Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 52-14 (2019) 111–116

Reinforcement Learning and Deep Neural Networks for PI Controller Tuning

William J. Shipman*. Loutjie C. Coetzee*

**MINTEK, Johannesburg, South Africa*
(Tel: +27-11-709-4980; e-mail: {williams;loutjiec}@mintek.co.za).

Abstract: Reinforcement Learning, using deep neural networks, has recently gained prominence owing to its ability to train autonomous agents that have defeated human players in various complex games. Here, Reinforcement Learning is applied to the challenge of automatically tuning a proportional-integral controller, given only the process variable, set-point, manipulated variable and prior controller gains. The training considers random changes in plant dynamics, disturbances and measurement noise. Two training procedures were tested in this work, one that built up the difficulty of the simulation over time, and another that used the full complexity of the simulation throughout the training. The results show that building up the difficulty of the simulation by introducing greater degrees of randomness as the training progresses, produces an agent that is better able to tune the controller in question. Additional experience gathered in completing this work is also discussed to enable the reader to avoid some of the challenges encountered.

Keywords: artificial intelligence, autotuners, neural networks, SISO, proportional plus integral controllers, reinforcement learning.

1 INTRODUCTION

There has been renewed interested in the field of Artificial Intelligence lately, with the focus on the ability of neural networks to learn a variety of tasks, using experience obtained from previous attempts to perform the tasks, rather than from repetitive examples. This principle is referred to as Reinforcement Learning (RL), and a key publication illustrating the concept was the paper by Mnih et al. (2015).

RL also has application in the process control industry. Strategies to efficiently tune simple controllers, e.g. a PID, can be taught to junior engineers, and software can be developed to perform these tasks using algorithms or rules. More complex Advanced Process Control (APC) systems are much more difficult to tune, owing to the large number of interacting tuning parameters. An experienced engineer should, however, be able to consider all these factors and find a suitable set of parameters for such a controller – an ability that is similar to that demonstrated for deep neural networks. Controllers require regular retuning to handle process variations, e.g. changes in ore properties or variations in equipment behaviour, e.g. owing to maintenance.

Neural networks and reinforcement learning have been applied to numerous process control problems, with varying levels of success, since the late 1980s (Chovan et al., 1994; Conradie and Aldrich, 2010; Okaya and Inoue, 1992). In general, these works have attempted to learn the dynamics of the processes under control, followed by learning the optimal control policy using the learned model. Recent research has developed improved RL methods for complex control tasks (Lillicrap et al., 2016; Schulman et al., 2015a) and has increased the depth and complexity of the neural networks in use. Lee et al. (2018) summarise numerous improvements in neural network architecture and training methods.

The aim of this investigation was to experiment with deep neural networks and RL to evaluate the suitability of applying this technology to controller tuning in the process control industry. The problem tackled in this work is the tuning of a PI controller. While this appears to be a comparatively simple problem, tackling it has provided insight into the effect of the training procedure on the ability of the RL agent to handle variations in plant parameters. This work was not done with the intention of beating existing PI auto-tuning algorithms, hence they do not form a part of the tests conducted, although a short comparison against an offline design method, in which the plant model must be known, is given.

In contrast to related works applying neural networks or RL to process control, the RL agents trained here are exposed to a variety of plant models during training. In effect, these agents are being trained to tune any PI controller applied to any plant with a first-order response that has a gain and time constant within certain ranges given below. This paper covers the process used to train an agent to tune a PI controller, the performance of the system and the lessons learned. The challenge lies both in selecting the most appropriate reward function and in developing a suitable training procedure. This work will demonstrate two training procedures and their impact on the overall performance of the closed-loop system. The author's experience with different reward functions will also form part of the discussion.

2405-8963
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2019.09.173

2 LITERATURE REVIEW

2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) has recently gained prominence owing to significant successes in challenging games, such as the Atari games (Mnih et al., 2015), as well as in challenging control problems, such as locomotion and the classic cart-pole problem (Lillicrap et al., 2016; Schulman et al., 2015a). In RL, an autonomous agent interacts with its environment, observing the state, $x(t)$, of the environment at time t, performing an action, $a(t)$, and receiving a reward, $r(t)$. The method by which actions are chosen is called a policy, indicated by $\mu_\theta(x(t))$.

Recently, Actor-critic algorithms have been preferred for control tasks owing to their ability to handle continuous action spaces. Actor-critic methods consist of an actor that represents the policy, generating an action based on the observed environment, and a critic that estimates the discounted reward associated with performing that action while in the observed state, or just the maximum possible discounted reward achievable from the current state, depending on which RL algorithm is used. In DRL, both the actor and the critic are deep neural networks. These deep neural networks hold the promise of being able to learn complex patterns in challenging environments.

Two popular actor-critic strategies are Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) and Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016). A variant known as Advantage Actor-Critic (A2C) was developed in open-source software following the publication of A3C. A2C, like A3C, performs asynchronous simulations. However, A2C updates the actor and critic neural networks only after every asynchronous batch of simulations has finished. The simulation batch size is a tuneable hyper-parameter.

Wang et al. (2007) proposed an actor-critic method for tuning a PID controller and demonstrated its performance on a simulated non-linear system. The actor and critic made use of a neural network with a single hidden radial basis function layer. Wang et al. showed that the PID tuned by this method out-performed a PID tuned using the Zeigler-Nichols method. Arzaghi-Haris (2008) subsequently used this reinforcement learning method and network architecture to tune a PID to regulate wind turbine rotor speed to the optimal speed in order to maximise the power generated. The existing literature does not tackle the challenge of adjusting the controller gains online in response to changes to the plant model, as their aim was to determine optimal controller gains for a single plant model.

2.2 Reward functions

The reward indicates how good an action is, with several possibilities existing that are applicable to process control tasks. Integral Absolute Error (IAE) is one of the most common ways of comparing the performance of two controllers, while other options are to consider stability criteria. Reward functions encountered in the literature are discussed below.

2.2.1 Sparse rewards

Discrete rewards assign a fixed reward or penalty to an action, e.g. $+1$ for achieving a desired goal and -1 for failure. Different degrees of success are also possible. For example, Wang et al. (2007) assigns a penalty of -0.5 if the controlled variable (CV) is more than a desired amount ε from the set-point variable (SV) and a penalty of -0.5 if the error is increasing. These two penalties are weighted and summed to obtain the final reward, leading to four possible values of the reward. Binary rewards have two values, indicating success and failure, as used by Berenji and Khedkar (1992). Collectively, these are termed sparse rewards as the reward signal will remain unchanged for a substantial period of time, encoding no information about the progress being made. These rewards are common in game-playing environments, e.g. Atari games, as the agent must complete a task in the game before receiving a reward. The challenge posed by such rewards is in allocating the reward to the correct previous action.

2.2.2 Continuous rewards

In contrast to sparse rewards, continuous rewards combine one or more smooth functions to penalise or reward the action taken. IAE is one possible penalty. The locomotion tasks tackled by Schulman et al. (2015b) reward forward velocity while penalising the amount of torque applied to each joint to induce movement, as well as penalising any impact, i.e. falling down or hitting walls.

Deisenroth (2012, pp. 53–54) used a saturating cost function $1 - \exp\left(-\frac{1}{2a^2}(\mathbf{x} - \mathbf{x}_{\mathbf{target}})^T \mathbf{T}^{-1}(\mathbf{x} - \mathbf{x}_{\mathbf{target}})\right)$, where $\mathbf{T}^{-1}$ is a diagonal matrix with entries either 1 or 0, $\mathbf{x}$ is a vector of states and $\mathbf{x}_{\mathbf{target}}$ is a vector of target states, or SVs. This cost function can be converted to a reward by subtracting it from 1, so that increasing the distance from the target state yields positive rewards tending towards 0 in the limit, while perfect performance, i.e. zero distance, equates to a score of 1. Hall et al. (2011) note that this function encodes the concept of being far from the target state, without leading to large penalties when being far from the target state. Large gradients in the reward function can lead to unstable training of the RL agent (Deisenroth, 2012, p. 58).

3 EXPERIMENTAL METHOD

A first-order system with a PI controller was simulated. The gain, K, time constant, τ, and time delay, T, of the system were randomly varied in the ranges 0.5 to 15.5, 10 to 45 and 1 to 4 seconds respectively to simulate changes in process dynamics. The discrete first order model with sampling period, T_s, state, $x(t)$, input after actuator dynamics, $u_{act}(t)$, and disturbance, d, is as follows:

$$\alpha = \exp(-T_s/\tau) \tag{1}$$

$$x(t + T_s) = \alpha x(t) + (1 - \alpha)K[u_{act}(t - T_s) + d] \tag{2}$$

$$y(t) = x(t) + \mathcal{N}(0, 10^2) \tag{3}$$

Manipulated variable (MV) non-linearity was modelled by soft limiting the MV, using the following equation: $u_{act}(t) = 100\tanh(u(t)/100)$. The PI controller determines the value of u as shown in (4). The CV measurement, including Gaussian white noise, is $y(t)$. The disturbance, d, is modelled as a sequence of random step changes, where the step values are drawn from a uniformly distributed random number between -10 and 10. The probability of the disturbance undergoing a step change at any time in a simulation was fixed at 10^{-3}. The sampling period T_s is one second. The error between the SV and CV is $\delta(t) = SV(t) - y(t)$. The discrete PI controller model is as follows:

$$u(t) = u(t - T_s) + K_c\left[\left(1 + \frac{T_s}{T_i}\right)\delta(t) - \delta(t - T_s)\right] \quad (4)$$

The gain of the controller, K_c, and the integral time, T_i, were constrained to the ranges 0.005 to 10 and 0.5 to 50 respectively. K_c and T_i are chosen by the RL agent to maximise the discounted future reward obtained from the environment, i.e. the overall performance of the closed-loop system. The outputs of the actor network of the RL agent ranged from -1 to $+1$, owing to the use of the tanh activation function on the output layer of the neural network. These outputs were linearly scaled to the ranges shown above for K_c and T_i.

The A2C implementation in the OpenAI Baselines library (Dhariwal et al., 2017) was used in this work, owing to its ability to use multiple cores within a CPU to reduce the time required for training. The original A2C implementation was designed for discrete systems. It was adapted to systems with continuous action spaces by following the method described by Mnih et al. (2016) for A3C, viz. each action is represented by a mean value and a variance, instead of the actor generating a probability for taking an action. Mnih et al. (2016, p. 4) provide the policy gradient function used in this work. L2 regularization of the weights of both the actor and critic networks, has been included. Development was done in Python 3.6 using the Anaconda Python distribution

The overall aim is to produce a closed-loop system that responds rapidly to changes in system dynamics, while achieving good long-term performance, quantified by IAE in the experiments that follow. Deisenroth's saturating reward function (Deisenroth, 2012) was chosen for the reward signal to the RL agent, with $a^2 = 0.02 \times SV$ and a penalty for MV oscillations. The MV penalty is calculated by taking the gradient of the MV by using a second-order central finite difference, determining the median of the absolute values of the gradients within the observation period, dividing by the maximum MV limit, yielding a value Δ, and applying the following equation $r_{MV} = e^{-\Delta/0.02}$. The combined reward is

$$r(t) = 0.25\exp\left(\frac{-\big(SV(t) - y(t)\big)^2}{2a^2}\right) + 0.75r_{MV} \quad (5)$$

Intuitively, this reward function should result in the CV remaining within 2 % of the SV, subject to measurement noise, without overly incentivising aggressive controller gains. The MV reward should result in MV movements being minimised without penalising temporary spikes in the MV, which are needed to respond to SV changes.

The input to the RL agent is an $n \times 5$ matrix containing $[y(t) \quad SV(t) \quad u(t) \quad K_c(t) \quad T_i(t)]$ for all n time points from $t = t_0$ to $t = t_0 + T_{obs}$, where t_0 is the starting time of the observation, $1 \le T_{obs} \le T_{total}$ is the length of time used to construct an observation and T_{total} is the length of a single simulation. The RL agent is sampled at a different sampling frequency to that of the plant, labelled T_{agent}. The reward is calculated at each sampling period of the simulation and averaged over the time period T_{agent} in order to create a reward between 0 and 1. In this work, $T_{agent} = 60$, $T_{obs} = 120$ and $T_{total} = 7200$. All times are given in seconds. The neural network structure and A2C parameters are provided in Appendix A.

Initial experimentation revealed that training an agent from scratch to handle all of the randomness available in this simulated environment is challenging. Therefore, the training process was divided into stages, where each stage adds a new aspect of randomness to the simulation. The stages are as follows:

- S1: The plant model parameters are randomly selected at the start of training and new parameter values are randomly selected at the start of every 100th simulation. No noise is added to the CV and no disturbance is included. The SV is a random sequence of step changes.
- S2: The random MV disturbance is introduced to S1.
- S3: Random noise is added to the CV in S2.
- S4: The random plant changes take place at the start of every simulation, instead of every 100th simulation.

The results show the simulated performance of the RL agent trained by proceeding from S1 to S2 to S3 and finally S4, named RL agent 1, as well as the RL agent trained by proceeding directly to S4, named RL agent 2. RL agent 1 was trained for 40×10^6 steps (i.e. observations) on stage S1 and 20×10^6 steps on each subsequent stage. The training for S1 was conducted in two parts of 20×10^6 steps each, with the neural network weights learned in the first stage being reused in the second stage. The training simulations presented in part 1 were the same as in part 2 as the same seed for the pseudo-random number generator was used. RL agent 2 was trained for 100×10^6 steps on S4 only, hence it undergoes the same number of training steps as RL agent 1.

4 RESULTS

Fig. 1 shows the improvement in the discounted future reward (discounting factor $\gamma = 0.99$) obtained per episode vs. the number of training steps when training RL agent 1. The reward achieved by RL agent 2 during training, is shown in Fig. 2. Both figures include a 10 sample moving average to show the overall trend. The noise in the reward values indicates that the RL algorithm has not finished exploring the observation and action spaces. The discounted future rewards achieved at the

end of stage S4, averaged over the last 10 steps, are 53.33 and 44.39 for RL agents 1 and 2 respectively.

Following training, RL agents 1 and 2 were tested in a simulation, the results of which are shown in Table 1 and Figures 3 and 4. Table 1 shows the total reward without discount. The first two minutes of the simulation use a randomly selected K_c and T_i, causing the oscillatory behaviour observed in that time. The CVs have been rendered partially transparent in order to show the response of the underlying state variable in each simulation. This system has $K_p = 0.8$, $\tau = 14.26$ and $T = 2$. The proportional gain and integral time do fluctuate, depending on the SV that is chosen, and are inversely proportional for RL agent 1. The maximum reward corresponds to T_{total}/T_{obs} as this is the number of RL agent steps in a simulation multiplied by the maximum reward of 1.

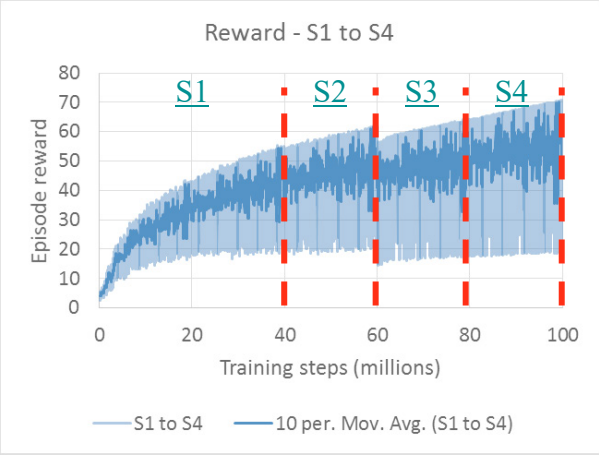

Fig. 1: Reward achieved during training RL agent 1.

Fig. 2: Reward achieved during training RL agent 2.

Figures 5 and 6 show the improvement in reward and IAE achieved by RL agent 1 over RL agent 2 in a suite of 20 test cases. No correlation was observed with the time constant of the plant. However, as shown in Figures 5 and 6, RL agent 1 outperforms RL agent 2 on plants with gains lower than 5. At higher gains, it is not clear which agent provides a better result. The improvement in IAE for low gain plants is particularly prominent.

Table 1: Simulation performance of the RL agents.

	RL agent 1	RL agent 2
Reward	86.46	86.54
IAE	4.37	6.06

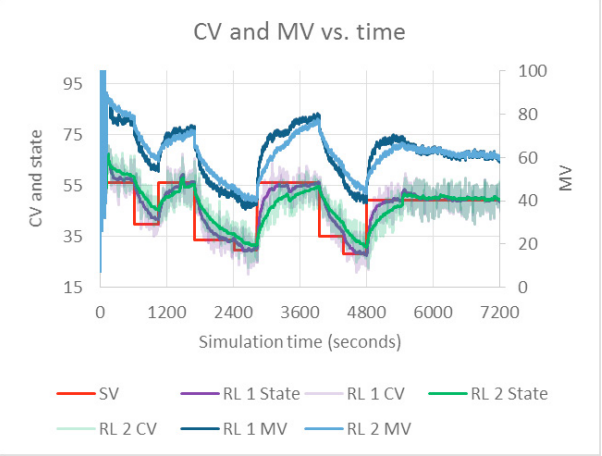

Fig. 3: Closed-loop system response when using RL agent 1 (RL 1) and RL agent 2 (RL 2).

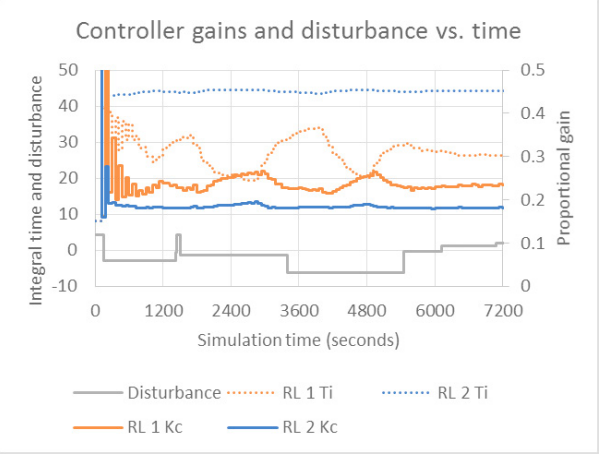

Fig. 4: Controller gains during simulation, using RL agent 1 (RL 1) and RL agent 2 (RL 2).

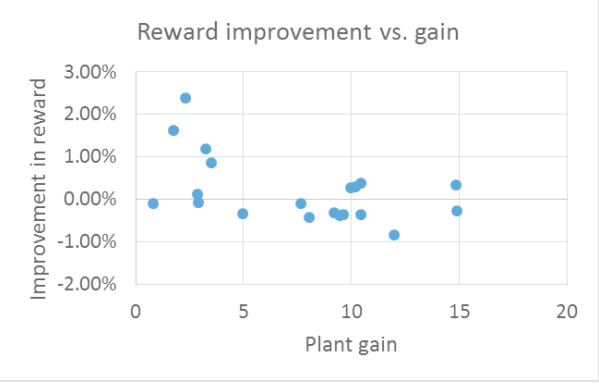

Fig. 5: Improvement in reward achieved by RL agent 1 over RL agent 2 in test simulations.

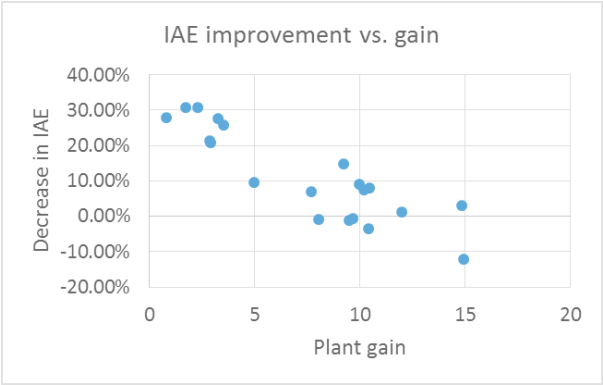

Fig. 6: Improvement in IAE achieved by RL agent 1 over RL agent 2 in test simulations.

5 DISCUSSION

5.1 Training methodology

RL agent 1 demonstrated a higher average reward at the end of its training compared to RL agent 2. The simulation results show that RL agent 1 has learned to handle plants with low and high gains well. RL agent 2 has learned to handle high-gain plants well, but is outperformed by RL agent 1 on low-gain plants. Reviewing several of the simulations showed that RL agent 1 has learned to adjust the controller gains on both low and high gain plants, e.g. decreasing integral time to produce a more aggressive response and increasing it when low MV movement and/or stability is required. RL agent 2 has only learned this behaviour on high-gain plants. However, RL agent 2 has learned to briefly double or triple the proportional gain on every SV change in order to achieve a rapid response, reducing the gain a minute later. Both agents achieved similar rewards in the simulation presented in Figures 3 and 4, despite significant differences in IAE, as agent 1 created larger changes in the MV, lowering its reward, compared to agent 2.

The authors hypothesise that frequent random changes in the plant model prevent the agent from learning to use changes in the proportional gain and integral time over short time periods to maximise reward. The duration of a simulation is unlikely to be sufficient for the agent to learn to optimally control the plant presented in that simulation. Allowing 100 simulations, before changing the plant model, may give enough time to learn the optimal behaviour for each plant. High-gain plants are more likely to exhibit a response that reaches SV within the two minute observation window given to the RL agents, making them easier to learn, as reward assignment is easier.

5.2 Reward function choice

The relationship seen between the SV and controller gains in Fig. 3 results from the definition of the reward in terms of a percentage of the current SV value. A larger SV reduces the penalty assigned to the error between the SV and CV, allowing the RL agent to prioritise reducing MV movement.

Prior experimentation considered the use of various reward functions. Negative reward functions, where the maximum reward is zero, are more natural to use as they can be obtained easily from any cost function, e.g. IAE. However, the A2C implementation used here was unable to function with a negative reward, as it would minimise the reward in training, instead of maximising it. The cause remains unclear. This behaviour was also observed when testing the DDPG implementation in Keras-rl (Plappert, 2016) with the same reward and simulation.

Rewards based on rise time and decay ratio were attempted. However, the presence of disturbances and noise makes their calculation challenging, hence they could not be used reliably. A reward based directly on IAE results in a highly oscillatory MV. As the results show, the reward function requires additional improvements as the difference in the reward achieved by RL agents 1 and 2 is small compared to the improvement in set point tracking achieved by RL agent 1 compared agent 2. In particular, the MV penalty term may require revising to use something other than median gradient.

5.3 Comparison against other common PI tuning formulae

Chen and Seborg (2002) show formulae for using direct synthesis (DS) to obtain K_c and T_i that result in a desired closed-loop time constant for the system. For this comparison, the close-loop time constant was chosen to be 100 owing to the level of measurement noise. This method is compared to the current work in Table 2, showing that RL agent 1 achieves a similar level of performance to DS without knowing the plant model.

Table 2: Reward and IAE achieved by various PI parameter selection formulae.

Method	K_c	T_i	Reward	IAE
DS	0.17	14.26	89.00	3.96
RL 1	-	-	86.46	4.37
RL 2	-	-	86.54	6.06

6 CONCLUSION

This work has investigated the use of Deep Reinforcement Learning for automatically tuning a PI controller under a wide range of plant models. Two approaches to the training of a Deep Reinforcement Learning agent have been presented in this work. The first builds up the complexity of the training process, starting first with model changes on every 100^{th} simulation, followed by adding a random disturbance and random noise. Once the agent has learned to tune a PI controller under these conditions, the complexity of the simulation is increased one final time by allowing the plant model to change on every simulation. This was compared to training the agent on the full complexity simulation from the beginning. The results showed that building up the complexity of the simulation allows the agent to learn suitable behaviours more rapidly, outperforming the agent trained entirely on the full complexity simulation, even though both agents underwent the same amount of training in total (100×10^6 steps). Challenges encountered in the development of the reward function used in this work have also been discussed.

REFERENCES

Arzaghi-Haris, D., 2008. Adaptive PID Controller Based on Reinforcement Learning for Wind Turbine Control. In: Recent Advances in Environment, Ecosystems and Development: Proceedings of the 6th WSEAS International Conference on Environment, Ecosystems and Development (EED'08). Cairo, Egypt, p. 7.

Berenji, H.R., Khedkar, P., 1992. Learning and tuning fuzzy logic controllers through reinforcements. IEEE Trans. Neural Netw. 3, 724–740.

Chen, D., Seborg, D.E., 2002. PI/PID Controller Design Based on Direct Synthesis and Disturbance Rejection. Ind. Eng. Chem. Res. 41, 4807–4822.

Chovan, T., Catfolis, T., Meert, K., 1994. Process Control using Recurrent Neural Networks. IFAC Proc. Vol., 2nd IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, Lund, Sweden, August 10-12, 1994 27, 135–140.

Conradie, A. v. E., Aldrich, C., 2010. Neurocontrol of a multi-effect batch distillation pilot plant based on evolutionary reinforcement learning. Chem. Eng. Sci. 65, 1627–1643.

Deisenroth, M.P., 2012. Efficient Reinforcement Learning using Gaussian Processes. Karlsruhe Institute of Technology.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P., 2017. OpenAI Baselines. OpenAI.

Hall, J., Rasmussen, C.E., Maciejowski, J., 2011. Reinforcement learning with reference tracking control in continuous state spaces. In: IEEE Conference on Decision and Control and European Control Conference. Presented at the 2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC 2011), IEEE, Orlando, FL, USA, pp. 6019–6024.

Lee, J.H., Shin, J., Realff, M.J., 2018. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Comput. Chem. Eng., FOCAPO/CPC 2017 114, 111–121.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2016. Continuous control with deep reinforcement learning. In: Proceedings of the 4th International Conference on Learning Representations. San Juan, Puerto Rico.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Okaya, K., Inoue, T., 1992. Mineral Process Control by Neural Network. IFAC Proc. Vol., IFAC Workshop on Expert Systems in Mineral and Metal Processing, Espoo, Finland, 26-28 August 1992 25, 167–171.

Plappert, M., 2016. keras-rl. Keras-RL.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2015a. Trust Region Policy Optimization.

Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., 2015b. High-Dimensional Continuous Control Using Generalized Advantage Estimation.

Wang, X., Cheng, Y., Sun, W., 2007. A Proposal of Adaptive PID Controller Based on Reinforcement Learning. J. China Univ. Min. Technol. 17, 40–44.

Appendix A. REINFORCEMENT LEARNING ALGORITHM PARAMETERS AND DEEP NEURAL NETWORK ARCHITECTURE

The parameters of the A2C algorithm are given in Table 3. The actor deep neural network consists of an input layer with dimensions 120x5 that is flattened into a 600 element vector. This input goes to a stack of three densely connected layers with 32 neurons each, the hidden layers of the network. The output layer consists of two neurons densely connected to the last hidden layer. The output is K_c and T_i scaled to the range $[-1; 1]$. All of the layers in the actor network use tanh as their activation function.

The critic network follows a similar architecture as the actor network. The main differences are:

- The actor network outputs are concatenated to the flattened observation vector, giving a 602-element vector.
- The output layer of the critic network has a single neuron and uses a linear activation function.

The weights of the hidden layers were not shared between the actor and critic networks.

Table 3: A2C parameters used.

Parameter	Value
Actor learning rate	10^{-5}
Critic learning rate	10^{-4}
Learning rate schedule	Constant
L2 regularisation weight	10^{-3}
A2C gamma	0.99
A2C alpha	0.99
A2C epsilon	10^{-5}
Value loss coefficient	0.05
Entropy coefficient	0.01
Maximum norm of gradient	0.5