# Data-Driven Process Control via Reinforcement Learning and Recurrent Neural Networks

Nathan P. Lawrence[1], Philip D. Loewen[1], Gregory E. Stewart[2], and R. Bhushan Gopaluni*[3]

[1]Department of Mathematics, University of British Columbia, Vancouver, BC V6T 1Z2, Canada.
[2]Honeywell, North Vancouver, BC V7J 3S4, Canada.
[3]Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, BC V6T 1Z3, Canada.

*Abstract Overview*

Model-based controllers are ubiquitous in process control. In order to maintain overall performance in the presence of gradual changes in the process (for example due to changes in feedstock, wear and tear on equipment and instrumentation, etc), such controllers require re-tuning. For Model Predictive Control (MPC), or other modern approaches where performance is directly correlated with the quality of the underlying process model, re-tuning the controller requires re-identifying the model, and this can take up to two weeks for some applications (Kano and Ogawa, 2009). Our recent results in (Spielberg et al., 2017) and (Spielberg et al., 2019) use Deep Reinforcement Learning (DRL) to develop an input-output controller for set-point tracking problems in discrete-time nonlinear processes by simply using the current output information from the system. Here, we propose a general RL state definition to include past inputs and outputs, and accompanying Recurrent Neural Network (RNN) structures to parameterize the actor and critic. Further, we propose a new reward function that involves both the current deviation from a set-point and the difference in control actions across several previous time-steps, thereby incentivizing smoother controls.

*Keywords*

Process control, Deep reinforcement learning, Actor-critic networks

## Introduction

The first successful implementations of RL methods in process control utilized approximate dynamic programming (ADP) methods for optimal control of discrete-time nonlinear systems (Lee and Lee, 2006; Lee et al., 2006; Kaisare et al., 2003; Lee and Lee, 2004, 2005). While these results illustrate the applicability of RL in controlling discrete-time nonlinear processes, they are also limited to processes for which at least a partial model is available or can be derived through system identification (Lee and Lee, 2006; Lee et al., 2006; Kaisare et al., 2003; Lee and Lee, 2004).

Recently, several data-based approaches have been proposed to address the limitations of model-based RL in control. In (Lee and Lee, 2005; Mu et al., 2017), a data-based learning algorithm was proposed to derive an improved control policy for discrete-time nonlinear systems using ADP with an identified process model. Similarly, (Lee and Lee, 2005) proposed a Q-learning algorithm to learn an improved control policy in a model-free manner using only input-output data. While these methods remove the requirement for having an exact model, they still present several issues. For example, the learning method proposed in (Lee and Lee, 2005; Mu et al., 2017) is still based on ADP, so its performance relies on the accuracy of the identified model.

In our preliminary work (Spielberg et al., 2017), we use Deep Reinforcement Learning and an actor-critic architecture to develop a model-free, input-output controller for set-point tracking problems in discrete-time nonlinear processes. However, we define the RL state[1] for a process to be

$$s_t = \langle y_t, y_{t,sp} - y_t \rangle, \tag{1}$$

where $y_t$ is the system output at time $t$ and $y_{t,sp}$ is the respective set-point. We propose a more general RL state definition that includes past input and output values, thereby better representing the system dynamics. Recall that Recurrent Neural Networks are specialized neural networks for processing sequential data of arbitrary length (Goodfellow et al., 2016). Therefore, in order to capture the temporal dependencies within the RL state, we adapt the architecture in (Spielberg et al., 2017) to use RNN's for both the actor and the critic. We assume the setting is a Markov Decision Process with action and state spaces $\mathcal{A}$ and $\mathcal{S}$, respectively, and state transition distribution $p$ satisfying the Markov property

$$p(s_{t+1}|S_t = s_t, A_t = u_t, \ldots, S_1 = s_1, A_1 = u_1) \tag{2}$$
$$= p(s_{t+1}|S_t = s_t, A_t = u_t), \tag{3}$$

where $s_t \in \mathcal{S}$ and $u_t \in \mathcal{A}$.

*Corresponding author: bhushan.gopaluni@ubc.ca

[1]The notion of *state* in Reinforcement Learning differs from that in control, such as in a state-space model

### Actor-Critic Framework

We outline the DRL controller with the deterministic actor-critic method. Further details and the precise algorithm can be found in (Spielberg et al., 2019). The *actor network* defines the agent's (deterministic) policy, while the *critic network* evaluates the policy proposed by the actor. More precisely, the actor-critic method is a combination of policy gradient methods and $Q$-learning via temporal difference (TD) update (Sutton et al., 2000; Degris et al., 2012; Bhatnagar et al., 2008). Our intended applications in process control have continuous state and action spaces. Thus, we consider a parameterized deterministic policy for the actor, $\mu(\cdot, W_a)\colon \mathcal{S} \to \mathcal{A}$ such that

$$u_t = \mu(s_t, W_a), \tag{4}$$

where $W_a$ refers to a collection of parameters that we will iteratively update using batch gradient ascent so as to maximize the expected reward

$$J(\mu(\cdot, W_a)) = \mathbb{E}_{h\sim p^{\mu}(\cdot)}\left[\sum_{t=1}^{\infty}\gamma^{k-1} r(s_t, \mu(s_t, W_a))\middle| s_0\right] \tag{5}$$

where $s_0 \in \mathcal{S}$ is a starting state, $0 < \gamma < 1$ is a *discount* factor, and $r$ denotes the reward signal obtained from a state-action pair.

To perform this update, we use the policy gradient theorem for deterministic policies (Silver et al., 2014) to approximate the gradient of $J$ in terms of the critic, $Q^{\mu}(\cdot, \cdot, W_c)$, as follows:

$$\widehat{\nabla}_{W_a} J(\mu(\cdot, W_a)) = \tag{6}$$
$$\mathbb{E}_{s_t\sim\rho^{\beta}_{\gamma}(\cdot)}\big[\nabla_a Q^{\mu}(s_t, a, W_c)|_{a=\mu(s_t, W_a)} \nabla_{W_a}\mu(s_t, W_a)\big],$$

where $\rho^{\beta}_{\gamma}(s) = \sum_{n=0}^{\infty}\gamma^t p(s_t = s|s_0, \mu)$ is a discounted state visitation distribution. We note that Eq. (6) is maximized only when the policy parameters $W_a$ are optimal, which then leads to the update scheme

$$W_{t+1} \leftarrow W_t + \alpha_{a,t}\widehat{\nabla}_{W_a} J(\mu(\cdot, W_a))\Big|_{W_a = W_t}. \tag{7}$$

The critic in Eq. 6 is a parameterized $Q$-function, $Q(\cdot, \cdot, W_c) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which indicates how desirable a given policy is from a starting state-action pair. The parameters $W_c$ are updated so as to minimize the loss

$$\mathcal{L}_t(W_c) = \tag{8}$$
$$\mathbb{E}_{s_t\sim\rho^{\beta}(\cdot), a_t\sim\beta(\cdot|s_t)}\big[(Q^{\star}(s_t, a_t) - Q(s_t, a_t, W_c))^2\big],$$

where $Q^*$ is the optimal $Q$-function and $\rho^{\beta}$ is a discounted state visitation distribution under an exploration policy $\beta$. In practice, we estimate the values of $Q^*$ with a bootstrap target.

We update the parameters in Eq. (8) for the critic using batch gradient descent, where our batch data come from a cache of tuples of the form $(s_t, u_t, s_{t+1}, r(s_t, s_{t+1}, u_t))$. Hence, it is important that our state properly captures the dynamics of the system it represents, so as to make meaningful parameter updates.

### General RL State and Rewards

We define the *RL state* in terms of trailing sequences of input and output values with possibly different parameters $d_y$ and $d_a$:

$$s_t = \langle y_t, y_{t,sp} - y_t, \ldots, y_{t-d_y}, y_{t-d_y,sp} - y_{t-d_y}, \tag{9}$$
$$u_{t-1}, \ldots, u_{t-d_a}\rangle.$$

Here $y_{.,\mathrm{sp}}$ is the user-specified sequence of set-point values, typically a constant sequence. We allow $d_y = d_a = 0$, which corresponds to a 'memoryless' state $s_t = \langle y_t, y_{t,sp} - y_t\rangle$ defined only in terms of the current output.

In (Spielberg et al., 2019), a feedforward neural network structure is used to represent the parameterized actor and critic. However, the more general RL state in Eq. (9) contains time-dependent terms, making a feedforward neural network inadequate for representing the actor and critic networks. For example, a feedforward neural network representation for the actor would take as its input the state $s_t$ from Eq. (9) and learn separate parameters for each individual position of the state vector, despite the fact that the measurements in the state vector are temporally related and the next state vector $s_{t+1}$ contains many of the same elements as its predecessor $s_t$.

In contrast, a Recurrent Neural Network (RNN) shares its parameters across each time-step (Goodfellow et al., 2016), leading to a structure with hidden states that depends explicitly on past values of a sequential input. Therefore, we parametrize the actor-critic architecture using RNN models. Specifically, both the actor and critic networks are sequentially fed tuples of the form $\langle y_t, y_{t,sp} - y_t\rangle$, but the output steps vary slightly. For the actor, the final hidden state passes through an affine transformation to give the action $u_t$. For the critic, we scale the action $u_t$ and the final hidden state, then pass them through a single-layer feedforward network with some non-linear activation. The RNN structure may have multiple layers and more complicated cell structures, such as Long Short-Term Memory (LSTM) (see (Goodfellow et al., 2016)).

Now, in the more general setting of our RL state given by Eq. (9), we propose a new reward function designed to encourage set-point tracking and discourage abrupt changes in the control. Define

$$\Delta\vec{u}_t := \big[u_t - u_{t-1}, u_{t-1} - u_{t-2}, \ldots, u_{t-d_a+1} - u_{t-d_a}\big]^{\mathsf{T}}. \tag{10}$$

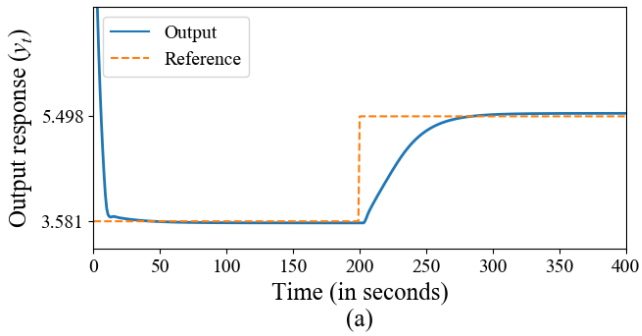
Then we express the cost associated with the state $s_t$ and control $u_t$ by

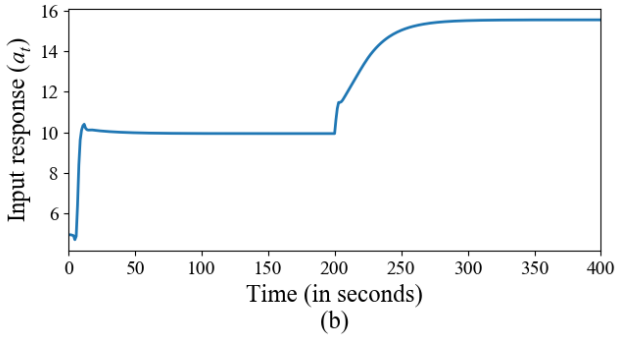
$$R(s_t, s_{t+1}, u_t) := |y_{t,sp} - y_t| + \sqrt{\Delta\vec{u}_t^{\mathsf{T}} Q \Delta\vec{u}_t} \tag{11}$$
$$= |y_{t,sp} - y_t| + \|\Delta\vec{u}_t\|_Q\,, \tag{12}$$

where $Q$ is positive definite. Alternatively, $Q$ can be taken to be zero. Our reward function is then defined to be

$$r(s_t, s_{t+1}, u_t) := -\,R(s_t, s_{t+1}, u_t). \tag{13}$$

Figure 1: *(left) Tracking of arbitrary set-points; (right) Input generated by the DRL controller.*




One straightforward choice for the matrix $Q$ in our cost function is

$$Q = \text{diag}[\alpha_1, \alpha_2, \ldots, \alpha_{d_a}] \tag{14}$$

where $0 < \alpha_{d_a} \leq \cdots \leq \alpha_1 \ll 1$.

## Results

We demonstrate our approach with a simple example. Consider a plant given by the following discrete-time transfer function:

$$G(z) = \frac{.05z^{-3}}{1 - .86z^{-1}}. \tag{15}$$

Note that Eq. (15) is used purely for simulation purposes. That is, our algorithm does not require a model for the plant dynamics. Fig. (1) shows the input and output responses of the DRL controller for two randomly generated set-points.

The DRL controller was trained on integer-valued set-points between in [0,5] for 300 episodes, each consisting of 200 time steps. We implemented our algorithm in `Python`. The actor and critic networks were built and trained using `Tensorflow`. The actor was modeled with a single-layer RNN architecture with LSTM cells and 150 neurons. The final output was wrapped to a scalar, so as to provide the next action $u_t$ following $s_t$, where $s_t$ is defined in Eq. (9) with $d_y = 2, d_a = 2$. The past two actions were used in Eq. (13). The critic architecture is the same as the actor, except we added a single feed-forward layer after the final output in order to incorporate the most recent action $u_t$. All layers in our models used ReLU activation.

## Conclusion

We propose a new RL state in terms of the past inputs and outputs of a process. This representation is more characteristic of the underlying dynamics in which the DRL controller is operating. As such, this representation motivates the the utility of an RNN structure for the actor and critic networks. To conclude, we define a new reward hypothesis that weighs the increments of the control actions and the current output error; maximization of this reward function is then obtained through smooth set-point tracking by the DRL controller.

## References


Bhatnagar, S., Ghavamzadeh, M., Lee, M., and Sutton, R. S. (2008). Incremental natural actor-critic algorithms. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 105–112, Vancouver, Canada.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv Preprint, arXiv:1205.4839*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Kaisare, N. S., Lee, J. M., and Lee, J. H. (2003). Simulation based strategy for nonlinear optimal control: application to a microbial cell reactor. *International Journal of Robust and Nonlinear Control*, 13(3-4):347–363.

Kano, M. and Ogawa, M. (2009). The state of the art in advanced chemical process control in Japan. *IFAC Proceedings Volumes*, 42(11):10–25.

Lee, J. H. and Lee, J. M. (2006). Approximate dynamic programming based approach to process control and scheduling. *Computers & Chemical Engineering*, 30(10-12):1603–1618.

Lee, J. M., Kaisare, N. S., and Lee, J. H. (2006). Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. *Journal of process control*, 16(2):135–156.

Lee, J. M. and Lee, J. H. (2004). Simulation-based learning of cost-to-go for control of nonlinear processes. *Korean Journal of Chemical Engineering*, 21(2):338–344.

Lee, J. M. and Lee, J. H. (2005). Approximate dynamic programming-based approaches for input–output data-driven control of nonlinear processes. *Automatica*, 41(7):1281–1288.

Mu, C., Wang, D., and He, H. (2017). Novel iterative neural dynamic programming for data-based approximate optimal control design. *Automatica*, 81:240–252.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China.

Spielberg, S., Tulsyan, A., Lawrence, N. P., Loewen, P. D., and Gopaluni, R. B. (2019). Towards self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*. `https://doi.org/10.1002/aic.16689`.

Spielberg, S. P. K., Gopaluni, R. B., and Loewen, P. D. (2017). Deep reinforcement learning approaches for process control. In *Proceedings of the 6th IFAC Symposium on Advanced Control of Industrial Processes*, pages 201–206, Taipei, Taiwan.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1057–1063.