Mario R. Eden, Marianthi Ierapetritou and Gavin P. Towler (Editors) Proceedings of the 13th International Symposium on Process Systems Engineering – PSE 2018
July 1-5, 2018, San Diego, California, USA
https://doi.org/10.1016/B978-0-444-64241-7.50008-2

Reinforcement Learning – Overview of Recent Progress and Implications for Process Control

Thomas A. Badgwell[a], Jay H. Lee[b], Kuang-Hung Liu[a]

[a]*ExxonMobil Research & Engineering Company, Clinton, NJ, USA, thomas.a.badgwell@exxonmobil.com*
[b]*Korea Advanced Institute of Science and Technology, Daejeon, Korea jayhlee@kaist.ac.kr*

Abstract

This paper provides a brief introduction to Reinforcement Learning (RL) technology, summarizes recent developments in this area, and discusses their potential implications for the field of process control. The paper begins with a brief introduction to RL, a machine learning technology that allows an agent to learn, through trial and error, the best way to accomplish a task. We then highlight two new developments in RL that have led to the recent wave of applications and media interest. A comparison of the key features of RL and Model Predictive Control (MPC) is then presented in order to clarify their similarities and differences. This is followed by an assessment of five ways that RL technology can potentially be used in process control applications. A final section summarizes our conclusions and lists directions for future RL research that may improve its relevance for process control applications.

Keywords: Reinforcement Learning, Model Predictive Control, Process Control.

1. Introduction

In March 2016, a computer program named AlphaGo defeated Lee Sedol, an 18 time world champion, four games to one, at the ancient Chinese game of Go, generating widespread media attention. In May 2017, a new and improved version called AlphaGo Master soundly defeated Ke Jie, the reigning world No. 1 ranked player, three games to none. It is said that Google DeepMind was already in possession of AlphaGo Zero, a version much stronger than the Master version, at the time (Silver et al., 2017). Go was invented more than 2,500 years ago and is believed to be the oldest board game widely played today. Despite its relatively simple rules, Go is significantly more complex than chess in terms of combinatorial possibilities and defeating the best human players had been a goal of Artificial Intelligence researchers for many years. AlphaGo owes its success to recent developments in Machine Learning (ML), specifically to a branch of ML known as Reinforcement Learning (RL) (Sutton and Barto, 2016). Other notable successes of RL include manoeuvring self-driving cars, soccer robots, and optimizing the operation of a data center (Knight, 2017). One can find many interesting demonstrations of Reinforcement Learning on YouTube, including a robot that learns to defend a soccer goal and one that learns how to flip pancakes.

Given the recent successes of RL technology, and given the similarities between RL tasks and process control problems, it is natural to ask if there are any implications of these developments for the field of process control. This paper attempts to address this question, building on a prior assessment of Machine Learning presented at the

FOCAPO/CPC 2016 meeting by Lee et al. (2016). The paper begins with introduction to RL, a machine learning technology that allows an agent to learn, through trial and error, the best way to accomplish a task. We then highlight two new developments in RL that have led to the recent wave of applications and media interest. A comparison of the key features of RL and Model Predictive Control (MPC) is then presented in order to clarify their similarities and differences. This is followed by an assessment of five ways that RL technology can potentially be used in process control applications. A final section summarizes our conclusions and lists directions for future RL research that may improve its relevance for process control applications.

2. Reinforcement Learning

Machine Learning is customarily divided into three classes of algorithms: Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL) (Bishop, 2012). In Supervised Learning, an agent is provided with labelled examples (X and Y) and learns a mapping that allows it to predict values of Y for new values of X. For example, an agent may be provided with a number of pictures of animals, each labelled properly with the types of animals in the picture. The agent then learns a mapping that allows it to determine, for example, which of the pictures contain a particular type of animal. In Unsupervised Learning the agent is provided with unlabelled data (X only) and learns about how the data is distributed ($p(X)$). For example, the agent may be able to group pictures according to the types of animals they contain. Reinforcement Learning differs significantly from both Supervised and Unsupervised Learning. A Reinforcement Learning agent has the goal of learning the best way to accomplish a task through repeated interactions with its environment (Sutton and Barto, 2016). In order to accomplish this the agent must evaluate the long-term value of the actions that it takes.

Of course the concept of teaching a computer to accomplish a task has been around since the earliest days of the computer age. Alan Turing, for example, wrote the following in his remarkable paper on computing and intelligence (Turing, 1950):

> *Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education, one would obtain the adult brain…We normally associate punishments and rewards with the teaching process. Some simple child machines can be constructed or programmed on this sort of principle…I have done some experiments with one such child machine, and succeeded in teaching it a few things, but the teaching method was too unorthodox to be considered really successful.*

The development of RL from such early concepts is well documented in what is probably the best overall introduction to the field, the classic text of Sutton and Barto (2016). Two essential threads of research contributed to the modern theory of RL: Animal Psychology contributed key concepts related to learning a task by trial and error, and Optimal Control Theory provided guidance on how to improve task performance as well as how to know when the task was completed in the best possible way.

2.1 Reinforcement Learning Basics

The basic setting for RL, as abstracted from Animal Psychology, is illustrated in Figure 1. In this setting there are only two entities, the Agent and the Environment. At time t the Agent takes an action A_t that affects the environment, causing it to transition from

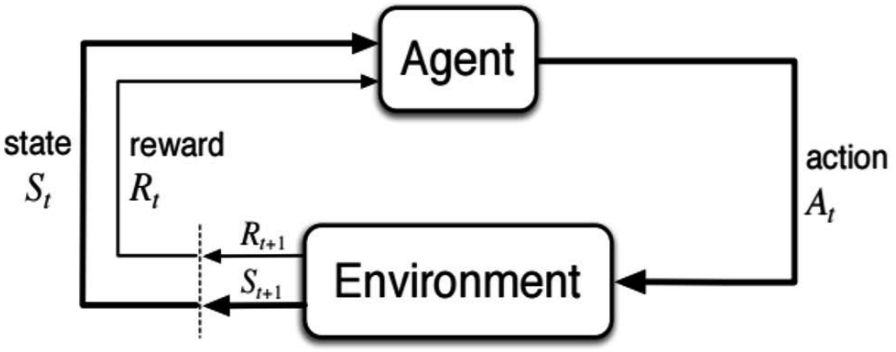


Figure 1 - Basic setting for Reinforcement Learning

state S_t to state S_{t+1}. As a result of the action A_t, and the subsequent transition from state S_t to state S_{t+1}, an immediate reward R_{t+1} is generated. The Agent then uses the state information S_{t+1} and immediate reward R_{t+1} to generate the next action A_{t+1}, continuing the cycle. The goal of the Agent is to learn a mapping from states to actions called a policy $\pi(A_t = a|S_t = a)$ that maximizes a *long-term* sum of future rewards called a value function $v_\pi(s)$:

$$v_\pi(s) = E\{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots |S_t = s\} \quad (1)$$

Here $\gamma \in [0,1]$ is a discount factor that determines the importance of future rewards. When $\gamma = 0$ the Agent acts in a greedy manner, maximizing the reward at the next time step. When $\gamma = 1$ the Agent will place equal importance on all future rewards. Note that every value function $v_\pi(s)$ is associated with a particular policy π that determines the future path through the space of states.

RL algorithms can be thought of as iterative updates that improve the policy so that the associated value function increases for all states. If the Agent modifies the policy appropriately at each iteration, then the policy continues to improve, leading to larger and larger values for the value function at any given state. It is natural to ask if the value function ever attains its best possible, or optimal value, $v_*(s)$. Optimal Control Theory answers this question in the form of Bellman's optimality equation, a condition which must be true for all states if the policy is optimal (necessary condition):

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s,\ a)\,[r + \gamma v_*(s')] \quad (2)$$

Here the term $p(s', r|s,\ a)$ is the transition probability for the environment, known as the transition model. It provides, given the current state s and action a, the probability that the environment will transition to state s' with reward r. For a discrete set of actions and states, and for a given process model, Bellman's optimality equation provides a set of nonlinear equations that can, in principle, be solved directly to obtain the optimal value function $v_*(s)$. Once the optimal value function is known, the associated optimal policy $\pi_*(s)$ can be found using the transition model. In practice, however, these equations usually cannot be solved directly because we either do not know the environment model, or the state dimension is so large (possibly infinite) that the solution cannot be found in a

reasonable time. Sutton and Barto consider all RL algorithms as approximate solutions to Bellman's optimality equation, dealing in various ways with these two limitations (Sutton & Barto, 2016).

For continuous states and actions, the most relevant case for process control applications, the state and action dimensions are infinite, requiring that function approximation methods be used to estimate the optimal value function $v_*(s)$ and policy $\pi_*(s)$. Note that while neural networks are often used for this, other types of function approximations can also be used. Let us assume that the policy function is parameterized by parameters θ:

$$\pi(a|s) \cong \pi_\theta(a|s) \tag{3}$$

Sergey Levine (2017) explains that for this case the goal of the RL algorithm is to solve the following optimization problem:

$$\theta_* = \underset{\theta}{argmax} E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right] \tag{4}$$

This says that the parameters θ are to be chosen so that the resulting policy maximizes the expected value of future rewards. Here $r(s_t, a_t)$ is a reward function and the expectation is over the sequence τ of future actions and states, distributed according the probability $p_\theta(\tau)$:

$$\tau = (s_1, a_1, s_2, a_2, \ldots.) \tag{5}$$

From this perspective, Levine (2017) divides RL algorithms into four main classes:

- Model-Based (MB): estimates the transition model $p(s', r|s, a)$ and then uses this directly for control, to estimate the value function, or to estimate the policy.
- Value-Based (VB): estimates the so called Q function $Q_*(s, a)$ which combines the transition model and the optimal value function in (2), then determines the policy that selects the most beneficial (greedy) action from the current state.
- Policy-Gradient (PG): estimates the value function by summing observed rewards $r(s_{t,} a_{t,})$, then uses the gradient of the objective in (4) to improve the policy by gradient ascent
- Actor-Critic(AC): estimates optimal value function $v_*(s)$ in (2), uses this to predict future rewards $r(s_{t,} a_{t,})$, then uses gradient of objective in (4) to improve the policy by gradient ascent (combines best features of Policy-Gradient and Value-Based)

The last three of these are often referred to as model-free, since they do not require an estimate of the transition model. We will emphasize model-free RL methods here so as to provide a clear contrast with MPC techniques that are widely used in process control today.

The effectiveness of these algorithms on any particular problem may vary widely, due to a number of trade-offs (Levine, 2017):

- Sample efficiency: how many interactions with the environment are required to achieve good convergence?

- Stability and ease of use: how easy is it to choose the algorithm's meta-parameters so as to get reliable convergence?
- Stochastic versus deterministic: does the algorithm work better for stochastic or deterministic environments (and/or policies)?
- Continuous versus discrete: is the algorithm better suited for continuous or discrete states, actions, and/or rewards?
- Episodic or infinite horizon: is the algorithm better suited for episodic or infinite time environments?
- Parallelization: Can the algorithm be parallelized easily when a good simulation of the environment is available?

For example, PG algorithms typically require significantly more interactions with the environment than the other methods (low sample efficiency). However, the actual wall clock time to convergence may be lower if a good simulator is available, since PG algorithms are highly parallelizable. Most of the recent developments in RL theory and application are based on AC algorithms that use Deep Neural Networks (DNN) for function approximation. AC algorithms, first introduced by Witten (1977) and later popularized by Bart, Sutton, and Anderson (1983), combine some of the best features of VB and PG algorithms, resulting in faster, more stable learning.

2.2 Latest developments

We now highlight two relatively recent developments in RL technology that have enabled a significant expansion in the scope and capability of the technology: Deep Learning and Parallelization.

Deep Learning refers to a neural network architecture with a relatively large number of layers, in which the first several layers mostly extract significant features from the data (X), and the last layers are mostly engaged in correlating the extracted features to outputs (Y). This type of architecture, shown in Figure 2, is often referred to as a Deep Neural Network (DNN). Lee et al. (2017) provide more details on the development of Deep Learning technology. The use of DNNs to approximate the value and policy functions is a key factor in the many of the recent RL success stories (Silver, et al., 2016). The ability of a DNN to extract useful features significantly reduces the time and effort required for feature engineering. In the AlphaGo application, for example, the DNNs were able to use the raw 19x19 matrix of board positions to directly deduce the game state (Silver, et al., 2016). One implication for process control applications is that it may be possible to pass all available process measurements to a DNN and allow it to predict future process outputs directly, without having to design and implement a state estimator. This would greatly simplify the design of new control systems, and may provide a straightforward path to improving existing control systems.

A second recent development is the use of parallelization to speed up and stabilize the learning process, which might otherwise be a significant barrier for process control applications. A recent paper from Mnih, et al. at Google DeepMind (2016) discusses the advantages of this. Previous work had found that combining simple online RL algorithms with DNNs led to learning stability problems in that the DNN weights would not converge in a stable manner. This was thought to be due to the correlated nature of the updates that the RL algorithms received online. One popular solution to this problem is to store the

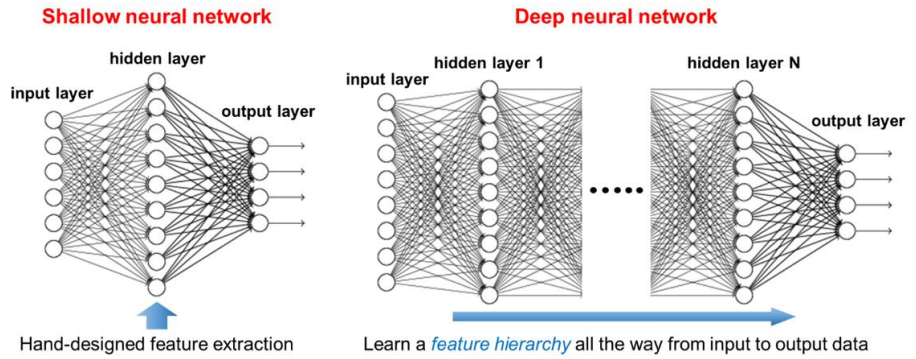


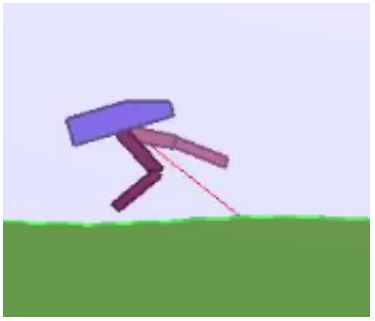
Figure 2 – Comparison of the structure of a Shallow neural network and a Deep Neural Network, from (Lee, at al., 2017)

online data in an experience replay memory, where they can be batched or randomly sampled to train the DNN. Mnih et al. (2016) present an alternative method in which a number of distributed agents execute in parallel on identical copies of the environment, each progressing on its own trajectory through the state space due to the use of different exploration policies. Gradients computed locally by the distributed agents are periodically transmitted to a master agent, where they are used to update a master policy. The master policy is, in turn, transmitted periodically to the distributed agents. There are two significant benefits to this approach. The first is that the learning rate increases dramatically, roughly in proportion to the number of distributed agents. The second is that learning is more stable because the gradient updates are uncorrelated. This means that there is no requirement for an experience replay memory. They demonstrate superior performance of their Asynchronous, Advantage, Actor-Critic (A3C) algorithm on a variety of games and control tasks that include both continuous and discrete actions. They conclude that their A3C algorithm is the most general and successful RL algorithm to date. Of course, all of this depends upon the existence of a quickly executing process simulator of sufficient accuracy to be useful.

An extension of this concept, which may prove useful in robust control applications, is to run a number of agents in parallel on slightly different environments, each corresponding to a possible realization of an uncertain process. The knowledge gained in each environment can be assembled in the master agent in the same way. The resulting master policy would be expected to produce more cautious, robust actions.

2.3 An example RL application – OpenAI Gym Bipedal Robot

We now present a simulated bipedal robot example. The bipedal robot environment comes from the OpenAI Gym, an open-source toolkit for comparing reinforcement learning algorithms. Figure 3 illustrates the bipedal robot, which consists of a head (also called a hull) and two legs. Each leg has two joints – at the hip and the knee. Twenty-four continuous states are measured, without error, to determine the dynamic state of the robot. These include position, speed, and angular velocity of the robot components, a sensor that determines if the legs are in contact with the ground, and ten Lidar distance measurements. The four continuous actions are torques applied to each joint. The reward



24 Continuous States	= [hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints, joints angular speed, legs contact with ground, 10 LIDAR rangefinder measurements]
4 Continuous Actions	= [leg joint torques]
Reward	+x: Every time it moves forward. - 0.1: Apply leg motor torque. -100: Fall down.

Figure 3 - Bipedal robot example – states, actions, and rewards

is how many units of forward motion have been achieved with a small penalty for each action (torque) and a large penalty for falling down. The environment is episodic in that each episode begins with the robot standing at the edge of a field, and the episode ends when the robot either falls down or reaches the other edge of the field.

A Deep RL AC algorithm was implemented for this example. The results at various stages of learning are illustrated below in Figure 4. After 3k episodes the robot is able to make some forward progress but its rear leg is folded beneath it much of the time. After 40k episodes it is able to move forward much more rapidly. Further training leads to a very surprising policy, in which the robot hops on one leg, using the other leg as a tail to maintain its balance. This happens because the reward signal penalizes movement of the joints, so the robot has found a way to optimize its overall reward by decreasing its forward speed and moving its legs less vigorously. This points out an interesting feature of RL technology – RL agents can sometimes produce actions that a human control designer would never think of. The AlphaGo agent exhibited similar behaviour in that it sometimes made moves that no experienced Go player would consider. This can be viewed as either a problem or a learning opportunity depending on your point of view.

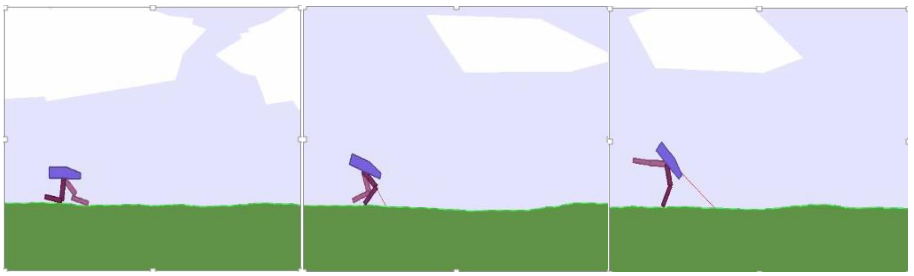


Figure 4 – Results for the bipedal robot using an AC algorithm. On the left after 3k episodes, in the middle after 40k episodes, and on the right after additional training.

Figure 5 presents results from a more sophisticated bipedal robot that learns to traverse a more challenging obstacle course consisting of steps, walls, and pits. This robot uses a parallel AC algorithm similar to that presented by Mnih et al. (2016) discussed above so

that it can learn from a number of environments simultaneously. Each of distributed environments contains the same types of features, but their number and location vary randomly. After sufficient training the robot is able to traverse the obstacles easily, occasionally raising its rear leg in a tail-like manner to restore its balance.

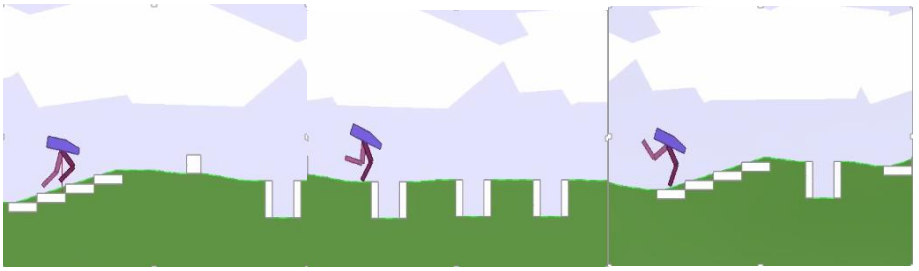


Figure 5 – Results for a bipedal robot using a parallel AC algorithm to traverse a more complex environment consisting of steps, walls, and pits.

3. Comparison of Reinforcement Learning and Model Predictive Control

In order to assess the strengths and weaknesses of RL for process control applications it is instructive to compare RL with MPC, the most widely-used technology developed by the process control community for complex, multivariable control applications (Qin and Badgwell, 2003). In doing so we consider a model-free, on-policy RL algorithm with continuous states and decisions applied to a game, since this is the type of application that has driven much of the recent effort in RL theory and applications. And we consider MPC with a fixed discrete time state-space model, since this is the form that has arguably driven most of the recent developments for this technology. With this in mind, Table 1 provides a comparison of the salient features of these technologies.

The two technologies make somewhat different assumptions about the underlying system. For MPC the underlying system is typically assumed to be a discrete-time state-space system of the form:

$$x_{k+1} = f(x_k, u_k, w_k)\ ;\ \ y_k = g(x_k, v_k) \tag{6}$$

Here x_k is the system state at time k, u_k is the input (action), and y_k is the measured output. The term w_k is a stochastic state disturbance, and the term v_k represents measurement noise. This model form describes how the state traverses through the state space, given a sequence of actions, state disturbances, and measurement noises. RL focuses instead on how the probability of being in a particular state changes from one time step to the next. This is described by a Markov Decision Process (MDP) with the following components (Levine, 2017):

- S - state space, states $s \in S$ (discrete or continuous)
- A - action space, actions $a \in A$ (discrete or continuous)
- T - transition operator, $p(s_{k+1}|s_k, a_k)$
- $r(s, a)$ - reward function, $r: SxA \rightarrow \mathbb{R}$

So basically, a MDP is an unstructured model in discrete state space, describing how the probability of being in a particular state changes over time. It is possible, in principle, to convert the discrete-time state-space model (6) into a MDP by sampling the state transitions. However model-free RL does not require a system model of the form in (6). So model-free RL has the advantage that no explicit system model is required. It also works naturally in a stochastic system environment. However, it has the disadvantage that the relevant system behaviour must then be learned by trial and error.

Table 1: A comparison of Reinforcement Learning and Model Predictive Control

	Reinforcement Learning (model-free)	**Model Predictive Control**
Underlying System Assumption	Markov Decision Process	Discrete-time state space system
Goal(s)	Simple, fixed goal: Just win (don't care about path through state-space)	Complex goal that changes with time: Estimate system state, determine best steady-state consistent with current constraint and setpoint values, follow smooth path consistent with constraints through state-space to best steady-state (smooth, stable path through state-space is critical)
Modeled component	Value function or Policy	Discrete-time state space system
Model learning paradigm	Adaptive model learned through trial and error either offline using simulation or online with real process	Fixed model developed through first principles and/or process identification experiments
Exploration/Exploitation	Simultaneous	Exploration first to get model/Exploitation thereafter
Feedback	Adapting value function or Policy	Disturbance estimation
Online execution time	Very fast - just a forward run of the policy network	Slow - potentially solving three optimization problems at each execution
Stability	Closed-loop stability is not considered	Closed-loop stability is essential.
State Constraints	State constraint enforcement is not considered	State constraint enforcement is essential
Failure tolerance	Failure is necessary for learning (simulation can be used)	Failure cannot be tolerated
Sweet-spot	Accomplishing a task (winning a game)	Keeping a stationary multivariable process within constraints and close to setpoints in the face of measurement noise and process disturbances

The typical goals of RL and MPC algorithms are also quite different. The goal of a RL application is often to complete a task, such as winning a game, where you do not care how the algorithm traverses the state space or how long it takes to win. In the AlphaGo application, for example, the agent does not care how long it takes to win, or how many of its stones get taken. All that matters is to win the game. And the goal of winning does not change during the game. In contrast, an MPC application cares very much about how it traverses the state-space. It is preferable to follow a short, smooth path from the current state to the desired steady-state, and state constraints must not be violated along the way. And the goal for the MPC algorithm can change frequently, in the worst case at every sample, as the operator changes setpoints or constraint limits, and as disturbances perturb the process. However, these distinctions should not be interpreted as fundamental limitations as each method can, in principle, be formulated to address problem type of the other. For example, an RL agent can carry a general stage-wise reward function, which can be used to shape or constrain the transient movement in the state space. And the RL agent can be trained using data that includes many setpoint changes, constraint limit changes, and disturbances. However this additional required training may be very challenging in practice.

For RL the "model-free" label is a bit misleading since these algorithms must still build up models of the value function and/or the policy. The value function for RL is similar to the objective function for MPC with the system model and control policy substituted into it (with a negative sign). So in this respect RL is like MPC where one builds a model of the "closed-loop" objective function rather than of the process itself.

RL learns the value and policy functions primarily through trial and error experiments. If a high-quality simulation is available, which is often the case for games, then one can initialize the value and policy function estimates by performing these experiments on the simulation. In the context of game playing where the environment follows a same set of rules as the agent, one can even train the agent by having it play copies of itself (Silver, et al., 2017). Once the agent is online, it typically spends most of its time exploiting the models to improve the average reward, with the remaining time dedicated to random actions that allow it to update the value and policy function models if necessary. This balance between exploitation and exploration allows the value and policy function models to adapt as the system changes, and is a fundamental aspect of RL technology. So for RL, model prediction error due to time varying nature of the problem is ultimately addressed (feedback) by adapting the value and policy function models.

With MPC the system model is typically developed through some combination of first principles theoretical arguments (e.g. mass and energy balances) and carefully chosen process identification experiments on the real system. It is assumed that the process will not change significantly in the near future so there is no need to update the system model, at least not on a continuous basis. This can be viewed as exploration first (through plant testing), followed by exploitation thereafter (through control with the identified model). Prediction error is integrated away by estimating disturbances that can enter the process through some combination of input, state, or output channels. This would seem to provide an advantage over RL technology for those cases where the core process dynamics do not change significantly and disturbance patterns remain the same.

The RL agent typically executes very quickly online since this involves only a forward run of the policy network. In contrast, MPC may take much longer to execute since it involves solving three optimization problems sequentially (state estimation, steady-state

target calculation, dynamic optimization). This is necessary in MPC because the control problem may change significantly from one time step to the next.

Closed-loop stability is a concept that does not receive much attention in the RL literature, probably because the emphasis is on batch tasks rather than continuous tasks. For MPC applications, closed-loop stability is essential and is now well-characterized following decades of study (Rawlings et al., 2017). Characterizing closed-loop stability for RL algorithms would seem to be a good research opportunity.

State constraint enforcement is something that is critical to MPC applications but does not seem be addressed by RL technology. When operating a fired heater, for example, it is critical to ensure that the tubeskin temperatures in the firebox remain below a critical value. With RL, by contrast, it does not matter where the agent goes on the board as long as it wins the game. As said before, this is not necessarily a fundamental limitation of RL as stage-wise reward or penalty function can be used to shape the transient performance. Working through the details of how to implement state constraints in RL algorithms would seem to be another good research opportunity.

The tolerance for failure in RL is also quite different that for MPC. Since much of RL learning is accomplished by trial and error using random actions, the system must be allowed to fail occasionally, at least during training, where failure is defined as the lowest possible reward. For MPC, failure in this sense cannot be tolerated. For RL this could be mitigated in a process control scenario in three ways. First a high quality simulation could be used initially to train the RL agent. Second, an imminent failure online could be detected through some form of feature engineering, generating an automatic action to restore the system to a better state. Third, the reward function can be designed to reflect transient constraints on the value function or policy learned, but a careful balance between exploration and exploitation should maintained, esp. during the initial learning period.

To summarize this comparison, RL technology appears to be best suited for batch tasks with a simple-fixed goal, carried out on a potentially non-stationary stochastic system, for which a high-quality simulation is available, and for which state and time constraints are not important. In contrast, MPC seems best applied to continuous tasks with complex, time-varying goals, carried out on stationary systems, for which high-quality simulations may not be available, and for which state and time constraints are very important. However, these distinctions are not necessarily fundamental and only reflect the types of applications each has targeted. Even though RL and MPC have been targeted at very different control problems thus far, they are in some ways complementary, and this opens the door for using RL, or at least some of its aspects, for process control applications.

4. Implications for Process Control

In general, a RL agent has the potential to perform any task that requires knowledge and experience gained by interacting with the process. However, as we have seen, this comes at the cost of requiring significant interaction (training) with the process, or at least with a high fidelity process simulation. Nevertheless there are several ways that RL technology may have an impact on the theory and practice of process control. Here we discuss five broad classes of opportunities.

4.1 Directly replace existing process control technology with RL

The most straightforward use of RL in process control involves using it to directly replace existing control technology. On the other hand, since RL technology was developed to

solve a very different class of problems, we view direct replacement to be the least likely path for RL to have an impact on process control. A much more likely way for RL to have an impact will be in combination with conventional control technology such as MPC, or when applied to different, but related problems where MPC would have some difficulty.

4.2 Integrate aspects of RL technology with MPC

Several authors, including some from the process control community, have discussed how RL can be used to complement existing MPC technology. These proposals make use of RL technology in some way to address weaknesses of MPC. Here we summarize three such proposals.

Lee and Wong (2010) point out that MPC has two inherent limitations. First is the high cost of online computation, which scales with the state dimension as well as the horizon length. The second inherent limitation is that the use of an open-loop dynamic optimization means that MPC cannot make use of information about future uncertainty as it is revealed. These limitations are particularly relevant in the case of nonlinear stochastic control problems for which MPC can be highly suboptimal. Lee and Wong propose to mitigate or remove these limitations through the use of Approximate Dynamic Programming (ADP), another name for the class of technology that we have labelled as RL. They propose several possibilities for integration of ADP with MPC. One is to use MPC in the initial stages of simulation sampling in order to determine likely state sequences and state space regions to focus on. A second method is to use the learned cost-to-go (negative of the value function) as the terminal penalty in a MPC formulation in order to reflect the effect of future uncertainties in closed loop and reduce the depth of prediction and optimization needed. A third way is to alternate between the MPC and ADP algorithms, switching to MPC when a new region of the state space is encountered in which the ADP algorithm cannot be trusted.

Morinelly and Ydistie (2016) present an interesting Dual MPC algorithm in which RL is used to introduce the effect of anticipated information. The algorithm is developed for the simple case of a discrete linear system with uncertain dynamics. They demonstrate that the algorithm converges to the optimal linear state feedback policy (Linear Quadratic Regulator) in this case. A simulation example shows good behaviour for the proposed algorithm, which they call RL dual MPC (RLdMPC). This proposal seems quite promising, provided that it can be generalized to full multivariable systems.

Kamthe and Deisenroth (2017) present an algorithm that combines data-efficient RL with probabilistic MPC. Their algorithm improves the data efficiency of RL while simultaneously ensuring the model uncertainty is properly accounted for in the MPC dynamic optimization. They accomplish this by learning a probabilistic transition model using Gaussian Processes (GPs) to incorporate model uncertainties into long-term predictions. This reduces the impact of model errors and results in a RL framework that makes maximum use of available data. Long-term predictions are obtained from the GP model by iterating forward using a deterministic moment-matching Gaussian approximation. Pontryagin's Maximum Principle is then used to solve the MPC dynamic optimization subject to constraints. They show remarkably efficient learning for two standard tasks – the under-actuated cart-pole swing-up and the fully-actuated double-pendulum swing-up - both of which are solved in less than 15 encounters.

We view these suggestions for integrating RL with MPC as very encouraging and worthy of further investigation.

4.3 Use RL to help manage process control systems – Proportional-Integral-Derivative (PID) controller tuning, MPC model gain adjustments, etc.

Process control systems require significant management by operators and control engineers in the form of adjusting limits and tuning parameters as process conditions change. When the feed to a unit is cut by 20%, for example, an experienced control engineer will know which PID and MPC controllers will need to be adjusted and will have a good idea for the types of adjustments that will be required. This type of application seems ideally suited for RL technology, if good process simulators or sufficient amounts of historical data are available.

4.4 Using RL to advise operators during unusual situations (process upsets/start-up/shutdown)

Operating a chemical process is similar in many ways to flying an airplane in that there are long periods of normal, quiet operation punctuated by brief periods of extreme tension when things go wrong. During a process upset, an operator will immediately attempt to bring in additional experienced operators to help diagnose and mitigate the problem. In cases requiring an immediate shutdown, the operator will typically consult a large manual of written instructions that carries with it an implicit assumption of the shutdown scenarios most likely to be encountered. In these situations, RL technology might be particularly helpful as a tool to advise the operator. The RL agent can be trained on the same process simulators that are employed to train the operators, but it can base its recommendations on many thousands of randomly generated scenarios, some of which may not have been anticipated before.

4.5 Use a hierarchical network of RL agents to simplify operation of a chemical plant.

In the not too distant future one can imagine a hierarchical network of RL agents that learn how to perform most of the tasks that human operators currently perform while operating a chemical plant. The hierarchies would reflect the operational priorities of the company operating the plant. These could be, for example:

1. Process Safety
2. Environmental Compliance
3. Reliability
4. Economics

Agents at the same level would cooperate in a distributed manner to optimize the overall goals at that level. Agents at different levels would act according to set priorities. For example, the actions of an economics agent on a particular piece of equipment could be interrupted at any time by a safety agent that has detected a problem and needs to shut down the equipment right away.

There has been significant work in the area of RL agent networks, by Foerster, et al. (2016), for example, but nothing that seems appropriate for off-the-shelf use on this problem.

One should be skeptical, however, of a proposal to replace *all* of the operators in a control room. The development of automation technology such as RL is more likely to lead to a

significant reduction in the need for human intervention, rather than the elimination of all operators in the control room. The march of automation for commercial aircraft has, for example, reduced the number of required cockpit crewmembers from four in the early days to two today. But it is unlikely that passengers will step onto an airliner with no cockpit anytime soon.

5. Future research directions

From our analysis there are several opportunities for process control researchers to have an impact on RL theory. These ideas are aimed augmenting RL technology to make it more appropriate for process control applications:

- Prove algorithmic convergence (value function and policy function approximations converge to optimal value and policy function) for a state-of-the-art RL algorithm such as A3C
- Prove closed-loop stability for the case of a state-of-the-art RL algorithm such as A3C acting on a nonlinear dynamic system
- Augment a state-of-art RL algorithm such as A3C so that it enforces state constraints
- Investigate additional ways to integrate RL and MPC, building on work already published
- Develop robust control technology based on an A3C algorithm in which each parallel environment represents a realization of the system uncertainty.
- Develop technology that allows a hierarchy of prioritized RL agents to accomplish a complex task similar to operation of a chemical plant.

6. Conclusions

In general, a RL agent has the potential to perform any task that requires knowledge and experience gained by interacting directly with the process. Recent technical developments, including deep learning and parallelization, greatly increase the potential scope and capability of RL. Five broad classes of potential process control applications for RL technology have been assessed, and several research directions aimed at improving the relevance of RL for process control applications have been suggested. In summary, we believe that RL technology has the potential to significantly impact both theory and applications in the field of process control.

Acknowledgements

The authors wish to thank Michael Kovalski of the ExxonMobil Technical Computing Company for his assistance in implementing the parallel AC algorithm.

References

A. Barto, R. Sutton, C. Anderson, 1983, Neuronlike elements that can solve difficult learning control problems, IEEE Transactions on Systems, Man, and Cybernetics, 13, 835-846.

C. Bishop, 2006, Pattern Recognition and Machine Learning, Springer.

J. Foerster, Y. Assael, N. Frietas, S. Whiteson, 2016, Learning to Communicate with Deep Multi-Agent Reinforcement Learning, 30th Conference on Neural Information Processing Systems (NIPS 2016)

W. Knight, 2017, 10 Breakthrough Technologies: Reinforcement Learning, MIT Technology Review.

J. Lee, W. Wong, 2010, Approximate dynamic programming approach for process control, Journal of Process Control, 20, 1038-1048.

J. Lee, J. Shin, and M. Reallf, 2017, Machine Learning: Overview of the Recent Progress and Implications for the Process Systems Engineering Field, FOCAPO/CPC 2017.

S. Levine, 2017, CS294 Deep Reinforcement Learning, Lecture 3, YouTube.

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, 2016, Continuous Control with Deep Reinforcment Learning, International Conference on Learning Representations (ICLR) 2016

S. Manthe, P. Deisenroth, 2017, Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control, arXiv:1706.06491v1

J. Morinelly, E. Ydstie, 2016, Dual MPC with Reinforcement Learning, IFAC Papers Online j.ifacol.2016.07.276

V. Mnih, A. Badia, M. Mirza, A. Graves, T. Harley, T. Lillicrap, D. Silver, K. Kavukcuoglu, 2016, Asynchronous Methods for Deep Reinforcement Learning, Proceedings of the 33rd International Conference on Machine Learning, New /York, NY.

S. Qin, T. Badgwell, 2003, A Survey of Industrial Model Predictive Control Technology, Control Engineering Practice, 11/7, 733-764.

J. Rawlings, D. Mayne, M. Diehl, 2017, Model Predictive Control: Theory, Computation, and Design, Knob Hill Publishing.

D. Silver, A, Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lancot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, 2016, Mastering the game of Go with deep neural networks and tree search, Nature, 529, 484-489.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassibis, 2017, Mastering the game of Go without human knowledge, Nature, 550, 354-359.

R. Sutton, A. Barto, 2016, Reinforcement Learning: An Introduction, Second Edition, in progress (online).

A. Turing, 1950, Computing machinery and intelligence, Mind, 59, 433-460.

I. Witten, 1977, An adaptive optimal controller for discrete-time Markov environments, Information and Control, 34, 286-295.