6th International Symposium on
Advanced Control of Industrial Processes (AdCONIP)
May 28-31, 2017. Taipei, Taiwan

Deep Reinforcement Learning Approaches for Process Control

S.P.K. Spielberg[1], R.B. Gopaluni[1], P.D. Loewen[2]

Abstract— **In this work, we have extended the current success of deep learning and reinforcement learning to process control problems. We have shown that if reward hypothesis functions are formulated properly, they can be used for industrial process control. The controller setup follows the typical reinforcement learning setup, whereby an agent (controller) interacts with an environment (process) through control actions and receives a reward in discrete time steps. Deep neural networks serve as function approximators and are used to learn the control policies. Once trained, the learned network acquires a policy that maps system output to control actions. Though the policies are not explicitly specified, the deep neural networks were able to learn policies that are different from the traditional controllers. We evaluated our approach on Single Input Single Output Systems (SISO), Multi-Input Multi-Output Systems (MIMO) and tested it under various scenarios.**

I. INTRODUCTION

It is common in the process industry to use controllers ranging from proportional controllers to advanced predictive based controllers such as Model Predictive Controllers (MPC). However, classical controller design procedure involves careful analysis of the process dynamics, development of an abstract mathematical model, and finally, derivation of a control law that meets certain design criteria. In contrast to the classical design process, reinforcement learning is geared towards learning appropriate closed-loop controllers by simply interacting with the process and incrementally improving control behaviour. The promise of such an approach is appealing: instead of using a time consuming design process, the controller learns the process behaviour by interacting directly with the process. Moreover, the same underlying learning principle can be applied to a wide range of different process types: linear and nonlinear systems; deterministic and stochastic systems; single input/output and multi input/output systems. In addition, from a system identification perspective, both model identification and controller learning are performed simultaneously. This specific controller differs from traditional controllers in that it assumes no predefined control law nor does it have an explicit predefined control law, but rather learns the control law from experiences. With the proposed approach the reward function serves as an objective function indirectly. The drawbacks of standard control algorithms are: (i) a complex dynamic model of the process is required, (ii) model maintenance is often very difficult and (iii) and online adaptation is rarely achieved. The proposed reinforcement learning controller is an efficient alternative to standard algorithms and it automatically allows for continuous online tuning of the controller.

Reinforcement learning (RL) has been used in process control for more than a decade,[10]. However, the available algorithms do not take into account the improvements made through recent progress in the field of artificial intelligence especially deep learning [1]. Remarkably, human level control has been attained in games [2] and physical tasks[3] by combining deep learning and reinforcement learning [2]. Recently, these controllers have even learnt the optimal control policy of the complex Go game [15]. However, the current literature is primarly focussed on games and physical tasks. The objective of these tasks differs slightly from that of process control. In process control, the task is to take the outputs close to the set point while meeting certain constraints, whereas in games and physical tasks the objective is rather generic. For instance: in games, the goal is to take an action to eventually win a game; in physical tasks, to make a robot walk (or) stand. This paper discusses how the success in deep reinforcement learning can be applied on process control problems. In process control, action spaces are continuous and reinforcement learning for continuous action spaces has not been studied until [3]. This work aims at extending the ideas in [3] to process control applications.

The main advantages of the proposed algorithm are as follows (i) it does not involve deriving an explicit control law; (ii) it does not involve deriving first principle models as they are learnt by simply interacting with the environment (iii) the learning policy with deep neural networks is rather fast.

The paper is organized as follows: Section II provides background information. Section III highlights the system configuration in a reinforcment learning perspective. Section IV outlines the technical approach of the learning controller. Section V empirically verifies the effectiveness of our approach. Section VI includes discussions and extensions of our approach followed by conclusion in Section VII.

II. BACKGROUND

Reinforcement learning (RL) is an approach to automating goal-directed learning and decision-making. This approach is meant to solve problems in which an agent interacts with an environment and receives reward signal at each time step. RL algorithms aim to find a policy, which is a mapping from state to action, that maximizes the expected cumulative reward (value function) under that policy. The two main approaches used to achieve this goal are (1) Policy based approach: searches directly for the optimal policy which achieves maximum future reward; (2) Value based approach:

[1]Department of Chemical and Biological Engineering, University of British Columbia spiel@chbe.ubc.ca
[2]Department of Mathematics, University of British Columbia loew@ubc.ca

978-1-5090-4396-5/17/$31.00 ©2017 Crown

estimates the optimal value function which is the maximum value achievable under any policy. Storing the value function (or) policy might not be possible especially if the state-action pairs are high dimensional. Hence, function approximators like linear regression, Neural networks are used with RL. Since it has been found that deep neural networks serve as effective functional approximators and have found great success in image[1], speech [7] and language understanding [8], deep neural networks are used to approximate value function or policy resulting in Deep Reinforcement Learning (DeepRL). DeepRL aims at learning the policy and/or value function end-to-end with or without the plant model. However, using neural networks as such might not necessarily gaurantee convergence and stability [13]. Recent success of Deep Q nework [2],[9] 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. Process control applications involve continuous states and action space. Policy gradient algorithms are widely used RL techniques to address continuous control problems [12]. We have used actor-critic based deterministic policy gradient similar to [3]. The actor learns the policy with the help of critic which in turn learns the value function. At each time step as the agent interacts with the environment both critic and actor are updated and the actor policy continues to improve with respect to the number of iterations. The learnt actor is then used for control.

III. Policy Representation

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. The following subsections provide further details about the representations.

A. State

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, the state is the plant output and setpoint tuple $< y, y_{set} >$.

B. Actions

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

C. 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: 1 shows state-action transition and corresponding reward of the controller in a reinforcement learning framework. For process control task, the objective is to take the output reach 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. The reward function is provided by (1),

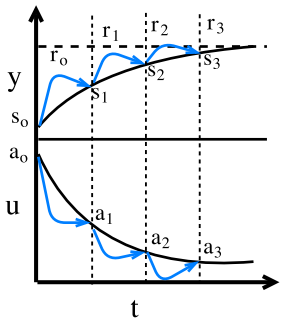


Fig. 1: Transition of states and actions

$$r(s,a,s') = \begin{cases} c, & \text{if } |y^i - y^i_{set}| \leq \varepsilon, \forall i \\ -\sum_i^n |y^i - y^i_{set}|, & \text{otherwise} \end{cases} \tag{1}$$

where i represents i^{th} input 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.

The goal of learning is to find a control policy, π that maximizes the expected value of the cumulative reward, R. Here, R can be expressed as the time-discounted sum of all transition rewards, r_i, from the current action up to a horizon T (where T may be infinite) i.e.,

$$R(s_0) = r_0 + \gamma r_1 + ... + \gamma^T r_T \tag{2}$$

where $r_i = r(s_i, a_i, s'_i)$ and γ is a discount factor. The sequence of states and actions are determined by the policy and the dynamics of the system. The discount factor ensures that the cumulative reward is bounded, and captures the fact that events occurring in the distant future are likely to be of less consequential than those occurring in the more immediate future. The goal of RL is to maximize the expected cumulative reward.

D. 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 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 layers are batch normalized [10]. The critic network predicts the Q-values for each actor and actor network proposes an action for the given state. During the final runtime, 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.

IV. LEARNING

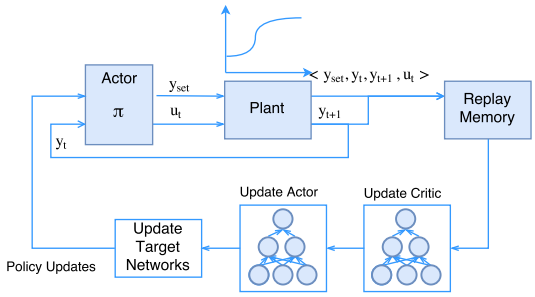


Fig. 2: Learning Overview

The overview of control and learning of the control system is shown in Fig 2. The algorithm for learning the control policy is given in Algorithm 1. The learning algorithm is inspired by [3] with modifications to account for set point tracking and other recent advancements discussed in [4],[5]. In order to facilitate exploration, we added a noise sampled from Ornstein-Uhlenbeck (OU) process [14] discussed similarly in [3]. 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. 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. 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 motivation for using replay memory is to break the correlated samples obtained used for training. The learning algorithm uses actor-critic framework. The critic network serves as a value estimator and it estimates the 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, $L(W_c) = \mathbb{E}\left[(r + \gamma Q_t - Q(s, a, W_c))^2\right]$. Hence, the critic network is updated using this loss with the gradient given by,

$$\frac{\partial L(W_c)}{\partial W_c} = \mathbb{E}\left[(r + \gamma Q_t - Q(s, a, W_c))\frac{\partial Q(s, a, W_c)}{\partial W_c}\right] \quad (3)$$

where $Q_t = Q(s', \pi(s', W_a^t), W_c^t)$, denotes target values and W_a^t, W_c^t are weights of target actor and critic respectively. The discount factor, $\gamma \in [0, 1]$ is the present value of future rewards. 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. We chose $\gamma = 0.99$ in our simulations. The expectation in (3) is over the mini-batches sampled from RM. 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

Algorithm 1 Learning Algorithm

$W_a, W_c \leftarrow$ initialize random weights
initialize Replay memory, RM with random policies
for episode = 1 to E **do**
  Reset the OU process noise $\mathcal{N}$
  Specify setpoint, y_{set} at random
  for step = 1 to T **do**
    $s \leftarrow < y_t, y_{set} >$
    $a \leftarrow$ action, $u_t = \pi(s, W_a) + \mathcal{N}_t$
    Execute action u_t on the plant
    $s' \leftarrow < y_{t+1}, y_{set} >$, observe state at next instant
    $r \leftarrow$ reward
    Store the tuple $< s, a, s', r >$ in RM
    Sample a minibatch of n tuples from RM
    Compute $y^i = r^i + \gamma Q_t^i \ \ \forall i \in$ minibatch
    Update Critic:
    $W_c \leftarrow W_c + \alpha(\frac{1}{n}\sum_i^n (y^i - Q(s^i, a^i, W_c)\frac{\partial Q(s^i, a^i, W_c)}{\partial W_c})$
    Compute $\nabla_p^i = \frac{\partial Q(s^i, a^i, W_c)}{\partial a^i}$
    Invert ∇_p^i by (5)
    Update Actor:
    $W_a \leftarrow W_a + \alpha\frac{1}{n}\sum_1^n (\nabla_p^i \frac{\partial \pi(s^i, W_a)}{\partial W_a})$
    Update Target Critic:
    $W_a^t \leftarrow \tau W_a + (1 - \tau)W_a^t$
    Update Target Actor:
    $W_c^t \leftarrow \tau W_c + (1 - \tau)W_c^t$
  end for
end for

is updated using gradient given by,

$$\frac{\partial J(W_a)}{\partial W_a} = \mathbb{E}\left[\frac{\partial Q(s, a, W_c)}{\partial a}\frac{\partial \pi(s, W_a)}{\partial W_a}\right] \quad (4)$$

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.

Target Network: Similar to [3], we use seperate target networks for actor and critic. We freeze the target networks and replace it with existing networks once in 500 iterations. This makes the learning an off-policy algorithm.

Prioritized Experience Replay: While drawing samples from replay buffer, we use prioritized experienced replay as proposed in [4]. Emprically, we found an acceleration in learning compared to uniform random sampling.

Inverting 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 inverted the gradients of $\frac{\partial Q(s,a,W_c)}{\partial a}$ given in [4] using the following

transformation,

$$\nabla_p = \nabla_p. \begin{cases} (p_{max} - p)/(p_{max} - p_{min}), & \text{if } \nabla_p \text{suggests increasing p} \\ (p - p_{min})/(p_{max} - p_{min}), & \text{otherwise} \end{cases} \tag{5}$$

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.

V. Simulation Results

The output and input response from the learned polices are illustrated in Fig. 4-7. The final policies for the learned SISO system, given in (6), are the result of 7,000 iterations of training collecting about 15,000 tuples and requiring about 24 hours of training on a NVIDIA 960M GPU. For learning the MIMO system given in (7) the compute time on the same GPU was about 30 hours requiring 50K tuples. The deep neural networks are built using Tensorflow [6]. The learning time remains dominated because of the mini-batch training at each time step.

A. Example 1- SISO Systems

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 manufacture of paper in a paper 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 transfer function of the system is given below:

$$G(s) = \frac{0.05s}{1 - 0.6s} \tag{6}$$

The time step used for simulation is 1 second.
The system is learned using Algorithm 1. The portion of the learning curve of the SISO sytem given in (6) is shown in Fig 3 and the corresponding learned policy is shown in Figs 4 and 5.

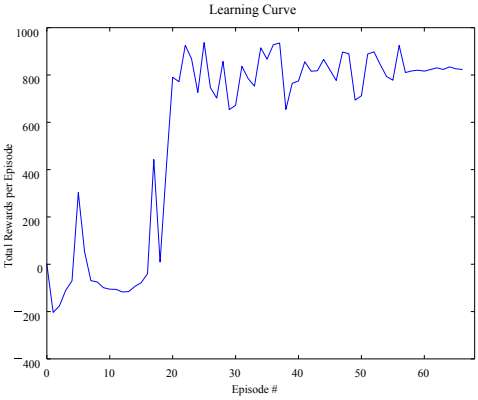


Fig. 3: Learning Curve

The learned controller is also tested on various other cases, such as the response to setpoint change and the effect of

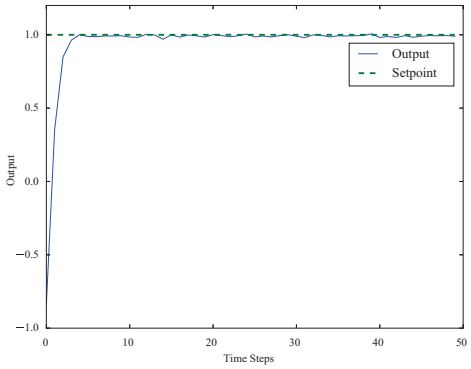


Fig. 4: Output response of the learned policy

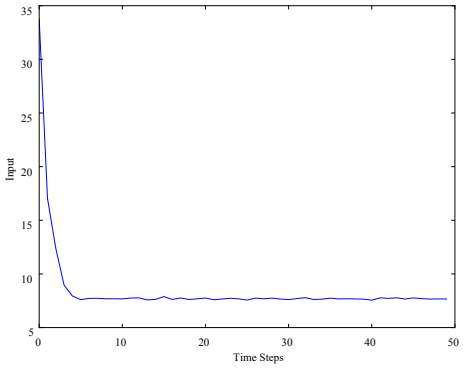


Fig. 5: Input profile of the learned policy

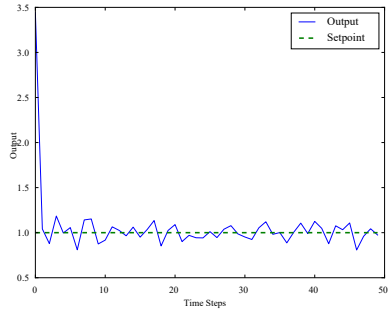
output and input noise. The results are shown in Fig. 6. For learning setpoint changes, the learning algorithm was shown integer-valued setpoints in the range $[0, 10]$. It can be seen from Fig. 6 (i), (l) that the agent has learned how to track even setpoints it was not shown during training.

B. Example 2- MIMO System

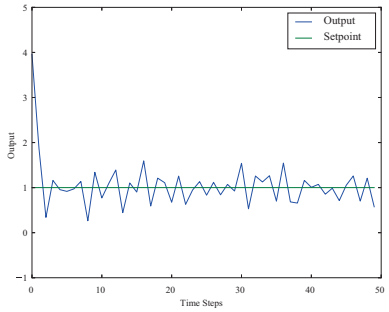
Our approach is tested on a high purity distillation column system. This MIMO system, whose transfer function model is given in (7), is ill-conditioned from a systems point of view. The learned policy of the system is shown in Fig. 7

$$G(s) = \begin{bmatrix} \frac{0.878}{\tau s+1} & -\frac{0.864}{\tau s+1} \\ \frac{1.0819}{\tau s+1} & -\frac{1.0958}{\tau s+1} \end{bmatrix} \tag{7}$$

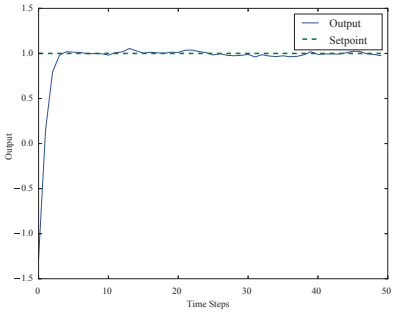
We choose the number of steps per episode to be 200 for learning the control policies. When systems with larger time constants are encountered, we usually require more steps per episode. The reward hypothesis formulation serves as a tuner to adjust the performance of the controller, how setpoints should be tracked, etc. When implementing this approach on a real plant, effort can often be saved by warm-starting the deep neural network through prior training on a simulated model. Experience also suggests using exploration noise with higher variance in the early phase of learning, to encourage high exploration.



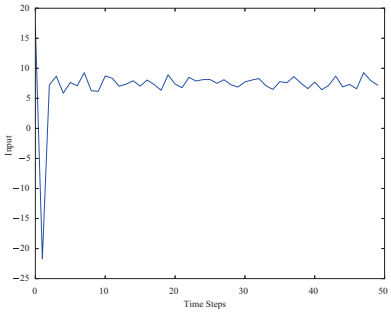
(a) Output response with output noise of $\sigma^2 = 0.1$



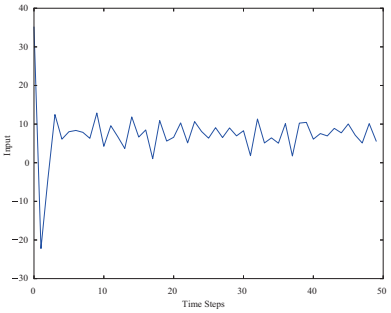
(b) Output response with output noise of $\sigma^2 = 0.3$



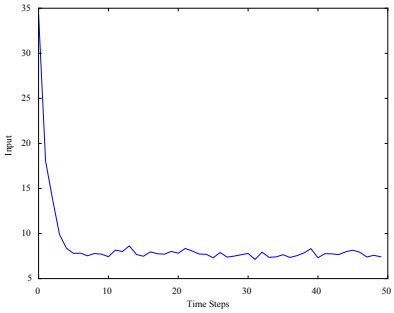
(c) Output response with input noise of $\sigma^2 = 0.1$



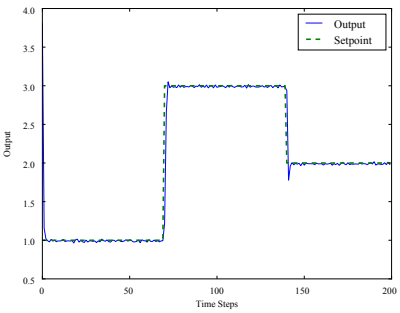
(d) Input response of Fig 6(a) with output noise of $\sigma^2 = 0.1$



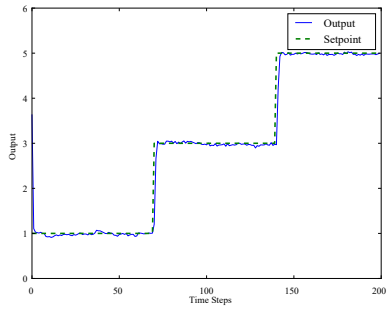
(e) Input response of Fig 6(b) with output noise of $\sigma^2 = 0.3$



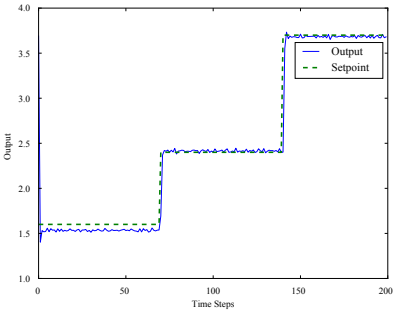
(f) Input response of Fig 6(c) with input noise of $\sigma^2 = 0.1$



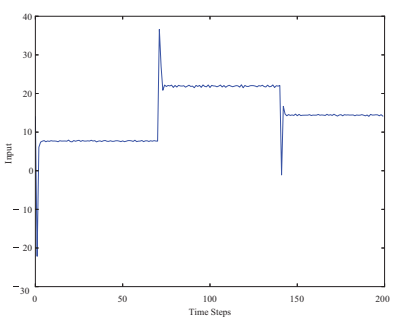
(g) Output response - setpoint change with output noise of $\sigma^2 = 0.1$



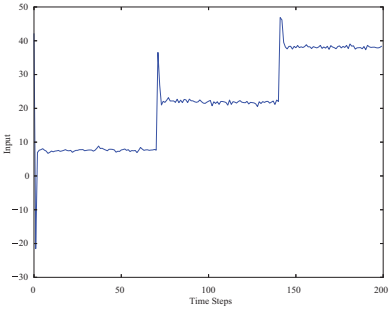
(h) Output response - setpoint change with input and output noise of $\sigma^2 = 0.1$



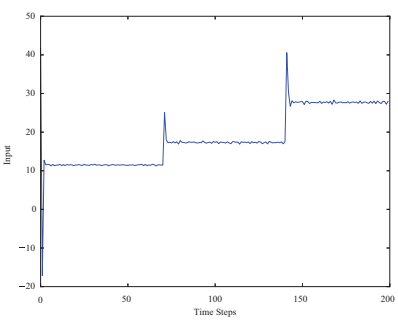
(i) Output Response - setpoints unseen during training



(j) Input response of Fig 6(g) - setpoint change with output noise of $\sigma^2 = 0.1$

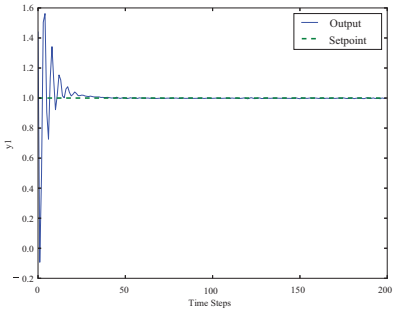


(k) Input response of Fig 6(h) - setpoint change with input and output noise of $\sigma^2 = 0.1$

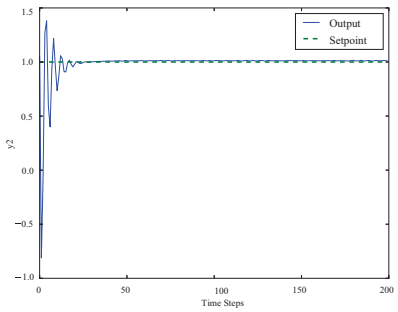


(l) Input response of Fig 6(i) - setpoints unseen during training

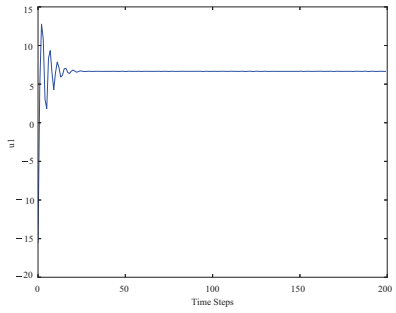
Fig. 6: Output and Input Responses for various cases



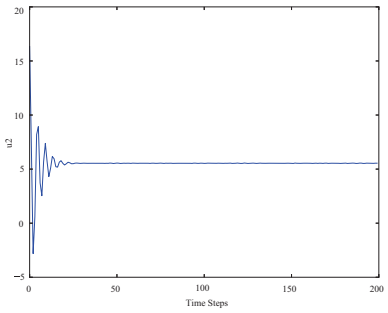
(a) Response of output y_1 with output noise of $\sigma^2 = 0.01$



(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. 7: Output and Input response of system given in eqn (2)

VI. Conclusion

We have developed an artificial intelligence based approach to process control using deep learning and reinforcement learning. This framework supports the development of control policies that can learn directly even with high-dimensional states. This avoids the need for hand-crafted feature descriptors, controller tuning, deriving control laws, and developing mathematical models. However, the current approach is tested on linear systems. Future work will extend this approach on non-linear systems. We believe industrial process control will see rapid and significant advances in the field of deep and reinforcement learning in the near future.

ACKNOWLEDGEMENT

We would like to thank Mitacs for financial support through the Globalink Graduate fellowhip, and Smriti Shyammal 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] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg,et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529-533, 2015.

[3] 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.

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

[5] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver. Prioritized Experience Replay, International Conference on Learning Representations, 2016

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

[7] 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.

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

[9] 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.

[10] Hoskins, J. C., & Himmelblau, D. M. (1992). Process control via artificial neural networks and reinforcement learning. Computers & chemical engineering, 16(4), 241-251.

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

[12] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In ICML, 2014.

[13] D. C. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming. Athena Scientific, Belmont, Mass., 1996

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

[15] 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.