricky to train as care should be taken to optimize the memory allotted to perform computations in the GPU. The reinforcement learning based controller involves training two neural networks simultaneously, tuning those two neural networks for the first time to train the plant was challenging. The reason is the enormous number of options and space available as tunable parameters (hyper parameters). In addition, running simulation takes a lot of time, hence re tuning the hyper parameters to retrain the network was bit challenging.

In recent years, there has been significant progress in the fields of computer vision and natural language processing that followed the success of deep learning. The deep learning era started in 2012 after it won the image classification challenge in 2012, by a huge margin compared to other approaches [1]. In Machine translation conventional probabilistic approaches were replaced with deep neural networks [4]. From then on, self driving cars started to take the advantage of deep nets [5]. They use deep learning to learn the optimal control behaviour required to drive the car without the help of human. This is a form of a learning control problem. Human level control has also been attained in games [6] and physical tasks [7]. Computers were able to beat human beings in classical Atari games [6]. Deep neural nets have even learnt the complex game of Go [8]. This is again posed as a learning control problem. The Go game has 1.74×10^{172} states, which is more than the number of atoms in the universe. The question is can we gain advantage of those powerful models to build a learning controller to overcome the limitations of the current control approach? Our work aims to answer this question. To support using such powerful models, the number of calculations per second has been growing exponentially over the years.

2. BACKGROUND

2.1 The Reinforcement Learning Problem

Reinforcement learning is a sequential decision making problem in which an agent takes an action at time step t and learns as it performs that action at each and every time step, with the goal of maximizing the cumulative reward. The decision maker is the agent and the place the agent interacts or acts is the environment. The agent receives observations x_t (or states) from the environment and takes an action a_t. Once the action is performed, the environment then provides a reward signal r_t (scalar) that indicates how well the agent has performed. In all the environment (in process control) considered here the actions are real valued $a_t \in \mathbb{R}^N$. Usually, the environment may be partially observed so that the entire history of observation and action pairs $s_t = (x_1, a_1, ..., a_{t-1}, x_t)$. We assumed the environment is fully-observed so $s_t = x_t$.

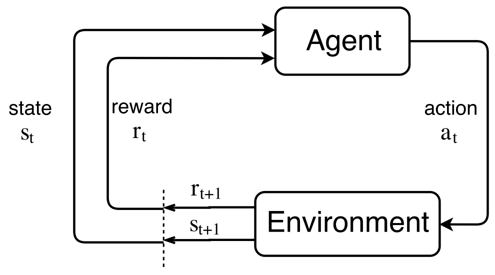


Fig. 1. Reinforcement Learning block diagram

2.1.0.1. Policy Policy fully defines the behaviour of an agent. An agent takes action using policy, π, which is a distribution over action given states. $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. The environment, E, may also be stochastic. The reinforcement learning environment is modelled as a Markov Decision Process (MDP) with a state space $\mathcal{S}$, action space $\mathcal{A} = \mathbb{R}^N$, an initial state distribution $p(s_1)$. transition dynamics $p(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$.

2.1.0.2. Return The return from a state is defined as the sum of discounted future reward,

$$R_t = r(s_1, a_1) + \gamma r(s_2, a_2) + \gamma^2 r(s_3, a_3).. = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \tag{1}$$

where $\gamma \in [0, 1]$ is a discounting factor. If γ is smaller only rewards in immediate present matters and vice versa. It also makes the summation to be bounded and finite. The return depends on action which depends on the policy, π and may be stochastic.

2.1.0.3. Value function The value function $V^\pi(s)$ gives a long-term value of states s. The state value function is the expected return starting from state s,

$$V^\pi(s) = \mathbb{E}[R_t|S_t = s] \tag{2}$$

The goal in reinforcement learning is to learn a policy, π that maximizes the expected return from the start distribution, $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_t]$

2.1.0.4. Action value function In value function only states are considered into account, hence an other value function which is important is action value function. The action value function is the expected return starting from state s, taking action a, and then following policy π is given by,

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | S_t = s, A_t = a] \tag{3}$$

More specifically, the action value function is given by, $Q^{\pi}(s_t,a_t) = \mathbb{E}_{r_i \geq t, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t | s_t, a_t]$

2.1.0.5. Bellman Equation Many approaches in reinforcement learning make use of recursive relationship derived from value function,

$$\begin{aligned} V(s) &= \mathbb{E}[R_t | S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3}..| S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma R_{t+1} | S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | S_t = s] \end{aligned} \tag{4}$$

The Bellman equation for action value function is given as,

$$Q^{\pi}(s_t,a_t) = \mathbb{E}_{r_i, s_{t+1} \sim E}[r(s_t,a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{5}$$

If the target policy we are considering is deterministic, we can describe it as a function $\mu : \mathcal{S} \rightarrow \mathcal{A}$ and eliminate the inner expectation as follows:

$$Q^{\mu}(s_t,a_t) = \mathbb{E}_{r_i, s_{t+1} \sim E}[r(s_t,a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \tag{6}$$

If you see in the above equation π is replaced with deterministic function μ.

2.1.0.6. Q-learning The expectation in equation (6) depends on environment. Hence, it is possible to learn Q^{μ} off-policy (when the current transition does not depend on the current policy) using transitions which are generated from a different stochastic behaviour policy β.

In general in reinforcement learning, two steps are involved to learn an agent,

- Policy evaluation i.e., Estimating the value function
- Policy improvement

Q-learning [9] uses the greedy policy
$\mu(s) = \arg\ \max_a Q(s,a)$ for policy improvement. In Q-learning the action-value function is estimated as given in equation (7). Since the number of states and action pairs are high, the action value of those many states cannot be stored in table, hence function approximators parameterized by θ^Q, are used to learn action value function. θ^Q is obtained by minimizing the loss function:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t | \theta^Q) - y_t)^2] \tag{7}$$

where,

$$y_t = r(s_t,a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1} | \theta^Q)) \tag{8}$$

The discounted state visitation distribution for the policy β is ρ^{β}

2.2 Actor Critic Approach

The process control application we are considering in this work has continuous states and actions. Q-learning cannot be directly applied to continuous action spaces, because the greedy policy in Q-learning requires computing *max* over actions. Hence finding the greedy policy requires an optimization of a_t at every timestep; this optimization step is slow with unconstrained, large function approximators with large number of action variables. Hence, actor-critic approach is used to tackle this challenge.

In actor-critic, the critic learns the action value function and the actor learns the policy. The actor-critic framework maintains two set of parameters,

- *Critic* Updates action-value function parameters W_c
- *Actor* Updates policy parameters W_a, in the direction suggested by critic

In deterministic policy gradient, the policy is represented by a function approximator with weights W_a given by $a = \pi(s, W_a)$. In a big picture, the objective function is defined as total discounted reward as follows:

$$J(W_a) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + ..] \tag{9}$$

The goal of learning is to adjust parameters W_a and W_c to achieve more reward.

Critic estimates the value of current policy by Q-learning. The parameter W_c of the critic is updated using the gradient derived from Q-learning as follows:

$$\frac{\partial L(W_c)}{\partial W_c} = \mathbb{E}\big[(r + \gamma Q(s', \pi(s'), W_c) - Q(s,a,W_c)) \frac{\partial Q(s,a,W_c)}{\partial W_c}\big] \tag{10}$$

The actor updates policy in direction that improves Q, hence the gradient with which the actor is updated is given as,

$$\begin{aligned} \frac{\partial J(W_a)}{\partial W_a} &= \mathbb{E}\big[\frac{\partial Q(s,a)}{\partial W_a}\big] \\ &= \mathbb{E}\big[\frac{\partial Q(s,a,W_c)}{\partial a} \frac{\partial \pi(s,W_a)}{\partial W_a}\big] \end{aligned} \tag{11}$$

The policy gradient is the direction that improves Q.

The gradients given in equation 10 and 11 are used with gradient descent based optimization technique to update parameters W_c and W_a. In equation 11, the gradient propogate backwards from critic to actor.

Since it has been found that deep neural networks serve as effective functional approximators and have found great success in image [1], speech [2] and language understanding [3], deep neural networks are used to approximate value function or policy resulting in Deep Reinforcement Learning (DeepRL).

Introducing non-linear function approximators in Q-learning does not gurantee convergence. Recent success of Deep Q nework [6],[10] along with improvements in the field of deep learning like Batch Normalization [11] have addressed the issue of using neural networks in reinforcement learning. With those advancements, this work aims at using DeepRL for process control applications.

2.3 Neural Network

2.3.0.7. Neural Network regression The neural networks can be expressed as set of operations as shown in Figure 2:

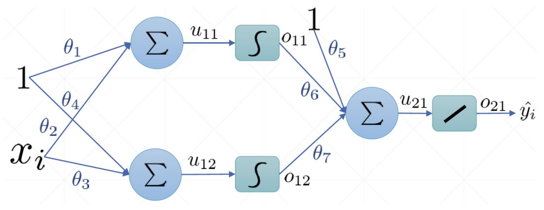


Fig. 2. A Neural networks on 1 dimensional input data set.

From Figure 2, first the linear regression model which is a weighted combination of inputs and weights are used. Followed by a sigmoid activation function ($\sigma(x) = \frac{1}{1+e^{-x}}$), which says if the neuron has to be activated or not (1 if activated). Other common activation function used with neural network are, Sigmoid, $\sigma(x) = 1/(1+e^{-x})$, ReLU, $\sigma(x) = max\{0, x\}$, and tanh $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

The θ_1 and θ_2 are weights or parameters here. Ultimately, the final layer has linear activation because we are interested in using neural networks for regression.

In Fig. 2, the output $\hat{y}_i$ is given by,

$$\hat{y}_i = \theta_5 + \theta_6\sigma(\theta_1 + \theta_2 x_i) + \theta_7\sigma(\theta_4 + \theta_3 x_i) \tag{12}$$

Where,

- $\theta's$ are the parameters.
- x_i is the input of the i^{th} sample
- Activation function (sigma represented in green circular rectangle) is the sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$)
- u_{11}, u_{12} are the outputs of the neurons before activation
- o_{11}, o_{12} are the outputs of the neurons after activation in first layer
- o_{21} is the output of the neuron after activation in the second layer
- Since linear activation is used in final layer, $\hat{y}_i = o_{21} = u_{21}$

Now our goal is to learn the parameters θ of the neural network. If g is the loss function, the parameters are computed using the gradient descent update given as,

$$\theta^{t+1} = \theta^t - \alpha \times \frac{\partial g(\theta)}{\partial \theta} \tag{13}$$

The gradient of loss, g with respect to the parameter is given as,

$$\frac{\partial g(\theta)}{\partial \theta_j} = -2(y_i - \hat{y}_i(x_i, \theta))\frac{\partial \hat{y}_i(x_i, \theta)}{\partial \theta_j} \tag{14}$$

The value of $\hat{y}_i(x_i, \theta)$ can be computed while doing a forward pass (discussed below) and the derivative of $\hat{y}_i$ with respect to the parameter is computed during the back propagation step discussed below.

2.3.0.8. Forward Pass In forward pass, each element of θ has a known value. Then from Fig. 2, the values of u, o and $\hat{y}_i$ are given as,

- $u_{11} = \theta_1 \times 1 + \theta_2 \times x_i$
- $u_{12} = \theta_4 \times 1 + \theta_3 \times x_i$
- $o_{11} = \frac{1}{1+e^{-u_{11}}}$
- $o_{12} = \frac{1}{1+e^{-u_{12}}}$
- $u_{21} = \theta_5 \times 1 + \theta_6 \times o_{11} + \theta_7 \times o_{12}$
- $\hat{y}_i = o_{21} = u_{21}$

2.3.0.9. Backpropagation The next step is to compute the partial derivatives of $\hat{y}_i$ with respect to the parameter θ. Each $\hat{y}_i$ can be represented as a function of previous layer's weights as,

$$\hat{y}_i = \theta_5 \times 1 + \theta_6 \times o_{11} + \theta_7 \times o_{12} \tag{15}$$

From the above equation the gradient of $\hat{y}_i$ with respect to $\theta_5, \theta_6, \theta_7$ is given as,

- $\frac{\partial \hat{y}_i}{\partial \theta_5} = 1$
- $\frac{\partial \hat{y}_i}{\partial \theta_6} = o_{11}$
- $\frac{\partial \hat{y}_i}{\partial \theta_7} = o_{12}$

Similarly, $u_{12} = \theta_2 \times x_i + \theta_3 \times 1$, now the derivative of $\hat{y}_i$ with respect to θ_3 is given in equation (16). It is essential to note that θ_3 only influences $\hat{y}_i$ through u_{12}.

$$\frac{\partial \hat{y}_i}{\partial \theta_3} = \frac{\partial \hat{y}_i}{\partial o_{12}} \frac{\partial o_{12}}{\partial u_{12}} \frac{\partial u_{12}}{\partial \theta_3} \tag{16}$$

The $o's$ in the equation 16 are received during the forward pass. Similarly the gradient with respect to $\theta_1, \theta_2, \theta_4$ are computed.

Once the $\hat{y}_i$ and the derivatives of $\hat{y}_i$ with respect to the θ are computed, the gradient descent as given in 13 is used to update the parameters θ and the forward pass and update step continues until the optimum value of parameter (or minimal loss) is attained.

3. POLICY REPRESENTATION FOR PROCESS CONTROL PROBLEMS

This section connects the terminologies in reinforcement learning with process control. A policy is RL agent's behaviour. It is a mapping from states S to Actions A, i.e., $\pi(s) : S \mapsto A$. We have considered a deterministic policy with both states and actions in a continuous space. This section discusses how process control can be posed as a Reinforcement Learning problem.

3.1 States

A state s consists of features describing the current state of the plant. Since the controller needs both the current output of the plant and the required setpoint to take necessary action to get closer to the setpoint, the state must contain information about current plant output and the required setpoint. For instance it can be a tuple of current output and deviation from setpoint $< y, (y - y_{setpoint}) >$ or just current output and setpoint $< y, y_{setpoint} >$.

3.2 Actions

The action, a is the means through which RL agent interacts with environment. The controller input to the plant is the action in process control.

3.3 Reward

The reward signal r is a scalar feedback signal that indicates how well an RL agent is doing at step t. It reflects the desirablity of a particular state transition that is observed by performing action a starting in the initial state s and resulting in a successor state s'. Fig: 3 shows state-action transition and corresponding reward of the controller in process control. For process control task, the objective is to take the output closer to the set point, while meeting certain constraints. This specific objective can be fed to RL agent (controller) by means of a reward function. Thus, the reward function serves as a function similar to an objective function formulation in Model Predictive Control. We have formulated two reward hypothesis in this work. The first one is the linear reward hypothesis where the reward is a linear function with respect to deviation in setpoint. It is given in equation (17),

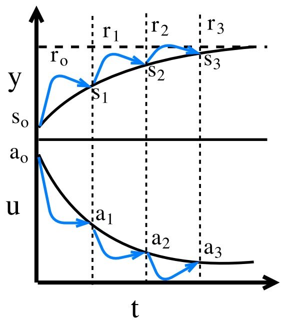


Fig. 3. Transition of states and actions

$$r(s,a,s') = \begin{cases} c, & \text{if } |y^i - y^i_{set}| \leq \varepsilon, \forall i \\ -\sum_i^n |y^i - y^i_{set}|, & \text{otherwise} \end{cases} \tag{17}$$

where i represents i^{th} output of n outputs in MIMO systems and $c > 0$, is a constant. A high value of c leads to larger value of r when the ouputs are closer to setpoint by ε. A high value of c also results in quicker tracking of setpoint. In MIMO system different weightage can be given to individual output variables by modifying equation (17).

The second formulated reward hypothesis which we call is the discrete polar reward hypothesis. It is given in equation (18). It gives a zero reward if all the outputs approaches toward the setpoint making the controller take actions at each time step to approach towards the setpoint.

$$r(s,a,s') = \begin{cases} 0, & \text{if } |y^i_t - y^i_{set}| > |y^i_{t+1} - y^i_{set}|, \forall i \\ -1, & \text{otherwise} \end{cases} \tag{18}$$

3.4 Policy and Value Function Representation

The policy π is represented by an actor using deep feed forward neural network parameterized by weights W_a. Thus, an actor is represented as $\pi(s, W_a)$. This network is queried at each time step to take an action given the current state. The action value function is represented by a critic using another deep neural network parameterized by weights W_c. Thus, critic is represented as $Q(s, a, W_c)$. The actor and critic neural network representation is shown in figure 4. The critic network predicts the Q-values for each states and action and actor network proposes an action for the given state. The goal is to learn the actor and critic neural network parameters by interacting with the plant. Once the parameters are learnt, the learnt deterministic policy is used to compute action at each step given the current state, s which is a function of current output and setpoint of the process. The next section discusses how to learn actor and critic neural network just by interacting with the plant.

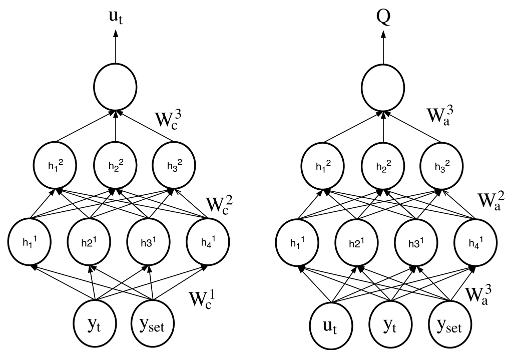


Fig. 4. Representation of actor and critic neural network

4. LEARNING CONTROL POLICIES

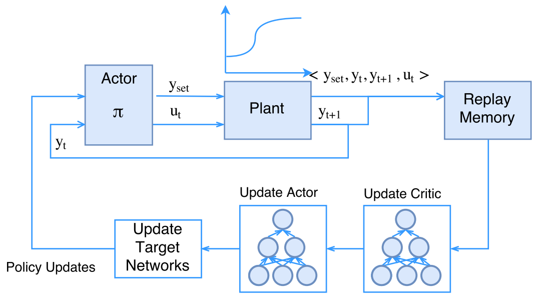


Fig. 5. Learning Overview

The overview of learning the actor and critic is shown in Fig 5. The corresponding algorithm for learning is discussed in Algorithm 1. The learning algorithm is inspired by [7] with modifications to account for set point tracking and other recent advancements discussed in [12],[11].

We ran our learning algorithm interms of episodes. An episode is terminated when it runs for 200 time steps or if it has tracked setpoint by a factor of ε continuously for five time steps. For systems with higher value of time constant, τ larger time steps per episode is preferred.

Initially the actor and critic neural networks are initialized with parameters W_a and W_c. The weights are randomly initialized. At each time step the actor, $\pi(s, W_a)$ is queried and the tuple $< s_i, s'_i, a_i, r_i >$ is stored in the replay memory, RM at each iteration. The replay memory RM is a queue data structure that stores state, action and reward transitions. The data are sampled from this replay memory for updating the weights of W_a and W_c. With replay memory batch gradient descent can be performed other than stochastic gradient descent. The replay memory also breaks the correlation between samples. The learning algorithm uses actor-critic framework. The critic network serves as a value estimator and it estimates the state action value of current policy by Q-learning. The critic provides loss function for learning the actor. The loss function of the critic is given by, $L(W_c) = \mathbb{E}\big[(r + \gamma Q(s', \pi(s'|W_c)) - Q(s, a, W_c))^2\big]$. The loss function is same as the loss function of Q-learning. Hence, the critic network is updated using this loss with the gradient given by,

Algorithm 1 Learning Algorithm

```
1: $W_a, W_c \leftarrow$ initialize random weights
2: initialize Replay memory, RM with random policies
3: for episode = 1 to E do
4:     Reset the OU process noise $\mathcal{N}$
5:     Specify setpoint, $y_{set}$ at random
6:     for step = 1 to T do
7:         $s \leftarrow < y_t, y_{set} >$
8:         $a \leftarrow$ action, $u_t = \pi(s, W_a) + \mathcal{N}_t$
9:         Execute action $u_t$ on the plant
10:        $s' \leftarrow < y_{t+1}, y_{set} >$, observe state at next instant
11:        $r \leftarrow$ reward
12:        Store the tuple $< s, a, s', r >$ in RM
13:        Sample a minibatch of n tuples from RM
14:        Compute $y^i = r^i + \gamma Q^i_t \ \forall i \in$ minibatch
15:        Update Critic:
16:        $W_c \leftarrow W_c + \alpha(\frac{1}{n}\sum_i^n(y^i - Q(s^i, a^i, W_c)\frac{\partial Q(s^i,a^i,W_c)}{\partial W_c})$
17:        Compute $\nabla^i_p = \frac{\partial Q(s^i,a^i,W_c)}{\partial a^i}$
18:        Clip gradients $\nabla^i_p$ by (21)
19:        Update Actor:
20:        $W_a \leftarrow W_a + \alpha\frac{1}{n}\sum_1^n(\nabla^i_p\frac{\partial \pi(s^i,W_a)}{\partial W_a})$
21:        Update Target Critic:
22:        $W^t_a \leftarrow \tau W_a + (1 - \tau)W^t_a$
23:        Update Target Actor:
24:        $W^t_c \leftarrow \tau W_c + (1 - \tau)W^t_c$
25:    end for
26: end for
```

$$\frac{\partial L(W_c)}{\partial W_c} = \mathbb{E}\big[(r + \gamma Q_t - Q(s, a, W_c))\frac{\partial Q(s, a, W_c)}{\partial W_c}\big] \quad (19)$$

Instead of using $r + \gamma Q(s', \pi(s'|W_c)$ as given in Q-learning . The gradient of critic uses $r + Q_t$. Where $Q_t = Q(s', \pi(s', W^t_a), W^t_c)$, denotes target critic neural network and W^t_a, W^t_c are weights of target actor and critic respectively. The target networks are used to improve convergence of the learning algorithm [10]. The target actor and target critic network are updated in line 22, and 24 of Algorithm 1. We choose, $\tau = 0.001$, hence the target actor and critic are 1000 time steps older than current actor and critic neural network.

We chose $\gamma = 0.99$ in our simulations. Thus the learnt value function provides a loss function to actor and the actor updates its policy in a direction that improves Q. Thus, the actor network is updated using gradient given by,

$$\frac{\partial J(W_a)}{\partial W_a} = \mathbb{E}\big[\frac{\partial Q(s, a, W_c)}{\partial a}\frac{\partial \pi(s, W_a)}{\partial W_a}\big] \quad (20)$$

Table 1.

Hyperparameter	**Value**	**Note**
Minibatch size n	32	
Policy learning rate	10^{-4}	Step size for ADAM
Critic learning rate	10^{-4}	Step size for ADAM
Policy weight decay	0	
Critic weight decay	10^{-2}	L2 cost for each parameter
Target update rate τ	10^{-3}	
Replay memory size RM	10^5	older transitions are discarded
Exploration noise	0.2	
Reward discount γ	0.99	
Warm-up time	50,000	Timesteps until training starts

where $\frac{\partial J(W_a)}{\partial W_a}$ is the gradient of critic with respect to the sampled actions of mini-batches from RM. Thus, the critic and actor are updated in each iteration, resulting in policy improvement. The expectation in equation (19) and (20) is over the mini-batches sampled from RM.

In order to facilitate exploration, we added a noise sampled from Ornstein-Uhlenbeck (OU) process [13] discussed similarly in [7]. Apart from exploration noise, the random initialization of system output at each start of an episode ensures that the policy is not stuck in a local optimum.

Clipping Gradients: Sometimes the actor neural network, $\pi(s, W_a)$ produces output that exceeds the action bounds of the specific system. Hence, the bound on the output layer of the actor neural network is specified by controlling gradient required for actor from critic. To avoid making the actor returning outputs that are outside action bounds, we clipped the gradients of $\frac{\partial Q(s,a,W_c)}{\partial a}$ given in [12] using the following transformation,

$$\nabla_p =$$

$$\nabla_p . \begin{cases} (p_{max} - p)/(p_{max} - p_{min}), & \text{if } \nabla_p \text{suggests increasing p} \\ (p - p_{min})/(p_{max} - p_{min}), & \text{otherwise} \end{cases} \quad (21)$$

Where ∇_p refers to parameterized gradient of critic. p_{max}, p_{min} refers to maximum and minimum action of the agent. p correspond to entries of the parameterized gradient, $\frac{\partial Q(s,a,W_c)}{\partial a}$. This also serves as an input constraint to the RL agent.

5. IMPLEMENTATION AND STRUCTURE

The learning algorithm was written in python and implemented on a Ubuntu linux distribution machine. It was trained on Amazon g2.2xlarge EC2 instances. The deep neural networks was built using Tensorflow [14]. The computation of high dimensional matrix multiplication was made parallel with the help of Graphics Processing Unit (GPU) available in the EC2 instances.

The details of our actor and critic neural networks are as follows: We used Adam [15] for learning actor and critic neural network. We used Rectified non-linearity as an activation function for all hidden layers in actor and critic. The actor and critic neural networks had 2 hidden layers with 400 and 300 units respectively. The layers are batch normalized [11]. The hyperparameters are discussed in table 1

6. SIMULATION RESULTS

6.1 *Case 1: SISO system*

A 1×1 process is used to study the effect of this approach of learning the policy. The industrial system we choose to study is the control of a paper-making machine. The target output, y_{set}, is the desired moisture content of the paper sheet. The control action, u, is the steam flow rate, and the system output, y, is the current moisture content. The differential equation of the system is given by,

$$y(t) - 0.6\dot{y}(t) = 0.05\dot{u}(t), \tag{22}$$

where $\dot{y}(t)$ and $\dot{u}(t)$ are output and input derivatives with respect to time. The corresponding transfer function of the system is,

$$G(s) = \frac{0.05s}{1 - 0.6s} \tag{23}$$

The time step used for simulation is 1 second.
The system is learned using Algorithm 1 with the reward hypothesis given in equation (17) . The portion of the learning curve of the SISO system given in (22) is shown in Fig 6 and the corresponding learned policy is shown in Fig 7. and Fig. 8

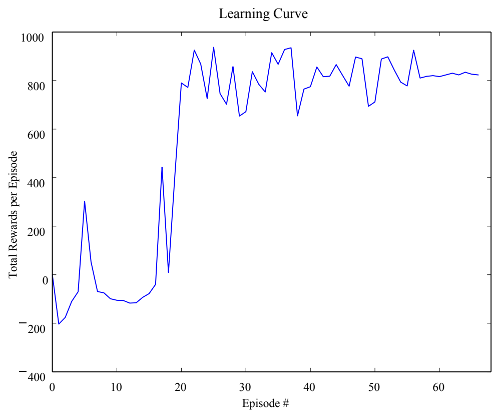


Fig. 6. Case 1: learning curve of SISO system

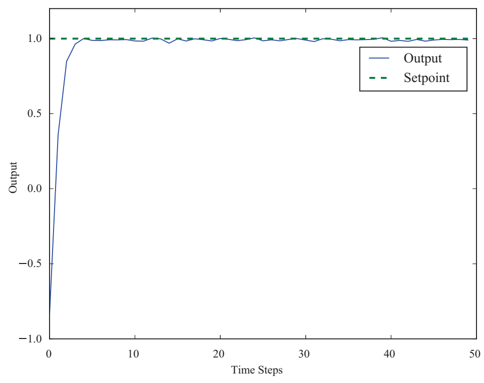


Fig. 7. Case 1: output response of the learned policy

6.2 *Case 2: MIMO system*

Our approach is also tested on a high purity distillation column Multi Input Multi Output (MIMO) system described in [16]. The first output, y_1 is the distillate composition, the second output, y_2 is the bottom composition, the first control input u_1 is the boil-up rate and the second control input u_2 is the reflux rate. The differential equations of the MIMO is,

$$\begin{aligned} \tau\dot{y_1}(t) + y_1(t) &= 0.878u_1(t) - 0.864u_2(t) \\ \tau\dot{y_2}(t) + y_2(t) &= 1.0819u_1(t) - 1.0958u_2(t) \end{aligned} \tag{24}$$

where $\dot{y_1}(t)$ and $\dot{y_1}(t)$ are derivatives with respect to time. The system is trained using the learning algorithm with the reward hypothesis given in equation (17). The output and input response of the learned policy of the system is shown in Fig 10.

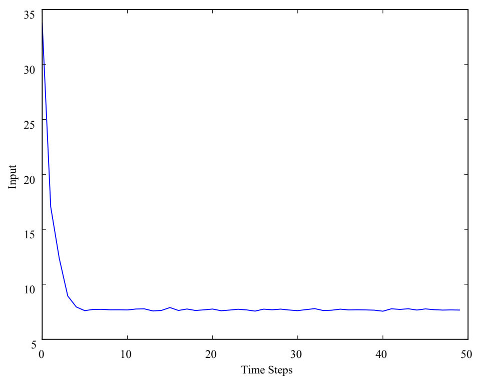


Fig. 8. Case 1: input profile of the learned policy

6.3 *Case 3: Non-linear MIMO system*

We tested our approach on a non-linear Multiple Input Multiple Output HVAC heating coil system given in [17]. The HVAC system has 3 exteranl process variables that the controller does not have influence on,

- T_{ai}, the inlet air temperature,
- f_a, inlet cold air flow rate,
- T_{wi}, the inlet hot water temperature.

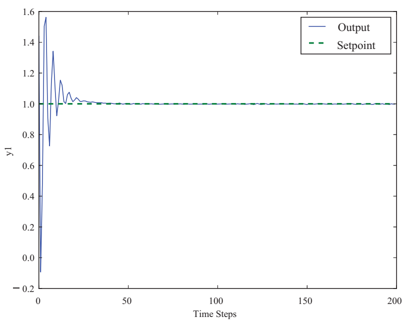
In HVAC, the hot water flows through a tube and heats up the entering air. The goal is to control the hot flow rate valve, i.e., input u to control the output air temperature T_{ao} to some desired output. The other process variables are,

- output air temperature, T_{ao},
- F_w output hot water flow rate,
- Hot water output temperature T_{wo}.

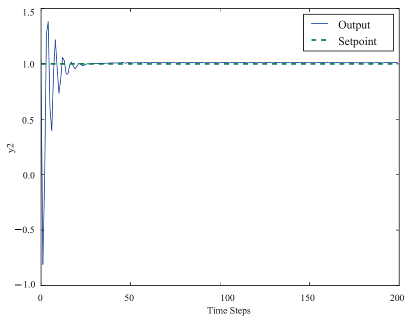
The model of the HVAC is given below:

$$\begin{aligned} f_w(t) = 0.008 + 0.00703 \times (&-41.29 + 0.30932 \times u_t \\ &-0.3681 \times 10^{-4} \times u_t^2 + 9.56 \times 10^{-8} u_t^3) \end{aligned} \tag{25}$$

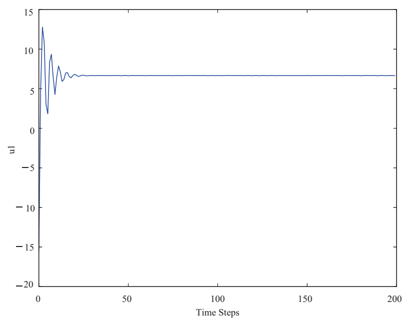
$$\begin{aligned} T_{wo}(t) = T_{wo}(t-1) + 0.64908 \times f_w(t-1) \times (T_wi(t-1) \\ -T_{wo}(t-1) + (0.02319 + 0.10357 \times f_w(t-1) \\ +0.02806 \times f_a(t-1)) \times \\ (T_{ai}(t-1) - \frac{T_{wi}(t-1) + T_{wo}(t-1)}{2}) \end{aligned} \tag{26}$$



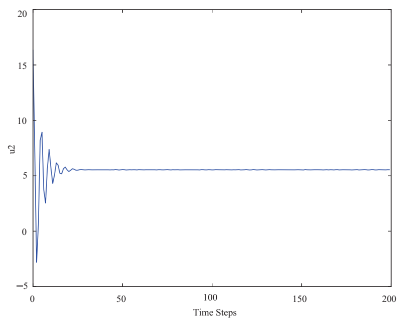
(a) Response of output y_1 with output noise of $\sigma^2 = 0.01$



(b) Response of output y_2 with output noise of $\sigma^2 = 0.01$



(c) Response of input u_1



(d) Response of input u_2

Fig. 10. case 2: output and input response of MIMO system system

$$
\begin{aligned}
T_{ao}(t) = T_{ao}(t-1) + 0.19739 \times f_a(t-1)(T_{ai}(t-1) \\
-T_{ao}(t-1)) + (0.03184 + 0.15440 \times f_w(t-1) \\
+0.04468 \times f_a(t-1))(\frac{T_{wi}(t-1) + T_{w0}(t-1)}{2} \\
-T_{ai}(t-1)) + 0.20569 \times (T_{ai}(t) - T_{ai}(t-1))
\end{aligned} \tag{27}
$$

The external disturbance variables f_a, T_{wi} and T_{ai} are modified by random walk on a per time step basis to model changing set points and various disturbances from the environment that would occur in actual HVAC systems. The ranges for the random walk are $0.6 \leq f_a \leq 0.9$ [kg/s], $73 \leq T_{wi} \leq 81$ [deg C], and $4 T_{ai} 10$ [deg C].

This system is complex to control compared to the SISO and MIMO system discussed in case 1 and case 2, because the system is oscillatory, non-linear, has 3 disturbance variables leading to a noisy system. The linear reward hypothesis (17) on this system resulted in an offset and is not able to learn the system. This lead to the formulation of discrete polar reward hypothesis (18). Hence, this system is trained with discrete polar reward hypothesis. With discrete polar reward hypothesis, we found the response to be less noisy, smooth and eliminated offset. In general we found discrete polar hypothesis (18) to have better performance (in terms of noise, offset) compared to linear reward hypothesis (17).

Our approach was tested on this non-linear system on a step-setpoint change condition, the output and input response of the learnt policy is shown in Fig. 11 and Fig. 12.

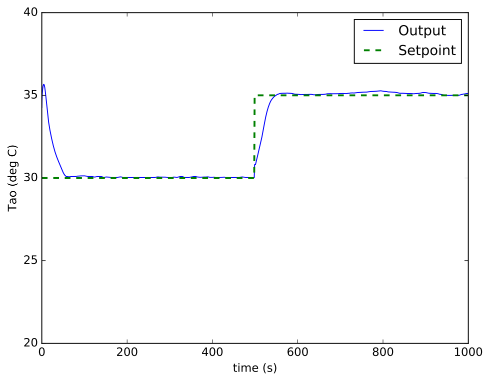


Fig. 11. Case 3: output response of the learned policy

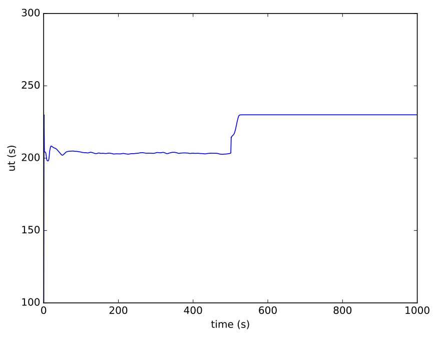


Fig. 12. Case 3: input profile of the learned policy

6.4 Case 4: Controller susceptibility to plant change

We tested our approach to see if it is adaptible to change in plant model. We ran our learning algorithm on the system given in equation 22 for setpoint 2 with linear reward hypothesis. The controller has learnt the control policies to take the output closer to setpoint at around 2700 seconds. Then we changed the plant model to see if the controller can learn the change in plant dynamics. We increased the gain by two times. Now the controller again started to learn and it took the output closer to the setpoint after around 1500 seconds once the model was changed. This case study demonstrates how the learning controller actually learns the plant and then control it. The output and input response is shown in Fig. 13 and Fig. 14.

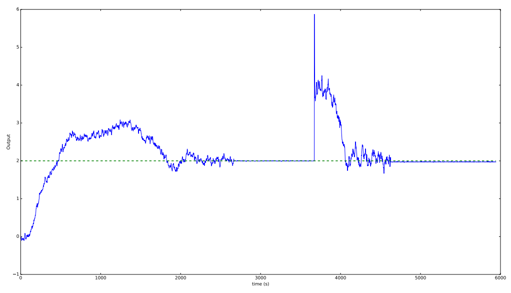


Fig. 13. Case 4: output response of the learned policy

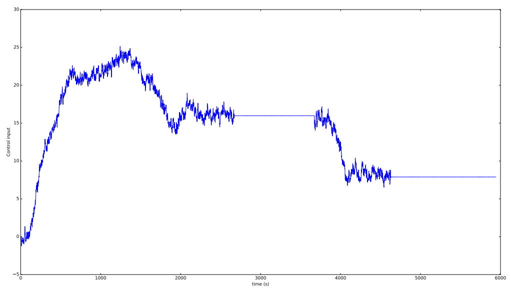


Fig. 14. Case 4: input profile of the learned policy

7. DRL CONTROLLER ANALOGY WITH MODEL PREDICTIVE CONTROL

7.1 Model

In Model Predictive Control explicit first principle or empirical model is necessary for the prediction step as well as the optimizaiton step. Also the performance of the controller depends on how accurate the chosen model is. On the other hand, in deep reinforcement learning, the model is learnt just by interaction with the system, and at anytime if the plant model gets changed the model gets updated.

7.2 Constraints

In MPC, the constraints such as constraints on input, rate of change of inputs and economic constraints are fed explicitly as a constraint in the optimization objective function. The optimization algorithm then operate with the given constraint to find the minimum. Whereas in reinforcement learning, the input and economic constraint can be given in reward hypothesis. The learning algorithm then learns policy that maximizes the expected future reward, which makes the algorithm meet the constraints.

The bounds on input in RL controller is controlled by the gradient clipping on critic neural network with respect to action given in equation 21

7.3 Future prediction with Model

The MPC uses model to predict the future time steps given by prediction horizon and minimizes sum of future error before taking an action. On the other hand, RL controller works by maximizing expected future reward. In this way, RL controllers accounts for future control error.

The γ controls how far to look ahead in the future. A value of $\gamma = 1$ considers all rewards from future states whereas a value of 0 considers reward from current state. The γ discounts rewards in future states by value of γ. This is a tunable parameter that makes our learning controller to check how far to look in the future.

7.4 Hidden states estimation

The prediction step in MPC relies on estimating the hidden state and then the states are rolled out with time for future prediction. This can be burdensome for complex systems. On the other hand, the RL controller learns controller policies directly from output and does not rely on hidden states. The hidden states information are stored as an abstract within the neural network.

7.5 Optimization

The MPC works by minizing the squared difference of predicted output and the setpoint and returning the control action at each time steps. Whereas the RL controller aims at finding a policy that maximizes the expected future reward. The optimization step is done at each time step using a gradient based algorithm to learn the policy. The policy gets better and better with respect to the experience.

8. TUNING OF DRL CONTROLLER

8.1 Choosing number of episodes

We considered 1 episode to be of 200 time steps (or) when the controller has tracked setpoint continuously for 5 time steps. For systems with high time constants, τ larger number of times steps per episode is preferred, because some system won't be able to attain steady state within 200 time steps.

8.2 Formulating reward hypothesis

Equation 17 and 18shows reward formulation for a controller to track setpoint. This formulation uses an absolute control error to generate reward. Other reward formulations that can be considered are,

- negative of 2 norm of control error
- In case of MIMO system, different weights can be given to different output so as to give weightage to each output to be tracked
- Economic constraint can be added to the reward hypothesis

8.3 Choosing bounds for gradient clipping

There should be a bound on the output of the controller to make sure the control actions are within the desirable operation range. They are attained by adjusting the gradient of the critic with respect to actions while updating the actor network as given in equation (21). Having this bound will also speed up the convergence of learning. Hence, lower the range of the bounds, the greater is the convergence rate.

8.4 Choosing states for actor and critic neural networks

The states (or) the observations depict the current state of the plant. This is an up-to-date information given to the controller to operate to take necessary action. The states we considered are a tuple of output and setpoint. If you have extra sensor information about the system those information will also be helpful for the RL algorithm to learn the policy.

8.5 Choosing Replay Memory, RM size

The action, observation and reward samples are collected at each time step and stored in a replay memory, RM. During learning, the samples from replay memory are randomly sampled. The greater the Replay memory size, the more the RL algorithm will have access to data of old policies. Also, once the policies are learnt and the system attains steady state the action and observation remains same hence these repetitive data are not sent to replay memory, as these are not useful information to be kept in replay memory.

8.6 Controlling the learning switch

Once the system tracks the setpoint and has reached a steady state, the learning of actor and critic neural network can be stopped. In addition to that once the learning is stopped, the action, observation and reward samples are not added to replay memory. This can be attained by comparing the running mean of the output and the setpoint, if the difference is less than a Negligible amount i.e., a small factor, the learning can be stopped.

8.7 Handling actor exploration noise

Once the learning of actor and critic is stopped, the OU noise added to action for expolration purpose can also be stopped, because this noise adds extra noise to control actions as well as the system output.

9. PRACTICAL CONSIDERATION

9.1 Starting the DRL controller

Since, DRL controller learns polices just by interaction with the plant, care should be taken to make sure the operating and safety margin of the system are met. Also, learning control policies by interaction leads to loss of resources during training as the controller does not act well during training. Hence, it is encouraged to train the RL algorithm on a model first before implementing in the real plant.

9.2 Operating the controller with bounds on control actions

To maintain the operating and safety margin, bounds should be added to control actions that is fed to the plant. Eventhough, the bounds are taken care during the learning as given in equation 21, sometimes the neural network output violates the bounds during the initial phase of training. These situation can be handled by hard-coding and clipping the control action within the lower and upper bound specified.

10. CONCLUSION

We have developed an artificial intelligence approach to process control with the recent advancements in the field of reinforcement learning and deep leartining. We have demonstrated our approach on a SISO, MIMO and a non-linear system with external disturbances. We believe process control will benefit from these reinforcement learning controllers.

11. ACKNOWLEDGEMENT

We would like to thank Mitacs for financial support through the Globalink Graduate fellowhip, and Smriti Shyamal of McMaster University for helpful discussions

REFERENCES

[1] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp.1097-1105, 2012.

[2] Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal Processing Magazine 29.6 (2012): 82-97.

[3] Yao, Kaisheng, et al. "Recurrent neural networks for language understanding." INTERSPEECH. 2013.

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio. "On the Properties of Neural Machine Translation: EncoderDecoder Approaches". arXiv:1409.1259

[5] Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches." arXiv preprint arXiv:1409.1259 (2014).

[6] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S. Human-level control through deep reinforcement learning. Nature. 2015 Feb 26;518(7540):529-33.

[7] Timothy P. Lillicrap , Jonathan J. Hunt , Alexander Pritzel, Nicolas Heess, Tom Erez, Yuvalassa, David Silver; Daan Wierstra, Continuous control with deep reinforcement learning, International Conference on Learning Representations, 2016.

[8] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S. Mastering the game of Go with deep neural networks and tree search. Nature. 2016 Jan 28;529(7587):484-489.

[9] Watkins, Christopher JCH and Dayan, Peter. Q-learning. Machine learning, 8(3-4):279292, 1992.

[10] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[11] Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

[12] Matthew Hausknecht, Deep Reinforcement Learning in Parameterized Action Space, International Conference on Learning Representations, 2016

[13] Uhlenbeck, George E and Ornstein, Leonard S. On the theory of the brownian motion. Physical review, 36(5):823, 1930.

[14] Abadi, Martn, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. Proceedings of the International Conference on Learning Representations, 2015.

[16] Patwardhan, Rohit S., and R. Bhushan Goapluni. "A moving horizon approach to input design for closed loop identification." Journal of Process Control 24.3 (2014): 188-202.

[17] Underwood, D.M. and Crawford, R.R. (1991). Dynamic nonlinear modeling of a hot-water-to-air heat exchanger for control applications. American Society of Heating, Refrigerating and Air-Conditioning Engineers