Aalto University
School of Electrical
Engineering

Master's programme in Automation and Electrical Engineering

Development of a Reinforcement Learning Agent for Advanced Process Control in Dynamic Simulations

Julius Lang

Master's Thesis
2023

Aalto University
School of Electrical
Engineering

Author Julius Lang

Title Development of a Reinforcement Learning Agent for Advanced Process Control in Dynamic Simulations

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems

Supervisor Prof. Simo Särkkä

Advisor D.Sc. (Tech.) Cesar Araujo

Collaborative partner Metso Oyj

Date 26 October 2023 **Number of pages** 75 **Language** English

Abstract

Established advanced process control methods, such as model predictive control, remain the industry standard for optimal control of industrial chemical processes. Traditional approaches for optimising operational indices often rely on first principles models of complex process dynamics, which are strongly affected by stochastic process disturbances and require expensive online computation. The fourth industrial revolution has introduced a new generation of data-based approaches to process optimisation, including reinforcement learning as a potential tool for advanced process control. The aim of this thesis is to develop a reinforcement learning based advanced process control solution to optimally control chemical processes in simulated environments. Firstly, an appropriate reinforcement learning framework is established to facilitate the development of a solution. Subsequently, a reinforcement learning agent for advanced process control is developed over iterative proof-of-concept stages while gradually increasing the problem complexity. The results of the final case study demonstrate that the developed reinforcement learning solution is able to handle the optimal control of an industrial process simulation. Specifically, a control policy trained by the developed algorithm learns the simultaneous regulation of multiple interacting process variables through trial-and-error, accounting for stochastic system disturbances robustly. The experimental results validate reinforcement learning as a potential approach for the optimal control of industrial processes based on aptly defined reward functions governing the control objective. The findings motivate future research into extended applications of reinforcement learning based process optimisation in a burgeoning domain of control engineering.

Keywords reinforcement learning, advanced process control, optimal control, deep learning, actor-critic algorithm, stochastic policy gradient, deep determinsitic policy gradient, process simulation

Acknowledgements

I would like to express my sincere gratitude to Metso and Christian Binder for giving me the opportunity to pursue an academically and professionally stimulating master's thesis topic. Open discussions around potential avenues of research and development have allowed me to combine my academic backgrounds in control engineering and machine learning in an interdisciplinary project. This undertaking has significantly furthered my academic knowledge in the respective fields and will hopefully serve as a foundation for further development and business opportunities at Metso.

I am deeply appreciative of my thesis advisor, Cesar Araujo, for providing tireless support, mentorship, and invaluable insights throughout the entirety of the thesis project. Their expertise and perspectives have been especially instrumental in shaping the practical aspects of the work, as well as continually motivating incremental solution improvements.

I extend my gratitude to my supervising professor, Simo Särkkä, for their academic guidance and meticulous feedback over the writing process. The inputs received in our exchanges have made a substantial contribution to the quality of the thesis.

Lastly, I wish to acknowledge my friends and family, who have been a source of energy throughout my academic journey leading up to the completion of this thesis. Special recognition goes to the amazing people of Skipoli. The shared passion for winter sports and the bonds we have formed on and off the slopes have provided me with much-needed escape and a sense of community. Your impact may often go unthanked, but never unnoticed.

Helsinki, 26 October 2023

Julius A. Lang

Contents

Symbols and abbreviations

Nomenclature

a_i	Output of neuron i in an ANN
a_t	Action at time step t in MDP
$\arg\max_{\pi}$	Policy arguments of the maxima of some function
A	Action space in MDP
b_j	Bias of neuron j in an ANN
$E[A]$	Expected value of A
$\overline{E}$	Mean empirical error of ANN output
$g(x)$	Activation function
$J(\pi)$	Expected cumulative reward (return) over a trajectory using policy π
L	Loss function used to update neural networks
$\max_{\pi}$	Maximum of some function given optimal policy
P_a	Probability of state transition given action a in MDP
$P(x\|y)$	Probability of x, given y
$Q(s,a)$	Action-value: expected cumulative reward following action a in state s
$Q^*(s,a)$	Optimal action-value obtained using the optimal policy
r_t	Reward received by agent at time step t in MDP
R	Reward
$R(\tau)$	Sum of rewards over trajectory τ, return of trajectory
$s_0 \sim \rho_0(\cdot)$	s_0 is randomly sampled from distribution ρ_0
s_t	State at time step t in MDP
S	State space in MDP
$V(s)$	State-value: expected cumulative reward following state s
$V^*(s)$	Optimal state-value obtained using the optimal policy
$w_{i,j}$	Weight of edge from neuron i to j
α	Learning rate of ANN
γ	Discount factor or rewards over a trajectory
ε	Proportion of random actions taken using an ε-greedy policy
ϵ	Hyperparameter defining the bounds of the clipped ratio function in the PPO algorithm, Hyperparameter defining the action bounds in the DDPG algorithm
θ	Parameter set in a parameterised RL agent policy
μ	Mean
μ_θ	Deterministic RL agent policy parameterised with θ
π_θ	Stochastic RL agent policy parameterised with θ
$\pi(a_t\|s_t)$	Probability of action a_t, given state s_t, following policy π
π^*	Optimal stochastic RL agent policy
ρ_0	Start state distribution
σ	Standard deviation
τ	Trajectory in MDP
ϕ	Parameter set of Q-function in DDPG algorithm

Operators

$\sim$	Is distributed as
$\in$	Belongs to
$\leftarrow$	Update to the value on the left side of operator
∇_θ	Gradient with respect to parameters θ

Abbreviations

ADP	Approximate dynamic programming
AI	Artificial intelligence
ANN	Artificial neural network
APC	Advanced process control
API	Application programming interface
CCPO	Chance constrained policy optimisation
CIL	Carbon-in-leach
CPS	Cyber-physical system
CPU	Central processing unit
CSTR	Continuous stirred tank reactor
DCS	Distributed control system
DDPG	Deep deterministic policy gradient
DFQL	Dynamic fuzzy Q-learning
DNN	Deep neural network
DO	Dissolved oxygen
DPG	Deterministic policy gradient
DQN	Deep Q-network
DRL	Deep reinforcement learning
EPPO	Ensemble proximal policy optimisation
FIS	Fuzzy inference system
GPU	Graphics processing unit
IDE	Integrated development environment
KPI	Key performance indicator
LQ	Linear quadratic (problem)
LQR	Linear quadratic regulator
MDP	Markov decision process
MFPC	Model-free predictive control
MIMO	Multiple input, multiple output
ML	Machine learning
MPC	Model predictive control
NMPC	Non-linear MPC
ONNX	Open neural network exchange
PID	Proportional-integral-derivative (controller)
RL	Reinforcement learning
POC	Proof-of-concept
PPO	Proximal policy approximation
REINFORCE	Reward increment = nonnegative factor × offset reinforcement × characteristic eligibility
RNN	Recurrent neural network
SISO	Single input, single output
SLMPC	Successive linearisation based MPC
SNN	Single-layer neural network

1 Introduction

The process industry has been experiencing rapid digitalisation as part of the fourth industrial revolution. Whereas mechanisation, electrification and automation characterised the first three industrial revolutions respectively, the ongoing fourth industrial revolution involves the development of digital and intelligent operations through Cyber-Physical Systems (CPS) (Zhou et al., 2015). The progression and characteristics of the four industrial revolutions can be seen in Figure 1. A growing number of capitalisable opportunities have emerged in the space of digital twins and process simulation as a result of the current transformation (Jiang et al., 2021). Combined with the ability to collect and pipeline large amounts of data from both physical plants and computer simulations, engineers are presented with a new set of tools to improve key performance indicators (KPIs) in process operation (Mowbray et al., 2022). Simulations of real industrial processes and data logging allow processes to be improved through machine learning (ML) methods (Sachio et al., 2022). However, there is currently limited literature on data-based approaches for optimisation of distributed control systems (DCS) in the process industry.

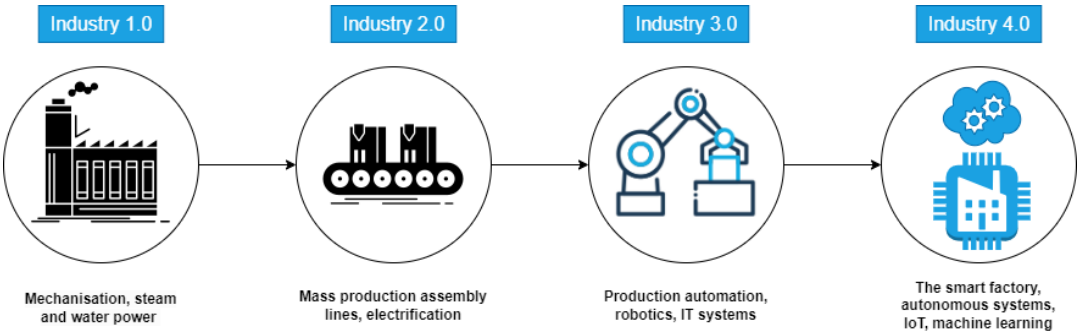

Figure 1: The four industrial revolutions.

Due to the natural incentives for optimising industrial processes in terms of their operational indices, such as product quality, efficiency, and resource consumption (Chai et al., 2014), solutions to optimal control problems have been investigated for decades (Chai et al., 2014; Petsagkourakis et al., 2020). Higher level control solutions designed to optimise process operation at a unit function level are collectively known as advanced process control (APC) (Nicolae et al., 2019). The current industry benchmark in APC is model predictive control (MPC) (Mowbray et al., 2022). MPC relies on dynamic system modelling and optimisation to generate future predictions on system state and impose current control actions accordingly (Petsagkourakis et al., 2020). This already proven technology has become a reliable and widely implemented tool in manufacturing and process industries. However, the reliance on a detailed system model, feed forward control, and online optimisation limits the potential of MPC in its current use as well as its adaptability to be extended to more abstract KPI optimisation tasks over a wider scope (Mowbray et al., 2022).

Proposed solutions to extend the capabilities of current APC techniques incorporate data-driven approaches (Mowbray et al., 2021a), which strive to leverage

available process data and Industry 4.0 advances as a decision-making tool in process control. One such approach is reinforcement learning (RL) (Görges, 2017). RL is a machine learning framework which attempts to solve a sequential decision-making problem through trial-and-error, utilising a reward system based on feedback from the environment. The framework iteratively improves on a control policy, which decides on actions to perform in each system state (Sutton and Barto, 2020). The potential of RL in optimal process control lies in its non-reliance on a known dynamic system model, and its ability to handle non-linear stochastic control problems with reduced online computational complexity (Kim et al., 2020). However, issues hindering the implementation of RL in industrial process control include the adaptability of RL algorithms to continuous and multi-dimensional action (control input) spaces (Lillicrap et al., 2016), the safe deployment of RL controllers in physical plants, and the inherent data-hungry nature of solutions (Mowbray et al., 2021b). Nevertheless, recent developments in policy gradient (Williams, 1992; Duan et al., 2016) and actor-critic (Fujimoto et al., 2018; Demidov and Al-Refai, 2021) RL algorithms have facilitated successful experimental solutions to optimal control in chemical processes characterised by continuous action spaces (Petsagkourakis et al., 2020; Sachio et al., 2022). The problem of data-inefficiency has been recently addressed with novel approaches such as apprenticeship learning (Abbeel and Ng, 2004), reducing the training iterations and data volume needed for convergence to an optimal control policy (Mowbray et al., 2021b). These developments in RL algorithms and implementations for continuous control problems present an interesting avenue for applications in chemical processes containing complex, stochastic, and continuous process state and control action spaces.

A fundamental practical challenge is the prototyping and physical development of RL-based APC for chemical processes. Naturally, an immature and unproven technology relying on trial-and-error optimisation cannot be directly deployed on an industrial scale. Hence, process simulation has the potential to form an integral component of RL-based APC development (Mowbray et al., 2022). Metso Oyj possesses a powerful proprietary chemical process simulation software known as HSC Sim, which can conceivably be used as a testbed for RL-based APC development.

To extend the capabilities of advanced process control with RL beyond current MPC solutions, the technical challenge of compromising complex, continuous and stochastic action space handling with data-efficiency and reliability needs to be addressed (Mowbray et al., 2021b, 2022). There is a strong business case for developing solutions in this domain and Metso is interested in exploring efficiencies and improvements in hydro-metallurgical processes through this approach, while leveraging proprietary process simulation technology.

The aim of this thesis is to develop an RL-based APC agent capable of optimally controlling simulated chemical processes with multi-dimensional control objectives to achieve favourable operational indices. A chosen RL algorithm from literature is implemented, adapted to the relevant application, and used to train an agent capable of controlling a simulated chemical process. A real time simulation data interface is established for the RL agent to facilitate online RL and a system is devised for trained controller deployment as part of the designed RL framework. The development

phase is divided into several proof-of-concept (POC) stages with gradually rising levels of complexity concerning the simulated test environment and control objective. POC experiments culminate in a case study on an industrial process that traditionally leverages legacy APC solutions for optimal control.

Three research objectives can be deduced from the motivation and aim of this thesis. The first objective is to evaluate and select an appropriate RL algorithm, and to apply it in an implementation that suits APC in a process simulation. The second objective is to successfully compromise effective APC performance in a stochastic and continuous POC control problem with acceptable data-efficiency and computational complexity, to make the solution feasible in practice. The final research objective is the effective utilisation of an RL agent in a simulated industrial process with comparable control objectives to current legacy APC solutions.

This thesis is limited to exploring RL-based APC as a proof-of-concept in a simulated environment. The exploration of RL control in physical plants is left for future research. Furthermore, the developed solutions are confined to the regulation of process variables in a relatively limited number of functional units to simplify problem formulation and hence aid in iterative prototyping. The analysis of case processes is limited to control philosophy and critical interactions between variables that affect optimal control. Case processes are used as test environments for control solutions and comprehensive understanding of the chemistry and reaction kinetics inside unit functions is out of the scope of this work.

The remainder of the thesis is divided into four chapters. Chapter 2 covers the theoretical background of RL and optimal process control. Additionally, the current state-of-the-art in RL-based APC is analysed in the form of a literature review. Chapter 3 presents the materials and methodology employed in developing solutions to the introduced aim and research objectives. Chapter 4 covers the iterative POC solutions and the final case study in more detail, presenting the performance results obtained during agent training and after controller deployment. Chapter 5 summarises and concludes the thesis, evaluating results against research objectives, in addition to suggesting avenues of related future research.

2 Background

This chapter summarises the theory of RL to formalise fundamental concepts discussed throughout the thesis. In addition to individual theoretical concepts, the three main algorithmic RL approaches of policy optimisation, approximate dynamic programming, and actor-critic methods are covered. This is relevant for explaining the algorithmic logic and theory behind implementations introduced in Chapters 3 and 4. In addition to generic RL, the concept of optimal control in the process industry is covered, relating it to potential RL solutions. To give context to conducted research and development in subsequent chapters, a literature review is included on existing chemical process control implementations using the three analysed algorithmic approaches. To conclude the chapter, current challenges in RL-based APC are analysed together with corresponding potential solution strategies.

2.1 Reinforcement learning

ML is a field in computer science stemming from the more general concept of artificial intelligence (AI). AI can be defined as the study of agents that receive perceptions from the environment and perform actions accordingly. Furthermore, the field of AI not only attempts to understand but also to build intelligent entities. AI is a rapidly advancing field in science and engineering, seeing significant growth since the latter half of the 20th century. The considerable potential of AI has seen it regularly be cited as one of the most interesting fields of study by scientists in other disciplines (Russell et al., 2010).

ML started to emerge as its own field of study under AI by the 1990's. This reorganisation came from a shift in focus away from AI in the traditional general sense to practical problems incorporating statistical and probabilistic approaches (Langley, 2011). As the name implies, ML deals with methods that allow computer programs to learn towards some goal. More explicitly, the methods in question utilise available data (experience) to improve performance against a set of criteria, with respect to a class of tasks and a performance measure (Mitchell, 1997).

The field of ML can be divided into three distinct subcategories or paradigms responsible for contrasting problem types: supervised learning, unsupervised learning, and RL. Most of the current and past research in ML deals with supervised learning. The central premise of supervised learning is the utilisation of a labelled training dataset, provided by an external supervisor, with underlying knowledge of desired agent behaviour (Sutton and Barto, 2020). Training samples comprise descriptions of situations and corresponding labels of correct actions that the agent should take. Training data is used to build a model with the aim of generalising or extrapolating behaviour to situations that are not present in the training examples (Sutton and Barto, 2020). Supervised learning agents are often used to categorise objects or fit regression models to predict outcomes (Russell et al., 2010).

Another paradigm of machine learning is known as unsupervised learning, in which an agent attempts to find hidden structure and meaning in unlabelled and uncategorised data (Sutton and Barto, 2020). Applications of unsupervised learning

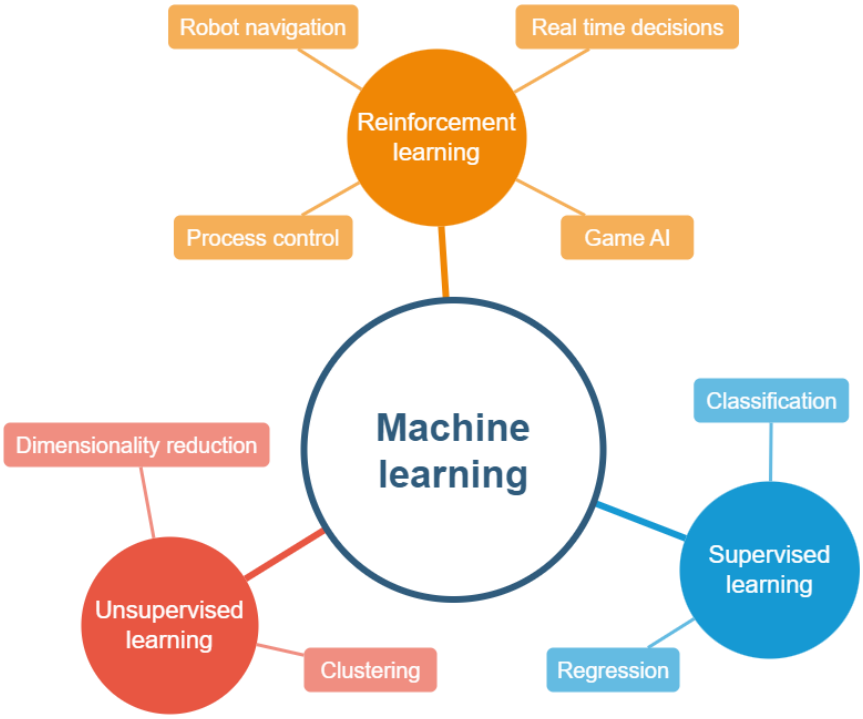

Figure 2: Taxonomy of ML paradigms.

often involve discerning categories or classifications in a collection of objects with no access to explicitly labelled examples or model feedback. An example of unsupervised ML would be learning types of stars in the cosmos and their characteristics from recorded spectra localised in image data (Russell et al., 2010).

RL, unlike the other two ML paradigms, is not designed to categorise data or fit regression models for extrapolation purposes. While RL shares a common feature with unsupervised learning in not incorporating a labelled training set, that is where the similarities end. An RL agent attempts to learn interactive behaviour trough feedback from the environment (Mitchell, 1997). RL can be defined as learning decision-making through mapping situations to actions, while maximising a numerical reward signal. The central difference in the learning process when compared to supervised learning is that the learner is not told what action to take. Instead, the RL agent must discover which actions yield the greatest reward through trial-and-error. Choice of action can affect the immediate reward received but also have a knock-on effect, impacting all subsequent system states and hence future reward signals. Trial-and-error exploration and delayed reward feedback are the defining features of RL (Sutton and Barto, 2020). This premise gives RL wide-reaching applications in problems ranging from control of mobile robots and algorithms designed to excel at board games, to optimisation of industrial factory processes (Mitchell, 1997). It has even been argued, somewhat paradoxically, that RL encompasses all of AI, in that an agent must independently learn to behave successfully within an environment (Russell et al., 2010). The taxonomy of ML can be seen in Figure 2, including example applications of the three ML paradigms. The following subsections explore RL in more detail from a conceptual

perspective as well as an algorithmic point of view, describing the implementation approaches in designing an effective RL agent.

2.1.1 Fundamental concepts of RL

RL can be considered as being made up of six key concepts: the agent, environment, policy, reward signal, value function, and optionally, a model of the environment. Fundamentally, RL involves the interaction between a decision-making agent and its environment over a discrete sequence of time steps (Sutton and Barto, 2020). The environment is represented at each time step as a state, which is a complete description of the world around the agent. The agent might, however, be unable to perceive the entire state, and be restricted to a partial observation of the world (Russell et al., 2010). The agent seeks to achieve some goal despite uncertainty about the surrounding environment. The RL agent is an abstraction of the learner and decision maker in the RL paradigm. Actions are decisions made by the agent and states, or observations of states, are the basis for making these decisions (Sutton and Barto, 2020). The interaction between the agent and environment is visualised in Figure 3.

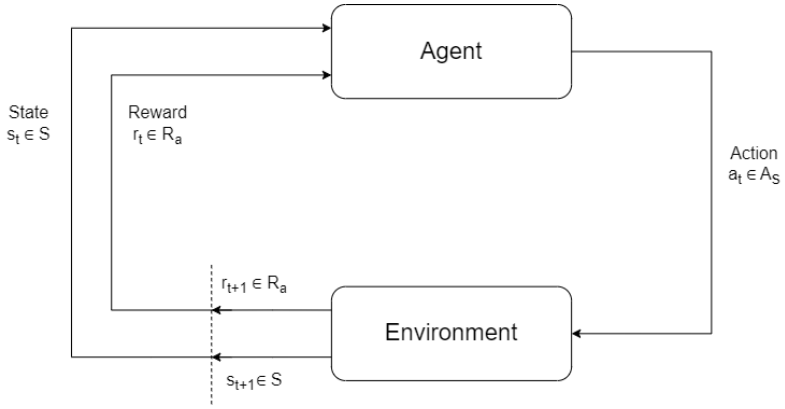

Figure 3: Agent-environment interaction in the RL paradigm.

The finite Markov Decision Process (MDP) is a common way to frame learning from interaction to achieve a goal through sequential decision making. The MDP is a discrete-time stochastic control process that can be stored in a 4-tuple (S, A_s, P_a, R_a), where S (state space) is the set of system states, A_s (action space) is the set of actions available in state s, $P_a(s_t, s_{t+1})$ is the probability of transitioning from state s_t to s_{t+1} given action a, and $R_a(s_t, s_{t+1})$ defines the reward received after taking action a and transitioning from state s_t to s_{t+1}. Through interacting with the environment, the agent, together with the MDP, forms the concept of a sequence or trajectory (also referred to as episodes or roll-outs), taking the following form:

$$\tau = (s_0, a_0, s_1, a_1, \ldots) \,. \tag{1}$$

Finite MDPs have a finite number of elements in the state, action, and reward sets. RL can be formalised using these ideas from dynamical systems theory as the optimal

control of incompletely known MDPs. Theoretically, any method able to handle this problem can be considered an RL method (Sutton and Barto, 2020). A schematic example of a finite MDP is shown in Figure 4.

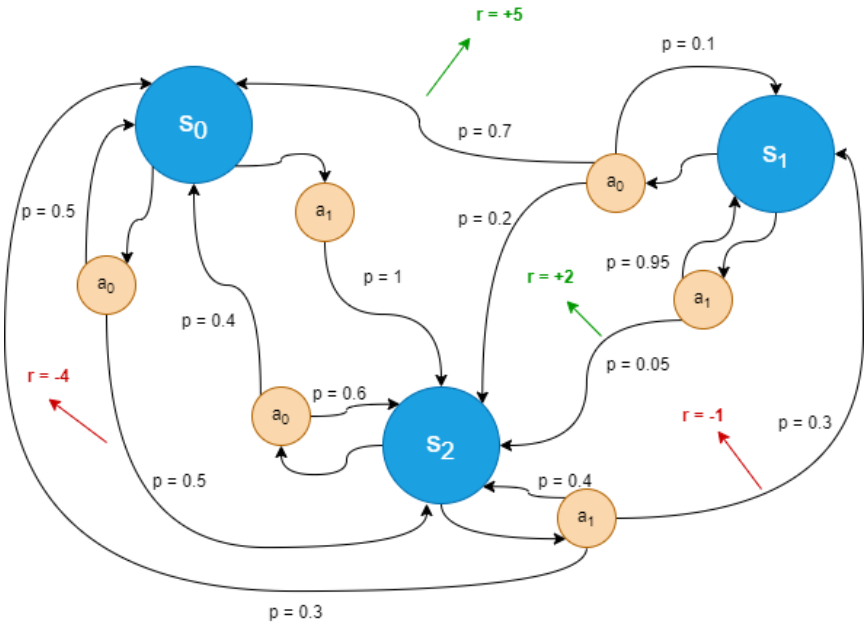

Figure 4: Illustration of an MDP containing states, actions, and corresponding probabilistic transitions and rewards.

The third fundamental concept of RL is policy. In simple terms, a policy is the function that maps perceived states to corresponding actions, to be taken when encountering states in the environment. The policy of an agent acts as the brains of the decision maker. Policies can be deterministic, where they map a deterministic action a_t to a given state s_t, or stochastic, where the probabilities of selecting available actions a_t, given a state s_t, are specified, and the agent selects an action from a probability distribution (Sutton and Barto, 2020). Deterministic policies are often denoted by μ and stochastic policies by π. Parameterised policies have functional outputs dependent on a set of parameters denoted by θ, which can be adjusted through optimisation to affect agent behaviour:

$$a_t = \mu_\theta(s_t), \tag{2a}$$

$$a_t \sim \pi_\theta(\cdot \mid s_t). \tag{2b}$$

The reward signal describes the immediate feedback from agent-environment interaction, analogous to pleasure and pain responses in human beings. The reward signal consists of a single numeric value r_t sent to the RL agent at each time step. The received reward determines the goodness of an action taken and the situation that the agent finds itself in. Policy is primarily adjusted based on the reward signal, as the only defined objective of the agent in the RL framework is to maximise total reward received in the long term over a trajectory (Sutton and Barto, 2020). The

state-value function $V(s)$ can be used to define this long term reward. The value of a state is defined as the expected cumulative reward that is obtained, starting from that particular state and following a policy. It gives a measure of the desirability of a state. The expected cumulative reward considers the likelihood of future states and the rewards available from them. The farsightedness of a value function can be adjusted through discounting future rewards to put less weight on potentially less reliable future estimates of rewards. The action-value function $Q(s, a)$ is closely related to the state-value function and gives the expected cumulative reward received from taking a specific action in the current state, following a given policy. Action choices and hence, policy, is determined by state-value or action-value judgements, as the agent pursues actions that result in high value states. Estimating state-value or action-value is no trivial matter, and it is a key component in the RL paradigm (Sutton and Barto, 2020).

Expressing a value function in a stochastic context begins by defining the probability of a T-step trajectory τ given a policy π:

$$P(\tau \mid \pi) = \rho(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t), \tag{3}$$

where $\rho(s_0)$ is the probability of the start state. The expected cumulative reward $J(\pi)$ over a trajectory is hence:

$$J(\pi) = \int_{\tau} P(\tau \mid \pi)R(\tau)d\tau = \underset{\tau\sim\pi}{E}\left[R(\tau)\right], \tag{4}$$

where the sum of rewards over a trajectory, $R(\tau)$, in the finite-horizon undiscounted case is:

$$R(\tau) = \sum_{t=0}^{T} r_t. \tag{5}$$

Alternatively, in the infinite-horizon discounted case, with a discount factor $\gamma \in (0, 1)$, the sum of rewards is:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{6}$$

In simple terms, RL can be condensed into the following optimisation problem:

$$\pi^* = \arg\max_{\pi} J(\pi), \tag{7}$$

where π^* is the optimal policy. Hence, the optimal state-value function, $V^*(s)$, can be expressed as:

$$V^*(s) = \max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau) \mid s_0 = s], \tag{8}$$

and the optimal action-value function, $Q^*(s, a)$, can be expressed as:

$$Q^*(s,a) = \max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau) \mid s_0 = s, a_0 = a]. \tag{9}$$

The implementation of parameterised policies and value functions in the context of RL agents is discussed in upcoming sections of this chapter.

An optional component of the RL framework is a model of the environment. Model-based RL methods leverage models for planning courses of action through considering future outcomes before they are experienced by the agent. A model allows, for example, to make inferences about resultant states and rewards following actions. Model-free RL methods, on the other hand, are strictly trial-and-error learners. Both methods are similar in terms of their goal, but model-based RL relies on planning as its primary component while model-free RL relies on learning (Sutton and Barto, 2020). This thesis focuses on model-free RL in order to handle difficult-to-model stochastic control problems in the context of industrial chemical processes.

Without access to a comprehensive model of known state transitions, probabilities and rewards, model-free RL is faced with the crucial explore-exploit dilemma. This challenge is also known as the multi-armed bandit problem (referring to a player faced with multiple one-armed bandit type slot machines). The agent must exploit actions and states it has deemed to offer a high return in the past, but in order to discover favourable actions and states, it must also explore the state space with some degree of randomness. Neither pure exploration nor exploitation leads to success. Logically, the RL agent should begin by trying a variety of actions, gradually converging to a policy that appears the most successful, but an explicit solution to the explore-exploit dilemma remains undiscovered. In stochastic environments, the problem is complicated further by having to try each action multiple times to obtain an estimate of the expected return (Sutton and Barto, 2020). Different approaches to solving the RL problem and hence, the multi-armed bandit problem are analysed in upcoming sections.

Another challenge in model-free RL is arbitrarily large and possibly non-discrete state and action spaces. This, combined with not having an explicit model of the MDP, means that one cannot expect to find the optimal solution even with unlimited time and data constraints. In this case, the goal of the RL agent becomes to find a satisfactory approximate solution using limited computational resources. In many practical applications, almost every state that the RL agent encounters, has not been seen before. This requires effective generalisation from a smaller subset of experience to good approximations in new situations. An effective method to accomplish this is function approximation. This generalisation technique attempts to approximate an entire function (e.g. parameterised policy or action-value function) by extrapolation using set of collected examples. Furthermore, extending RL with function approximation, makes it applicable for partially observable problems where the full state cannot be perceived by the agent. Supervised learning can be utilised for function approximation, and artificial neural networks (ANN) have become a powerful tool to accomplish this, introducing the concept of deep RL (Sutton and Barto, 2020).

2.1.2 Deep reinforcement learning

The increasing popularity of RL for addressing sequential decision-making problems over recent years can be attributed to the combination of RL with deep learning techniques. Deep RL (DRL) has expanded the potential of RL to high-dimensional and even continuous state and action spaces. In addition to the powerful function approximation made possible by ANNs, neural networks also address the issue of feature engineering and selection due to their ability to extract different levels of abstraction from raw data (Francois-Lavet et al., 2018).

In simple terms, ANNs are used as a supervised learning tool for function approximation. They can be viewed as functions that map a dataset of learning samples into a model. The loss function is a measure of model error compared to realised input-output pairs, and the learning objective is to minimise this error. Successful function approximation must consider the bias-variance decomposition, where the model should be sufficiently rich to reduce bias but not too complex to avoid overfitting. It has been shown in practice that the strength of ANNs is in their generalisation capability even with complex models incorporating a high number of parameters (Francois-Lavet et al., 2018).

An ANN is a network of connected neurons (also known as nodes), arranged in layers, loosely inspired by the biological neural networks in the brain. Each artificial neuron computes an activation function of the sum of its real-number inputs and relays the result as real-number output, adjusted by a weight and bias, to connected neurons. The network comprises an input layer, receiving the raw input data, and an output layer, releasing the outputs of the function approximator. In between the input and output layer, there is some number of hidden layers (Russell et al., 2010). The general structure of an ANN is visualised as a schematic diagram in Figure 5. Deep learning is simply characterised by ANNs containing two or more hidden layers of neurons. Each neuron produces a sequence of real-valued activations, with input neurons being activated through sensors that perceive the environment, and subsequent neurons being activated through weighted connections (edges) from previously activated neurons. Learning or function approximation is based on adjusting weights and biases, and hence the signal strength over connections, to make the ANN exhibit desired behaviour. Such behaviour may require extended causal chains of computational stages in the form of layers (deep learning), that transform the aggregate activation of the network (Schmidhuber, 2015). Each layer is responsible for elevating the abstraction level (starting from raw input data) to a higher degree. This facilitates the approximation of very complex functions, amplifying important aspects of inputs and suppressing irrelevant details (LeCun et al., 2015). Referring to Figure 6, the output of a neuron can be defined as:

$$a_j = g\left(\sum_{i=0}^{n} w_{i,j} a_i + b_j\right), \tag{10}$$

where g is the activation function, $w_{i,j}$ is the weight of an input edge, a_i is the output of the corresponding connected neuron, and b_j is the bias.

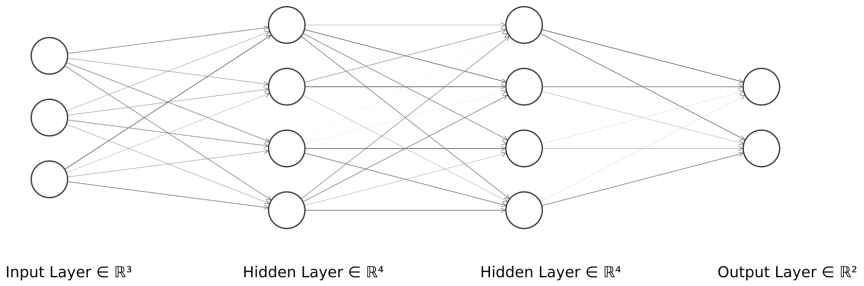

Figure 5: Visualisation of a basic feedforward artificial neural network structure. Edge opacity corresponds to weight.

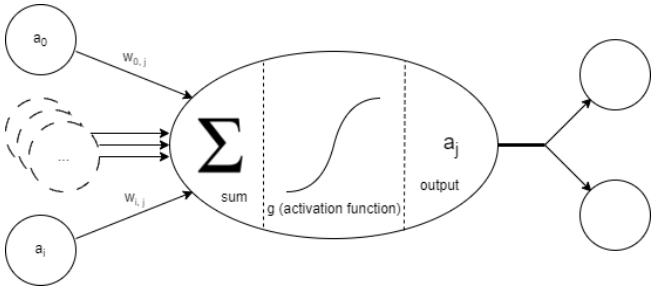

Figure 6: Visualisation of a single neuron. Neuron-specific weight b_j is added to the sum of inputs inside the activation function.

ANN's are trained using input-output examples by comparing the difference between the output of the network given an input (prediction) and the target output. All layers are trained to minimise a measure of the empirical error, $L(\theta)$, produced by this loss function. Parameters of the function approximator (weights and biases of the network) are commonly optimised using a gradient descent method via the backpropagation algorithm (Francois-Lavet et al., 2018). The gradient of the loss function is calculated with respect to the parameters of the function approximator for each input-output sample. Partial derivatives with respect to individual weights and biases are used to adjust parameters using gradient descent, working back one layer at a time from the last parameterised layer towards the input layer (Russell et al., 2010). For example, in basic gradient descent, at each training iteration, network parameters, θ, are adjusted using a step size determined by the learning rate, α:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta). \tag{11}$$

Beyond the basic principles of a feedforward ANN's and optimisation algorithms, the topology and hyperparameters determining the structure of the network need to be decided based on the desired application of the function approximator (Francois-Lavet et al., 2018).

In the context of RL, there are two main approaches to leverage function approximation to converge on an optimal policy. One way is to approximate the action-value

function using an ANN and hence give estimates of expected returns for actions, given an input state. After approximating values for actions, the policy can be constructed using a greedy approach, selecting the highest action-value with some degree of randomness to address the explore-exploit dilemma. This family of solutions is known as value-based methods, or alternatively as approximate dynamic programming (ADP). Another deep RL approach is to directly parameterise the policy function that maps actions to states as an ANN in the form of a policy network and optimise the weights to maximise expected return. This is known as the policy gradient approach. These RL algorithm families are analysed in the following sections, along with the actor-critic method, which combine both action-value approximation and the policy gradient approach (Francois-Lavet et al., 2018; Sutton and Barto, 2020).

2.1.3 Approximate dynamic programming

The class of algorithms known as approximate dynamic programming (ADP), or value-based methods, attempts to construct an estimated value function, which facilitates subsequently defining a policy for the RL agent (Francois-Lavet et al., 2018). This approach to RL has been very popular, finding applications in a range of progressively more advanced domains (Jang et al., 2019). In principle, the theory behind value-based methods is quite simple; an optimal policy can be found if expected cumulative rewards for state-action pairs are known and actions are chosen accordingly (Sutton and Barto, 2020). The optimal action can be obtained as follows if the optimal action-values or action-value function is known:

$$a^*(s) = \arg\max_a Q^*(s, a). \tag{12}$$

The novelty in different value-based solutions is how to find and keep track of expected values of state-action pairs.

In simple cases, where state and action spaces are small, tabular methods can often be used to find the exact optimal policy without needing approximation (Francois-Lavet et al., 2018). The basis of many modern ADP solutions was introduced in 1989 with Q-learning (Watkins, 1989), incorporating this tabular value-based approach. The original Q-learning algorithm keeps a lookup table of action-values, Q(s,a), having an entry corresponding to each state-action pair (Francois-Lavet et al., 2018). The Bellman equation is used to learn the optimal Q-value function:

$$Q^*(s_t, a_t) = \underset{s_{t+1}\sim P}{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})], \tag{13}$$

where $s_{t+1} \sim P$ (shorthand for $s_{t+1} \sim P(\cdot|s_t, a_t)$) indicates that the next state is sampled from the distribution produced by the transition rules governing the environment. The discount factor is represented by γ.

Larger state-action spaces in more advanced applications require approximation of the action-value function and the original Q-learning algorithm needs to be modified accordingly. Google DeepMind introduced the deep Q-network (DQN) in 2015 (Mnih et al., 2015), which uses the same value-based basic principle as Q-learning but

incorporates an artificial neural network structure to approximate the Q-value function. More specifically, DQN uses two separate ANN's to approximate the update target and the current Q-value. The central idea is to iteratively shift the Q-value estimate towards the update target through function approximation. An input-output pair is obtained consisting of the state and target action-value representation respectively (Mnih et al., 2015). By comparing the target, and the Q-value output approximated using the current Q-network, the approximation can be iteratively improved through supervised learning using the following loss function:

$$L(\theta) = \underset{s_t, a_t, r_t, s_{t+1} \sim \pi}{E} \left[(y_t - Q_\theta(s_t, a_t))^2 \right], \tag{14a}$$

where $Q_\theta(s_t, a_t)$ is the current Q-value estimate using the most recent Q-function parameterisation in the Q-network and y_t is the current update target approximated by the target network:

$$y_t := r(s_t, a_t) + \max_{a_{t+1}} \gamma Q_{\theta'}(s_{t+1}, a_{t+1}), \tag{14b}$$

where $Q_{\theta'}(s_{t+1}, a_{t+1})$ is the current target Q-value produced by the most recent target network parameterisation.

In the DQN algorithm, the target network is otherwise identical to the Q-network, but the weights of the target network are updated by copying the Q-network once every C episodes. Because of this detail, the target network updates more sparsely than the Q-network (Mnih et al., 2015). This has a stabilising effect on the algorithm since by locking the target, the risk of divergence due to "chasing one's tail" is reduced. The risk of instability if both networks use the same parameterisation is caused by the target and prediction not being independent from each other (Francois-Lavet et al., 2018). A schematic diagram of the DQN algorithm can be seen in Figure 7, showing how the loss function is constructed using the two ANNs. For simplicity, the output layers contain one node in the diagram, representing the highest Q-value out of the available actions. In the original DQN implementation, the output layers of the two networks contain a node for every available discrete action, representing the corresponding Q-value and target Q-value respectively (Mnih et al., 2015).

Several different algorithms have been developed in the value-based family, each having distinct advantages and characteristics (Jang et al., 2019; van Hasselt et al., 2016). However, some fundamental elements shared within the algorithm family need to be accounted for during implementation. Since the policy is constructed through selection of actions based on an estimated optimal value function, the explore-exploit dilemma needs to be accounted for using a greedy strategy with some randomness to enable exploration of the state and action space. In value-based methods, a so-called ε-greedy strategy is commonly deployed as the behaviour policy of the agent. In an example of such a strategy, a randomised action selection is made with a probability of ε. If ε is 0.1, every tenth action is chosen randomly for exploration and otherwise the behaviour policy exploits the current action-value function and chooses the action with the estimated highest cumulative reward (Sutton and Barto, 2020).

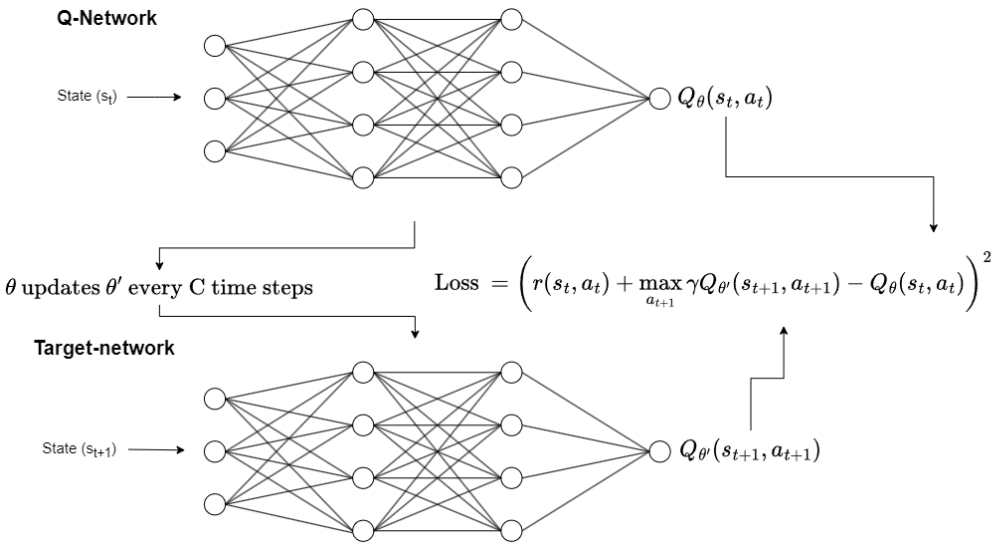

Figure 7: Construction of the loss function using the Q-network and the target network in the DQN algorithm.

A related concept is the on-policy versus off-policy approach in the learning process. On-policy RL uses the current target policy as an identical behaviour policy for exploring the state and action space. On-policy methods evaluate and improve the same policy used to make decisions and generate data (Francois-Lavet et al., 2018). In contrast, value-based methods tend to be off-policy in nature (Jang et al., 2019). This means that a distinct behaviour policy is used to generate data, experience, and explore the state and action spaces (Francois-Lavet et al., 2018). In Q-learning, the ε-greedy exploration policy constitutes the behaviour policy whereas the target policy is a greedy strategy of choosing the highest value action in each state (Jang et al., 2019). In off-policy algorithms, the separate behaviour policy is used to converge to an optimal target policy through an effective combination of exploration and exploitation (Mnih et al., 2015). An off-policy approach also allows for a concept known as experience replay, which involves storing state transitions in replay memory and utilising the data in future policy updates. This is because off-policy RL can leverage trajectories that have not been collected using the current target policy (Francois-Lavet et al., 2018). In practice, a replay buffer of past transitions is maintained and recalled from, when training the agent, in the form of minibatches consisting of current and past policy data. This improves data efficiency, removes correlations in the used observation sequences and smooths changes in data distributions (Mnih et al., 2015).

Indeed, the off-policy approach incorporated in value-based methods, make this algorithm class comparatively data-efficient when implemented appropriately. Sometimes, however, the action-value function itself might be highly complex and difficult to approximate (Sutton and Barto, 2020). Furthermore, value-based methods run into difficulties when dealing with continuous action spaces. In ADP, discrete state-action pairs are evaluated and chosen from accordingly. Using a value-based method for a continuous action spaces in domains such as chemical process control,

would require discretisation into arbitrarily small action intervals, raising complexity significantly (Duan et al., 2016). An alternative approach to RL, that can solve these problems, is the parameterisation of policy directly in the form of policy optimisation (Sutton and Barto, 2020).

2.1.4 Policy optimisation

Policy optimisation forms another class of algorithms in RL, using a concept known as the policy gradient. This approach attempts to maximise some performance objective, typically the expected cumulative reward, through directly parameterising the policy (Sutton and Barto, 2020). The policy, expressed as a policy network in deep RL implementations, maps actions directly to states without having to consult an approximation of the value function. In stochastic policy gradient algorithms, the output of the policy network is a continuous probability distribution that the agent uses to sample an action, dependent on the system state as an input (Francois-Lavet et al., 2018). In practice, the output layer can be made up of two nodes, representing the mean and standard deviation of the stochastic action distribution. Hence, policy gradient methods suit RL problems with stochastic policies or continuous action spaces. A visualisation of the policy network can be seen in Figure 8. The policy network is optimised through stochastic gradient ascent (or descent depending on problem formulation) with respect to the policy parameters, represented by weights and biases in the policy network (Sutton and Barto, 2020).

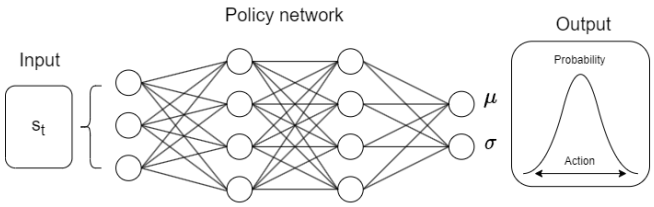

Figure 8: Visualisation of a policy network used in a policy optimisation algorithm. μ represents the mean of action probability or action distribution in a stochastic or continuous action policy respectively. The associated standard deviation is represented by σ.

The vanilla policy gradient algorithm can be derived as follows, starting from the central premise of optimising policy parameters, θ, through gradient ascent along the policy gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta), \tag{15}$$

where the gradient of the policy performance, $\nabla_\theta J(\pi_\theta)$, is called the policy gradient, and α is the learning rate. Through previously defined expected cumulative reward in Equation (4), the policy gradient can be computed as:

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{E}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau \mid \theta) R(\tau) d\tau \\
&= \int_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau) d\tau \\
&= \underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log P(\tau \mid \theta) R(\tau)] \\
&= \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right],
\end{aligned}
\tag{16}
$$

where $R(\tau)$ is the return over a sampled trajectory τ of T transitions, and $\log \pi_\theta(a_t|s_t)$ is the logarithmic probability of the current policy selecting action a_t given state s_t.

Policy optimisation consists of a policy evaluation element and a subsequent policy improvement step. In Equation (16) policy evaluation is realised through the measured return of a trajectory. In the conventional policy gradient formulation, the policy improvement step increases the probability of the policy selecting actions proportionally to the expected return over the associated trajectory. A critical element in policy gradient methods from a more general perspective is how to evaluate the policy, that is, how to obtain a measure or estimate for the return of said trajectory (Francois-Lavet et al., 2018).

The simplest way to evaluate the returns produced by a policy is to add up the total discounted cumulative rewards over realised trajectories with Monte Carlo sampling (Francois-Lavet et al., 2018). The vanilla policy gradient approach derived in Equations (15) and (16), forms the original Monte Carlo policy optimisation algorithm also known as REINFORCE (REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility) (Williams, 1992). REINFORCE updates the policy in retrospect after the completion of training episodes, increasing likelihood of state-action pairs proportionally to the realised discounted cumulative reward. Algorithm 1 presents the outline of REINFORCE in pseudocode.

The policy optimisation strategy in REINFORCE has attractive theoretical convergence properties since the expected update between episodes is in the direction of the performance gradient. This guarantees convergence to at least a local policy optimum with a sufficiently small learning rate, α. However, the Monte Carlo property of REINFORCE exhibits high variance, which may correlate with a slow learning process as several episodes are usually needed to obtain a good estimate of the return. To improve efficiency and speed up learning, variance in returns between sampled trajectories can be reduced with the use of a concept known as a baseline or control variate for the gradient estimator:

$$
\nabla_\theta J(\pi_\theta) = E_t\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(R(\tau) - b(s_t))\right],
\tag{17}
$$

where $R(\tau)$ is the return over a sampled trajectory and $b(s_t)$ is the baseline function.

Algorithm 1 REINFORCE: Monte-Carlo policy gradient

Input: Differentiable and parameterised stochastic policy network $\pi_\theta(a|s)$
Initialise learning rate α, discount factor γ and policy parameters θ
for episode = 1, . . ., M **do**
    Generate episode $(s_0, a_0, r_1), \ldots, (s_{T-1}, a_{T-1}, r_T)$, using π_θ
    for t=0, . . ., T-1 in episode trajectory **do**
        Calculate discounted cumulative reward for episode trajectory:

$$R(\tau) \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k.$$

        Update policy parameters after looping through each step of episode:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau).$$

    end for
end for

Instead of weighting policy updates with total returns of trajectories, variance can be reduced by lowering the magnitude of the weight terms, being careful to use a value that does not change the gradient estimator itself. This can be practically any term dependent on state, or even a random variable. Often a form of expected or average return for a state is employed as the baseline to measure relative goodness of actions in a trajectory (Francois-Lavet et al., 2018). Alternatively, instead of Monte Carlo sampling trajectories and calculating discounted cumulative rewards, a value-based approach can be used to estimate the future return of state-action pairs given a policy iteration. The average value of the state can then be subtracted as the baseline to give the advantage function. Using the advantage function in the vanilla policy gradient framework for weighting the policy update steps gives the following expression:

$$\nabla_\theta J(\pi_\theta) = E_t \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t) \right], \tag{18}$$

where the advantage function is defined as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t), \tag{19}$$

where $V(s_t)$ is the average action-value of state s_t. RL implementations which leverage policy optimisation together with value estimation for policy evaluation form a whole class of algorithms known as actor-critic methods (Sutton and Barto, 2020).

2.1.5 Actor-critic methods

As discussed in Section 2.1.4, successful policy iteration using the policy gradient requires evaluation of the actions produced by the current policy. An actor-critic framework achieves this through two separate function approximators. The actor in

this class of algorithms refers to a policy approximator and the critic refers to a value function approximator. As the names suggests, the actor takes actions based on a policy and the critic evaluates the actions taken, giving feedback which is then used to update the new policy. Conversely, the critic updates its value function parameters based on the new policy, forming an iterative actor-critic cycle (Francois-Lavet et al., 2018).

In deep RL implementations, the actor and critic are represented by two separate function approximators in the form of ANNs and are hence parameterised individually (Demidov and Al-Refai, 2021). An actor-critic approach known as Advantage Actor-Critic (A2C) was introduced in Equations (18) and (19), as an extension of REINFORCE using advantage function approximation, instead of Monte Carlo trajectory sampling for return estimation (Mnih et al., 2016). As it is based on the vanilla policy gradient approach, the actor in A2C produces a stochastic policy.

Even with reduced variance through the utilisation of baselines, a risk of using policy optimisation as part of an actor-critic algorithm is taking overly aggressive update steps between policy iterations. This can cause the agent to fall off a path towards the optimum policy and diverge. To improve stability and, slightly counter-intuitively, rate of convergence, policy updates need to be constrained. OpenAI developed the, now very popular, Proximal Policy Optimization (PPO) algorithm in 2017 to improve stability of A2C in a simple to implement and sample efficient solution (Schulman et al., 2017). PPO introduces a clipped surrogate objective function in the policy gradient, where $J(\pi_\theta)$ from Equation (18) becomes:

$$J^{CLIP}(\pi_\theta) = E_t \left[\min(r_t(\theta)A(s_t, a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A(s_t, a_t) \right], \tag{20}$$

where $r_t(\theta)$ is the ratio function:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{21}$$

describing the probability ratio between the current (π_θ) and old ($\pi_{\theta_{\text{old}}}$) policy. The ratio effectively gives an estimate for divergence over policy iterations. This ratio replaces the log probability previously used in the policy objective function. If action probability increases excessively between iterations, the resulting policy step could be overly aggressive, affecting performance negatively. The ratio is hence constrained by clipping it within a range from 1 defined by hyperparameter ϵ. A range of related algorithms have been developed to address the potential instability of unconstrained actor-critic implementations, including but not limited to Trust Region Policy Optimisation (TRPO) (Schulman et al., 2017).

Originally, stochastic policy gradient algorithms were proposed as the solution to RL problems with continuous action spaces as they allow action sampling from continuous probability distributions produced by the policy network. Stochastic policy gradient algorithms are indeed effective for optimising a single continuous action based on system state and associated rewards. However, the need for numerical integration over both state and actions spaces results in poor scalability and efficiency

when multiple continuous actions need to be optimised simultaneously in higher complexity systems (Demidov and Al-Refai, 2021). One solution is to deploy separate agents for individual actions. This approach, while simple in theory, is limited in addressing correlation between actions. Action outputs would need to be fed as inputs to the other policy networks in order to consider the collective impact of separate action policies. Actions would hence have to be chosen sequentially, limiting parallel operation capabilities of the sub-agents. An alternative approach would be to learn a multi-variate policy distribution for multiple parallel actions. This type of policy network would have to learn all values in the associated co-variance matrix describing the correlation between actions, increasing the required number of learned values from $2n$ (action mean and standard deviation), for n actions, to $n + n^2$ (n action means and the values in a $n \times n$ co-variance matrix). This characteristic rapidly blows up the computational complexity of the algorithm when presented with a multi-dimensional action space.

Recent development in actor-critic algorithms have leveraged deterministic policy gradient approximation together with deep Q-learning to train continuous, and multi-dimensional action space policies, with higher efficiency. Efficiency gains result from avoiding laboursome co-variance estimation and integration over both state and action-spaces in deterministic policy gradient approximation (Demidov and Al-Refai, 2021). Deterministic policy networks do not output a probability distribution of a continuous action but instead estimate the exact optimal actions as deterministic numerical values. This allows the policy network to optimise an arbitrary number of simultaneous actions. In an actor-critic framework, a value function approximator is used to evaluate the output of the deterministic policy network to update the actor (Demidov and Al-Refai, 2021; Francois-Lavet et al., 2018).

The original Deterministic Policy Gradient (DPG) method (Silver et al., 2014), which is restricted to discrete action spaces, has been further developed to specifically address continuous action spaces in a model-free off-policy actor-critic algorithm known as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016). DDPG combines elements from DPG and Q-learning using ANNs as the respective function approximators. A parameterised deterministic policy network is used as the actor to output exact actions from input states, specifying the current policy, μ_θ. A Deep Q-Network (DQN) is used as the critic, taking in corresponding actions and states, outputting the respective state-action value. The Bellman equation is used to learn the critic as with the regular DQN algorithm covered previously. As opposed to stochastic policy gradient implementations, DDPG needs to manually address the explore-exploit dilemma since there is no inherent stochasticity in action selection. This is achieved with a separate exploration policy μ' by adding noise $\mathcal{N}$ to the actor policy parameterised by θ:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N}. \tag{22}$$

The noise in the original DDPG implementation is in the form of an Ornstein-Uhlenbeck process, which is a type of stochastic process model with wide applications from financial mathematics to physics. DDPG is an off-policy algorithm which

improves data efficiency through using a replay buffer, $\mathcal{D}$, which stores sampled experience in the form of realised transitions produced by previous behaviour policy iterations. Each update of the actor and critic uses a random mini-batch from the replay buffer to make samples independently and identically distributed (Demidov and Al-Refai, 2021; Lillicrap et al., 2016).

There is a common motivation behind DDPG and Q-learning, stemming from the relationship between the optimal action-value function and optimal action, described in Equation (12). The algorithm alternates between learning an estimate for $Q^*(s,a)$ produced by the critic and $a^*(s)$ produced by the actor. The novelty in DDPG, allowing it to solve policies for continuous action spaces, relates to the way maximum over actions in $\max_a Q^*(s,a)$ is calculated (Lillicrap et al., 2016). In discrete action spaces, Q-values can be exhaustively computed for every action and compared through deep Q-learning, immediately giving the action which maximises the Q-value (Mnih et al., 2015). This cannot be done in continuous action spaces as even discretisation of the action space and subsequent evaluation of maximum Q-value would result in a prohibitively expensive subroutine. The central premise of DDPG excludes trying to compute $\max_a Q^*(s,a)$ through individual state-action values, but instead presumes that $Q^*(s,a)$ is differentiable with respect to the action produced by the current policy due to the continuous action space (Lillicrap et al., 2016). Hence it can be approximated as:

$$\max_a Q^*(s,a) \approx Q(s,\mu(s)). \tag{23}$$

The Q-function learning process in DDPG follows the general principle proposed in the DQN algorithm (Mnih et al., 2015) and discussed in Section 2.1.3. The objective is to minimise the Mean Squared Bellman Error (MSBE) loss between the current Q-value approximation and the target Q-value:

$$L(\phi,\mathcal{D}) = \underset{(s_t,a_t,r_t,s_{t+1},d)\sim\mathcal{D}}{E}\left[\left(Q_\phi(s_t,a_t) - (r+\gamma(1-d)Q_{\phi'}(s_{t+1},\mu_{\theta'}(s_{t+1})))\right)^2\right]. \tag{24}$$

Transitions from replay buffer $\mathcal{D}$ contain a boolean terminal state flag, d, indicating if a terminal state has been reached. Q_θ is the current iteration of the Q-network and $Q_{\phi'}$ is the target Q-network. The action in the target component takes the form, $\mu_{\theta'}(s_{t+1})$, using the relationship in Equation (23), where $\mu_{\theta'}$ is the target policy given by a target policy network. The target policy network has the same role in policy updates as the target Q-network has in Q-value approximation, as explained in Section 2.1.3. The critic Q-network parameters, ϕ, are updated via backpropagation through performing gradient descent on an average MSBE loss, calculated over a mini-batch of transitions sampled from the replay buffer (Lillicrap et al., 2016).

After producing an updated approximation of the Q-value function, the policy optimisation component of DDPG involves learning policy parameters, θ, which produce actions that maximise the current Q-value function iteration $Q_\phi(s,a)$. This is done through gradient ascent with respect to policy parameters, in order to solve:

$$\max_{\theta} \underset{s\sim D}{E}\left[Q_\phi(s, \mu_\theta(s))\right], \tag{25}$$

where Q-function parameters are considered constants (Lillicrap et al., 2016).

Algorithm 2 Deep Deterministic Policy Gradient (DDPG)

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta' \leftarrow \theta, \phi' \leftarrow \phi$
for episode = 1, . . ., M **do**
    Initialise a random process $\mathcal{N}$ for action exploration
    Observe state s
    for t=1, . . ., T **do**
        Select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
        Execute a in the environment
        Observe next state s_{t+1}, reward r, and terminal state signal d
        Store (s, a, r, s_{t+1}, d) in replay buffer $\mathcal{D}$
        Randomly sample mini-batch, $B = \{(s, a, r, s_{t+1}, d)\}$ from $\mathcal{D}$
        Compute targets:

$$y(r, s_{t+1}, d) = r + \gamma(1 - d)Q_{\phi'}(s_{t+1}, \mu_{\theta'}(s_{t+1})).$$

        Update Q-function by one step of gradient descent using the MSBE loss:

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s_{t+1},d)\in B} (Q_\phi(s, a) - y(r, s_{t+1}, d))^2.$$

        Update policy by one step of gradient ascent using:

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s)).$$

        Update target networks using polyak averaging:

$$\phi' \leftarrow \rho\phi' + (1 - \rho)\phi,$$
$$\theta' \leftarrow \rho\theta' + (1 - \rho)\theta.$$

    end for
end for

Since the target values used in the MSBE loss function depend on the same parameters that are being trained, the familiar stability issue from the DQN algorithm persists. In the DQN algorithm itself, the issue was addressed with a separate target network updated with parameters copied from the main network at sparse intervals (Mnih et al., 2015). DDPG employs a similar logic to improve learning stability. However, instead of less frequent "hard" target updates (directly copying network

weights), DDPG makes "soft" target updates at every main network iteration using polyak averaging:

$$\phi' \leftarrow \rho\phi' + (1-\rho)\phi, \tag{26}$$

where ρ is a hyperparameter between 0 and 1, ϕ' is the set of target network parameters and ϕ is the new set of learned main network parameters. Soft target updates are used for both the target Q-network and target policy network. This update rule results in target network parameters tracking main network parameters slowly. The target in the MSBE loss function is hence kept more stationary, moving the inherently unstable task of learning the action-value function closer to a robust supervised learning task (Demidov and Al-Refai, 2021). Algorithm 2 presents the outline of DDPG in pseudocode.

2.2 Optimal control in process industry

In industry, chemical processes transform raw materials into end products through a sequence of chemical reactions and unit operations, constituting a plant. In chemical plants, unit operations are characterised by varying temperature, pressure, pH and other operational conditions. Plants utilise sensor suites to monitor these physical process variables, as well as analysis instrumentation to track occurring chemical reactions. An observation of overall process state can hence be compiled (Chai et al., 2014).

In the past, before large scale industrial automation, chemical processes were controlled manually at the unit level by operators and engineers. This essentially meant tracking process variables through measurement readings and keeping them at desired values for optimised operation. With the introduction of industrial automation, distributed control systems (DCS), that communicate with field instrumentation and actuators, have taken over the autonomous control of chemical plants under the oversight of engineers and operators (Woolf, 2009).

Control systems are a crucial component in modern chemical plants and have two main objectives. Firstly, they need to reliably maintain the process at desired operational conditions defined by process variable set points. Secondly, control systems must be able to transition the process from one set of operational conditions to another, motivated by changing process specifications, and economics or operational constraints. Control systems produce control actions that manipulate the plant process through a range of actuators, such as valves, pumps and variable frequency drives. Process set points are selected to create operational conditions which facilitate the required chemical reactions for meeting process objectives and end product specifications. Control systems are responsible for reaching and maintaining these conditions (Woolf, 2009). Furthermore, sophisticated higher level control systems can be designed to optimise a compromise between production output and operational safety, reliability and efficiency, along with other more abstract KPIs (Chai et al., 2014).

A modern approach to the control of industrial processes can be thought of as an optimisation challenge consisting of a hierarchical structure of control layers, defined

by their respective levels of abstraction. The bottom layer of the process control hierarchy consists of field devices in unit functions such as sensors, transmitters, actuators and valves. These components measure process variables and exert control actions that affect process state. The second layer is made up of the base level regulatory controllers of the DCS. Regulatory control is usually implemented with closed loop feedback controllers that send control signals to the field device layer, keeping process values measured by sensors within specified limits or set points. PID (proportional-integral-derivative) controllers are a standard approach for base level automated process control (Nicolae et al., 2019). The regulatory control of individual unit function variables, within the framework of manually adjusted set points and limits, exhibits a low level of abstraction, and hence automation, from an optimisation perspective. The field device layer and the regulatory control layer, when combined in an integrated DCS system with an operator interface, are sufficient for operating an industrial process but leave limited tools at disposal for high level process optimisation (Chai et al., 2014).

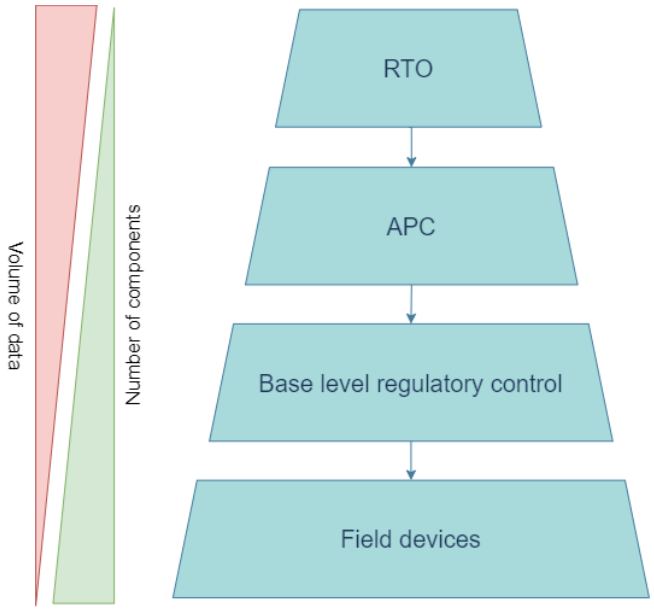

Figure 9: Process control levels.

Advanced process control (APC) forms an additional layer in the process control hierarchy, above the base level DCS control of individual unit variables. APC considers a collection of variables and evaluates how they relate to the overall performance of an entire process unit against some set of criteria. APC often also incorporates process modelling in order to make more intelligent control decisions based on forecasted future states. APC and the base level DCS system are not mutually exclusive as APC is an extension to the basic process control functionality, deployed to operate the unit function at its optimum operating point. This can be in the form of improving control performance and stability or optimising set points of variables within defined limits. The layer above APC in the process control pyramid deals with plant-wide production control. Overall plant KPIs are not necessarily dependent on the individual

optimisation of singular units, but instead require the coordinated management of unit function APC modules for plant-wide optimisation (Nicolae et al., 2019). In a conventional hierarchical control structure implemented in process industry, the level above APC is known as the real time optimisation (RTO) layer. RTO is a scheme used to optimise steady state performance of the process as a whole, accounting for market, process and corporate information to feed performance targets and KPIs to the APC level (Santander, 2015). Figure 9 visualises the hierarchical structure of process control layers in industrial plants. The scope of this thesis focuses on regulatory control and APC at the unit function level.

2.2.1 Closed loop regulatory process control

In DCS systems, individual process variables are controlled most commonly with closed loop feedback control to keep them at defined set points. A closed loop controller refers to a control system which utilises an error signal as feedback, representing the difference between process value and set point. The feedback signal is used to produce a compensating control input that attempts to minimise error by adjusting the manipulated variable. The simplest form of closed loop regulatory control is the on/off controller. As the name suggests, the on/off controller alternates the manipulated variable between fully on and fully off, based on the value of the controlled process variable in relation to the set point. On/off controllers are simple in design and although binary sensors and actuators are reliable and inexpensive, drawbacks include inefficiency and degradation of binary actuators, as well as response noise when seeking stability (Woolf, 2009).

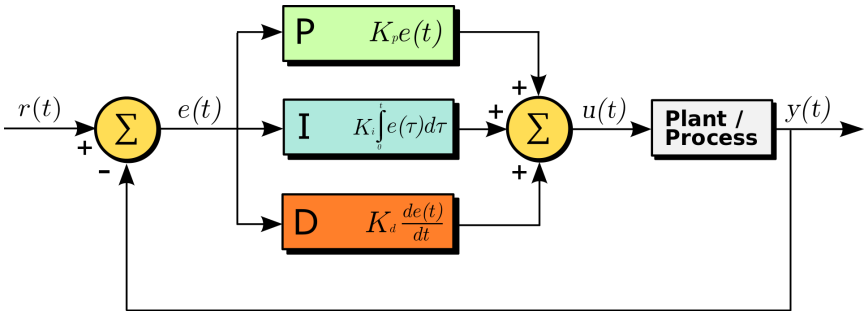

Figure 10: Block diagram of a PID controller.

Currently, the most popular regulatory controller in the process industry is the PID controller. It is a closed loop controller which continually or at discretised intervals (in digital systems) adjusts a continuous manipulated variable in response to the error signal. The error signal is altered by three parallel components to produce the control signal used to compensate for error. The proportional term (P) considers the current numerical error between the measured variable and process value and produces an output linearly proportional to this error. The integral term (I) integrates past error accumulation and hence aims to eliminate steady-state error over time, and the derivative term (D) anticipates future error by calculating a differential of the error over time. This dampens the response, reducing overshoot and oscillation by producing

an output proportional to the rate of change of error. The PID controller sums the three components together to produce the collective control signal. The components are weighted with gains that adjust their relative influence in the control law, having an impact on response characteristics. A block diagram of the PID controller structure can be seen in Figure 10. The effective deployment of PID controllers in industrial applications requires the tuning of gains for desired controller behaviour. A number of tuning schemes exist ranging from trial-and-error to mathematical methods (Woolf, 2009).

2.2.2 Current benchmark in advanced process control: Model predictive control

Model predictive control (MPC) is the industry standard in APC and process optimisation for extending the capabilities of DCS systems only relying on base level regulatory control (Mowbray et al., 2022). The central idea of MPC in process control is to leverage a dynamic system model to produce a finite horizon forecast of process behaviour. The forecast is then optimised against some criteria to produce an optimal control input at the current point in time. As the optimal control action is dependent on the state of the system, past process measurements are used to estimate this initial state. The most likely value for the initial state estimate is produced by reconciling past process data with the underlying dynamic model. The MPC problem is hence two-fold, consisting of the regulation problem of optimising control and the estimation problem of producing the optimal state estimate as a basis for the regulation problem. Both of these sub-problems require dynamic modelling and online optimisation (Rawlings et al., 2019). MPC is visualised as a block diagram in Figure 11.

Dynamic modelling can be done in the form of differential equation models, such as:

$$\begin{aligned} \frac{dx}{dt} &= f(x,u,t), \\ y &= h(x,u,t), \\ x(t_0) &= x_0, \end{aligned} \tag{27}$$

where $x \in \mathbb{R}^n$ is the system state, $u \in \mathbb{R}^m$ is the control input, $y \in \mathbb{R}^p$ is the system output, and $t \in \mathbb{R}$ is time. In dynamic modelling as part of the MPC framework, the idea is to seek a solution to the differential equation for time greater than t_0, given the initial condition of state x at time $t = t_0$. Using a model, process state predictions can be made based on some finite horizon control input sequence for N time steps into the future:

$$u = (u(0), u(1), \ldots, u(N-1)). \tag{28}$$

To optimise control inputs, an objective function needs to be defined which measures the cost or reward associated with control sequence u, initial state $x(0)$ and subsequent modelled state sequence. In a simple regulation problem, the objective function can

be the weighted total squared error of trajectory $(x(k), u(k))$ compared to the desired process value:

$$J(x(0), u) = \frac{1}{2} \sum_{k=0}^{N-1} [x(k)'Qx(k) + u(k)'Ru(k)] + \frac{1}{2} x(N)'Px(N), \tag{29}$$

where we assume for simplicity that the desired state set point of the regulation problem is 0. Weights Q, R, and P determine the influence that states and actions have on the penalty received from a trajectory. Weights can be tuned to drive a system quickly to the desired state at the the expense of large control actions and vice versa. The solution to the optimal control problem is obtained through minimising or maximising the objective function. In the case of a penalty scheme, the problem can be formulated as:

$$\min_{u} J(x(0), u). \tag{30}$$

The objective function allows multi-dimensional state and control input vectors, enabling applications in multiple input, multiple output (MIMO) control problems.

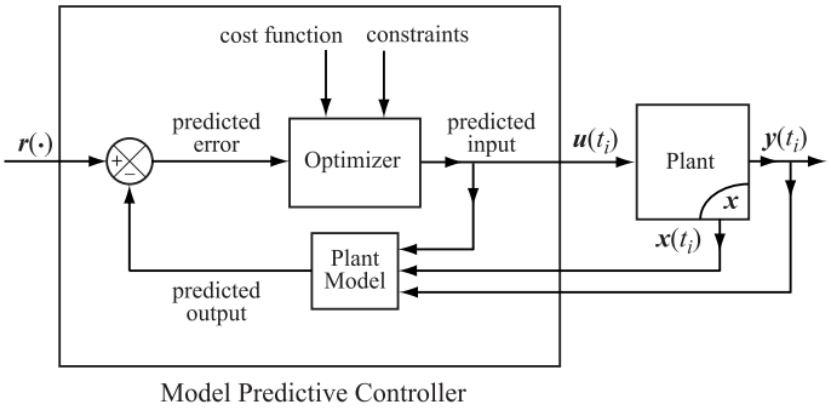

Figure 11: Block diagram of a state feedback model predictive controller.

If the modelled system is linear or linearised and the objective function is quadratic, the resulting MPC task is a linear-quadratic (LQ) problem. MPC in this application domain relates closely to another optimal control tool known as the linear-quadratic regulator (LQR). The difference between MPC and LQR control lies in the way that optimisation cost is calculated. LQR optimises a global control law for any state and time, whereas MPC optimises control inputs based on a receding time horizon, and updates the control rule dynamically. Because MPC deals with a local time frame, it makes no assumption of global linearity of the underlying process system and is hence capable of handling non-linear systems. Another distinguishing feature of MPC is the inclusion of hard constraints on control input sequences such as valve saturations and set point limits (Rawlings et al., 2019). Hence, MPC is used to optimise control actions over finite time horizons, while honouring operational constraints.

The customisability of the objective function and the inclusion of operational constraints allows MPC to be used as an APC tool to optimise the operating points of unit functions in a processes. The objective function and controlled variables can be chosen depending on application of the APC solution. When MPC is used in the APC layer on top of the base level regulatory control of unit functions, MIMO MPC systems can be used to manipulate and optimise set points of single input, single output (SISO) PID controllers. In this hierarchical strategy, lower level regulatory controllers maintain process variables at given set points, while a higher level MPC adjusts set points of multiple variables in order to optimise a unit-wide objective function based on chosen KPIs (Santander, 2015).

2.2.3 Data-driven approaches

A conceptual framework of the Industry 4.0 revolution, introduced by the German government, is widely accepted to have been in effect since 2011 (Mowbray et al., 2022). The devised framework involves the integration of information and communication technologies with industrial technology. A core element of Industry 4.0 is the development of Cyber-Physical Systems to realise digital and intelligent industrial facilities that promote information-led, customisable, and green manufacturing (Zhou et al., 2015). This transition in industrial practices is characterised by an abundance of data, digitalisation, and advanced hardware, facilitating digital and data-driven solutions to both emerging and established industry challenges. Particularly in the process modelling space over the past decade, a gradual shift has been observed from physical first-principles modelling into data-driven modelling, leveraging machine learning to discover complex relationships expressed implicitly in process data (Mowbray et al., 2022).

Industrial data-science can focus on a number of areas to improve the operational indices of chemical processes. Applications include the predictive maintenance of degrading components through anomaly detection and pattern recognition. The goal is to reduce maintenance downtime proactively, and hence increases production efficiency in the long term. Quality predictive models and soft sensors form another application area for industrial data-science. Machine learning models can be used to infer correlations between process data and quality measurements or KPIs, which would traditionally be samples analysed in a lab. These inferential sensors can be used to obtain online estimations of KPIs, giving better and more consistent feedback on process performance (Mowbray et al., 2022).

While predictive maintenance and soft sensors add significant value to operations as new features, today's competitive industrial setting, combined with stringent third-party regulations, has raised advanced process control and dynamic optimisation to paramount focus in process operation. Nevertheless, conventional feedback control laws for regulatory control (PIDs), and first-principles mechanistic models for online process optimisation (MPC) continue to dominate industrial practice in the automated control of chemical processes. The inherent motivation for developing more advanced process control, combined with the emerging capabilities through Industry 4.0, has resulted in the exploration of data-based process control approaches. Namely, RL

has been proposed as an attractive alternative for controlling and optimising chemical processes as it has the potential to outperform existing techniques in a range of applications (Mowbray et al., 2022).

2.2.4 Reinforcement learning as a novel approach to optimal control

The general RL problem described in previous sections can be intuitively applied to the framework of chemical process control. The control of an industrial process can be expressed as a sequential decision-making problem like any other RL problem and consists of three principal elements: the underlying system, a control element and an objective function. The purpose of the control element, or agent in RL terms, is to identify optimal control decisions and take actions accordingly, utilising the available means built into the process design, given measurements taken from the underlying system (process). The process evolves between control decisions according to underlying system dynamics. The optimality of the sequential actions and subsequent evolution of the process is assessed by the reward function, responsible for the feedback signal sent to the agent (Mowbray et al., 2022).

The reward function in RL control needs to reflect the desired behaviour of the control scheme given the environment and system dynamics. If the underlying process is stochastic, it is assumed under the RL paradigm, that the realisation of the uncertainty and the current state is expressed by an observation, or collection of measurements from the process. Hence, the control elements needs to be reactive to observations of stochastic states. In general RL terms, this implies that the control element in RL process control should be a stochastic (π) or deterministic (μ) control policy, depending on the implementation algorithm (Mowbray et al., 2022). The field of RL has produced a wide range of methods and algorithms capable of functionalising such a control policy including, but not limited to, the examples covered in Section 2.1.

In model-free RL, functionalisation of the control policy is done through closed loop feedback, without any explicit assumptions on the form of process stochasticity or system dynamics. Instead, the process state and action space is explored through sampling with varying control strategies, and functionalisation is thereafter improved using feedback from the reward function incorporating process measurements. This procedure of exploration and exploitation to derive a control policy for a sequential decision-making problem is defined as generalised policy iteration (Mowbray et al., 2022). The iteration of policy through trial-and-error, in the context of chemical process control, requires an effective simulation environment where the state-space can be explored freely. This is because it is not feasible to start a generalised policy iteration in a physical plant process with a completely untrained agent due to tight operating margins (Petsagkourakis et al., 2022), high computational complexity, and potential instability of RL algorithms (Petsagkourakis et al., 2020). The simulation needs to be a reliable and accurate representation of the physical plant process in order to produce a robust initial control policy.

The deployment of an initial simulation-based control policy can be done through the concept of transfer learning. The idea is to reuse and leverage information obtained by an agent in a source task environment, to operate in an unexplored target task

environment. In the context of deep learning, transfer learning relaxes the assumption that training data should be independent and identically distributed with test data (Faria et al., 2022). In practice, transfer learning can be applied in the deployment of an initial control policy by locking a subset of the policy parameters (network weights) to be constant. This effectively stores knowledge obtained from the source task, while at the same time reduces the required number of subsequent training epochs and episodes in the task environment through a lower number of optimisable parameters. Quicker learning in the physical plant process is beneficial as industrial scale plants are often more limited in terms of data and test episodes when compared to process simulations (Petsagkourakis et al., 2020).

On the condition that a robust simulation environment and an effective deployment strategy exists, RL-based process control has several advantages compared to legacy approaches. Firstly, generalised policy iteration combined with a stochastic process simulation theoretically facilitates the learning of an optimal control policy for a process with any form of uncertainty. The closed-loop feedback control logic of RL removes the need to identify models dealing with process dynamics and uncertainties in closed form, as is the case with MPC. Modelling system dynamics and stochasticity can be difficult and computationally expensive, and the online optimisation in MPC can form a bottleneck in performance. Additionally, the open-loop feed forward control in MPC can result in unideal predictions of future dynamics, resulting in overly conservative control actions. As a result, simulation and model-free RL can be leveraged as a powerful general paradigm for decision making in complex, non-linear and stochastic processes, exceeding the potential of legacy MPC systems (Mowbray et al., 2022).

Computational complexity and the engineering expertise in RL-based control are centered around the application of the chosen algorithm, development of a training framework and the actual offline training of the agent in some simulated environment before deployment (Petsagkourakis et al., 2020). Conversely, MPC in APC applications require engineering expertise and process knowledge for periodic recalibration of underlying models after deployment, as it is not a self adapting control scheme (Faria et al., 2022). Similarly, lower level regulatory PID control loops need to be tuned for stability depending on system dynamics (Woolf, 2009). The universality of the RL control framework in decision making results in application versatility and adaptability. An optimal control problem in RL can simply be described by a set of state-space observations, a suitable objective function and the action space. This quality makes RL theoretically suitable for all levels of process control, from base level regulatory control to APC, process scheduling, supply chain optimisation and beyond, simply by setting up the problem appropriately (Mowbray et al., 2022).

2.3 Literature review on current state-of-the-art in RL-based optimal process control

Although RL is closely related conceptually to optimal control theory, utilisation in the process control community has been thwarted by hardware and algorithm limitations.

The consolidation of deep learning theory in early 2010s, with the introduction of deep neural networks (DNN) as feature extractors and function approximators, facilitated RL in problems with high-dimensional state spaces (Faria et al., 2022). The resulting progress in DRL algorithms over the past decade and motivation from industry has spurred research into RL-based process control solutions. Approaches discussed in literature leverage value-based ADP, direct policy optimisation, and actor-critic methods, as algorithms and related applications range from batch process optimisation to higher level production scheduling problems (Mowbray et al., 2022). A brief summary of the current state-of-the-art in RL-based process control is presented in the following sections.

2.3.1 ADP solutions in optimal process control

DRL control approaches that utilise approximate dynamic programming, attempt to solve a sequential control problem by approximating the action-value function. Desirable control actions are selected based on the resulting value approximations of state and action combinations. For a more in depth explanation on ADP, refer back to Section 2.1.3.

Early research in ADP process control solutions include both a model-based and a model-free strategy introduced for data-driven control of non-linear processes (Lee and Lee, 2005). The model-based strategy, termed J-learning, leverages an empirical nonlinear model constructed from closed loop simulation test data, and performs conventional dynamic programming to improve on a control policy accordingly. Reliability of the data-driven model-based strategy is controlled by penalising value function extrapolation based on the local sparsity of the empirical data distribution. As a result, the exploration of the policy is restricted to areas of state-space with adequate empirical data. The second strategy utilises Q-learning for ADP in a model-less manner to estimate the Q-function (action-value) and hence optimise a control policy. The simulation case used to demonstrate performance was a continuous stirred tank reactor (CSTR) with the controlled variable being a dimensionless reactant concentration. Results showed the developed RL solutions outperforming conventional MPC for regulatory concentration control (Lee and Lee, 2005).

Simulation-based ADP has also been applied to a fed-batch bioreactor for obtaining optimal control of reactor feed under varying initial conditions (Peroni et al., 2005). In the presented method, simulation data generated by sub-optimal heuristic control laws are used to identify relevant regions of the process state-space and initialise a value function approximation. The initial approximation is then iteratively improved with a two hidden layer ANN receiving a state observation of 4 variables as input and outputting the corresponding value approximation. The Bellman equation (see Section 2.1.3) with backpropagation is used to update the network and hence improve the value function approximation. The obtained value function is implemented as an optimal controller, through state feedback, to optimise the feed profile of the bioreactor for maximum yield (Peroni et al., 2005).

More recently, a fuzzy inference system (FIS) has been used as a function approximator in conjunction with Q-learning as part of an ADP approach to adjust

PID controller parameters for optimal control output (Shah and Gopal, 2016). FIS is a decision-making framework based on fuzzy logic, which differs from boolean logic by considering uncertainty and imprecision. In a FIS structure, input values are placed in fuzzy sets with some degree of membership, determining the corresponding system outputs through an inference engine using a set of fuzzy input-output rules (Shah and Gopal, 2016). The use of a FIS for Q-function approximation, like the Q-network in deep Q-learning, enables the RL algorithm to be used with large state-action spaces, without having to store precise Q-values for every state-action pair. The developed algorithm is named dynamic fuzzy Q-learning (DFQL) and the resulting framework, described as model-free predictive control (MFPC), attempts to address previously mentioned challenges associated with MPC. Fuzzy Q-learning is an RL method used to tune FIS control outputs, where both control actions and Q-functions are inferred from fuzzy rules. In DFQL, fuzzy sets and membership functions dynamically adjust during the learning process. As a result, the system can adapt to changes in the environment and self-tune based on reinforcement signals. Iterations of the approximate Q-function are used to derive optimal PID gains based on optimality of actions in each system state (Shah and Gopal, 2016). A CSTR test environment was used to test validity of MFPC, where the concentration of a product is controlled by manipulating coolant flow. The results have indicated better control performance in MFPC optimised PID control compared to the regular MPC approach, especially in the presence of disturbances around nominal operating conditions (Shah and Gopal, 2016).

A limiting factor in the adaptation of RL-based control into industrial practice is the inability of most learning algorithms to satisfy state constraints. This causes often prohibitive risks in both training RL agents and operating pre-trained agents in industrial scale chemical processes. This challenge was addressed by a proposed oracle-assisted constrained Q-learning algorithm that guarantees the satisfaction of join chance constraints with a high probability. The novel approach of utilising self-tuned constraint tightening (back-offs), is a general methodology that can be introduced in ADP-based algorithms to ensure constraint satisfaction (Pan et al., 2021). In the proposed solution, an unconstrained agent is first trained using Q-learning with a Q-network function approximator. A surrogate constraint action-value function is also learned, which estimates the worst violation of a constraint that an action can cause in the future given the current state. These violation estimates are produced by Monte-Carlo simulations of the process. Subsequently, a constrained control agent is constructed by superimposing the unconstrained agent with the surrogate constraints. The control policy selects actions by optimising a fitness function comprising predictions of both the Q-network and the constraint violation network. After both networks have been trained, constraint tightening using back-offs is performed such that constraint satisfaction occurs with a desired probability while maximising future reward. Large back-off values ensure constraint satisfaction with a higher probability but renders the policy conservative and vice versa (Pan et al., 2021).

While function approximation and deep learning allows ADP methods such as deep Q-learning to handle high-dimensional state-spaces, value-based methods tend to be limited to low-dimensional and discrete action spaces. This is because the inherent logic of ADP methods relies on finding the action which explicitly maximises

the approximated action-value function (Lillicrap et al., 2016). Process control applications, however, are naturally more geared towards control schemes with high dimensional and continuous action spaces. Since discretisation of such action spaces result in exponentially growing solution complexity, policy optimisation has been researched as an alternative approach for RL-based process control in recent years.

2.3.2 Policy optimisation solutions in optimal process control

Policy optimisation strategies for sequential control tasks directly parameterise the control policy with a function approximator. The control policy, and hence action selection, is then optimised with respect to some objective function without consulting a value function. The natural advantages in this approach include the possibility of expressing optimal actions as probability distributions when optimal policies may be stochastic. Furthermore, expressing output actions as probabilities addresses the explore-exploit dilemma without the need for a heuristic exploration scheme. An action distribution as the policy output also naturally suits continuous action spaces which are common in process control, since actions are sampled from a continuous distribution. Furthermore, the direct policy optimisation strategy guarantees convergence to at least a locally optimal control policy, as opposed to ADP strategies where this is not guaranteed (Mowbray et al., 2022).

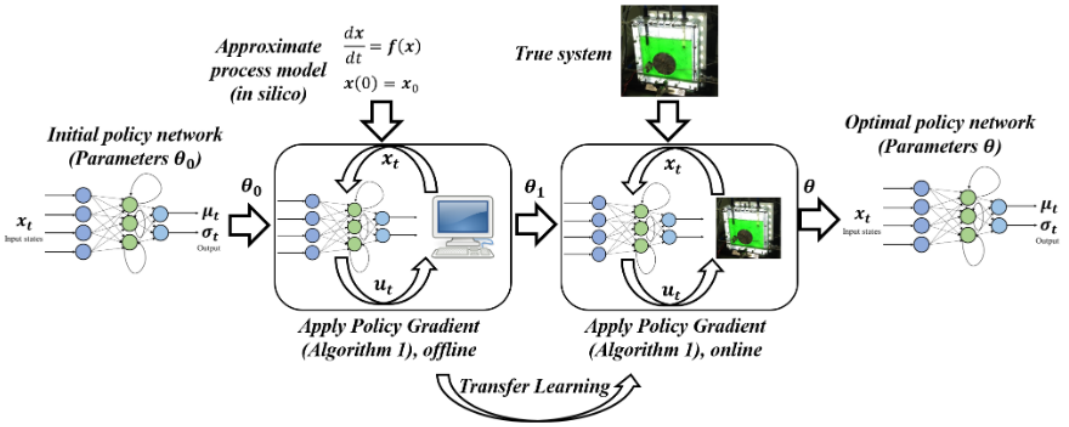

Figure 12: Transfer learning framework utilising policy iteration in a simulated environment and physical plant site (Petsagkourakis et al., 2020).

Recent publications in RL-based process control that leverage policy optimisation include an alternative approach to the optimisation of batch bioprocesses using the policy gradient method (Petsagkourakis et al., 2020). The introduced approach uses the REINFORCE algorithm (see Algorithm 1) and a recurrent neural network (RNN) for policy parameterisation to map process state to an optimal stochastic action based on product concentration. The output of the policy network is the mean and variance of an optimal action distribution, resulting in a stochastic policy. The developed framework uses a mechanistic model of the bioprocess to train an initial control policy

offline using generated simulation trajectories. The pre-trained agent is then introduced to the physical process through a phase of transfer learning, where all but the last hidden layer of the policy network is kept constant. This allows the control policy to address potential plant-model mismatch. After sufficient learning, the online policy update is terminated and an optimal RNN policy is deployed. The developed transfer learning framework is visualised in Figure 12. The performance of the RNN policy was benchmarked in an experiment where the goal is to maximise concentration of a target product with light intensity and an inflow rate as the manipulated variables. The policy gradient solution demonstrated that a near optimal control policy can be obtained even with unknown true system dynamics, outperforming a benchmark MPC solution (Petsagkourakis et al., 2020).

An alternative approach to pre-training a preliminary control policy before policy gradient optimisation in an industrial process was introduced in 2020 (Mowbray et al., 2021b). Instead of relying on a mechanistic model and simulation for "hot-starting" a control policy, a concept known as apprenticeship learning was used to learn an existing controller and a relevant objective function from historical process data. The case process in the published paper involves the control of the concentration of a reagent and the temperature of the reactor in a CSTR setup (Mowbray et al., 2021b). The proposed framework consists of two parts. In the offline preliminary training phase, inverse RL is employed to analyse historical process data and synchronously identify a reward function and parameterisation of the existing control policy in RL terms. This helps speed up training and improve data efficiency, since the subsequent RL phase does not need to begin from a randomly initialised naive control policy. After the pre-existing conventional control scheme (e.g., PID, MPC, or human control) has been identified as an RL policy network, the agent parameterisation is improved online using the REINFORCE policy gradient algorithm via transfer learning. The original controller is hence improved through RL, producing iterations of a stochastic policy network that outputs the optimal action distribution defined by the mean and standard deviation in the output layer (Mowbray et al., 2022). The results in the paper demonstrated successful improvements to an existing controller in a CSTR process unit given a sufficiently comprehensive and informative dataset of historical process data. The control policy improved using online REINFORCE displays quicker response times and lower over-shoot compared to the original controller (Mowbray et al., 2021b).

The idea of applying safety critical constraints to an RL control policy has been addressed in the context of policy optimisation methods (Petsagkourakis et al., 2022) using a similar approach to the ADP solution reviewed in Section 2.3.1 (Pan et al., 2021). The proposed chance constrained policy optimisation (CCPO) algorithm guarantees the satisfaction of joint chance constraints with high probability (Petsagkourakis et al., 2022). The approach is similar to oracle-assisted constrained Q-learning, achieving the goal through introduction of constraint tightening in the form of back-offs that are computed simultaneously with the feedback policy. In more abstract terms, back-offs restrict the feasible space perceived by the controller, affecting the state-space exploration of a learning control policy. The presented constraint handling methodology is devised as an additional feature for existing RL algorithms to make

them more suitable for safety critical process control applications. More specifically, CCPO addresses policy optimisation algorithms in this regard (Petsagkourakis et al., 2022), whereas oracle-assisted constrained Q-learning addresses ADP methods (Pan et al., 2021).

CCPO works by appending tightening constraints to the agent objective, removing the need for separate online optimisation to address constraint handling. The algorithm converges to an optimal policy that guarantees constraint satisfaction to a desired probabilistic tolerance. Similarly to the previous examples of policy optimisation control, the stochastic control policy is represented by a multilayer policy network that outputs the mean and variance of the control action. Policy optimisation is performed using the familiar REINFORCE framework, incorporating the novel constraint augmented objective function. The paper (Petsagkourakis et al., 2022) tested the practical performance of CCPO in a fed-batch bioreactor setting with two test cases where end product concentration was to be maximised. The results showed significantly better constraint satisfaction with CCPO compared to a nominal policy optimisation algorithm after 1000 Monte-Carlo trajectories (Petsagkourakis et al., 2022).

Although policy optimisation has inherent advantages compared to ADP techniques in process control, policy optimisation algorithms also have certain weaknesses limiting their usage. Policy gradient methods such as REINFORCE are on-policy algorithms and require a large number of sampled process trajectories to optimise a control policy. This contributes to poor data-efficiency in the policy optimisation approach, which can be unideal when dealing with computationally expensive process simulations and limited process data (Faria et al., 2022). Advances in RL theory have produced actor-critic algorithms capable of handling continuous, multi-dimensional actions spaces off-policy with improved data-efficiency. The combination of advantages from both ADP and policy optimisation approaches has resulted in a growing presence of state-of-the-art actor-critic implementations in the process control community.

2.3.3 Actor-critic solutions in optimal process control

Actor-critic-based RL process control relies on a policy network as the control agent or actor and ADP as a critic to iterate the control policy. The on-policy PPO algorithm (see Section 2.1.5) was used in a 2021 paper for optimising a solvent extraction process design (Plathottam et al., 2021). A largely unmodified generic PPO agent developed by OpenAI was used to optimise process design parameters in a simulated environment. Configuration variables constituting the design parameters included the number of solvent extraction stages and the operating variables of feed pH, extraction concentration and volumetric flows. The implemented PPO agents actor is in the form of a deterministic policy network that receives the state and process design parameters as inputs and outputs incremental changes in control action variables. The simulated process was optimised with respect to purity and recovery of final product. Results showed that the PPO agent generated designs were comparable to designs created by conventional optimisation (Plathottam et al., 2021).

In another application, PPO has been implemented in a multi-agent framework for

flatness control in a steel strip rolling process where metal strips are passed through rolling mills to reduce their thickness (Deng et al., 2022). In the developed ensemble proximal policy optimisation (EPPO) algorithm, multiple randomly initialised PPO actors interact with the environment in parallel. All actors are updated with experience from the actor obtaining the highest reward at each time step. The motivation behind using multiple parallel agents is to avoid falling into local policy optima caused by system dynamics or random initialisation of the PPO actor, which is a weakness of the original PPO algorithm (Deng et al., 2022). To evaluate the performance of the developed algorithm, the EPPO agent was first trained using production data from a real production site. The EPPO implementation for flatness control was then benchmarked experimentally against a conventional industry standard PI controller in a simulation designed to replicate the plant site. The results showed significantly better performance metrics in terms of flatness error margin obtained by the RL controller, when compared to the PI controller (Deng et al., 2022).

In another experimental study, the DDPG algorithm (see Section 2.1.5) has been utilised in a polymerisation process for optimal control (Yoo et al., 2021). The motivation in this case has been to solve issues related to NMPC-based optimal control in highly stochastic and challenging batch process control applications. DDPG is leveraged as part of the developed solution in order to handle high dimensional and continuous state and action spaces. The implementation modifies the original DDPG algorithm to use Monte-Carlo learning as opposed to temporal difference learning. The developed Monte-Carlo DDPG (MC-DDPG) algorithm updates the Q-network and policy network using return values obtained after the termination of a whole training episodes. As a result, MC-DDPG discards the target Q-network introduced in the DQN algorithm and carried over to the original DDPG (Yoo et al., 2021). This is because the original temporal difference approach updates the Q-network based on the Q-value estimate of the next state and not realised returns over entire training episodes (Lillicrap et al., 2016).

Since chemical processes may involve irreversible state transitions and require accurate terminal reward predictions very early to meet product specifications, Monte-Carlo learning has been hypothesised to be better suited for optimal process control even though convergence rate may suffer due to sparser policy updates (Yoo et al., 2021). Additionally, the paper (Yoo et al., 2021) introduces a novel phase segmentation approach to designing the reward function and function approximators to suit the distinct characteristics of phases during the production of a process batch. Common phases in batch processes include feed charging, processing and discharging. The optimal operation of each process phase is dependent on its respective purpose and overall process performance may hence benefit from tailored reward functions and agent designs (Yoo et al., 2021).

A case study for testing the optimal control characteristics of MC-DDPG has been conducted using a propylene oxide batch polymerisation process (Yoo et al., 2021). The case process involved significant non-linearity with both path and and terminal state constraints. The parameters of the actor and critic were initialised through supervised imitation learning against data obtained from a sub-optimal NMPC controller. The initialised agent was then deployed in a process simulation and trained using the

MC-DDPG framework. In the process simulation, path constraints included heat removal duty and adiabatic end temperature, while end-point constraints concerned a number of product specification variables. The controller manipulated variables were reactor temperature and propylene oxide feed rate. The case study involved varying reaction kinetics stochastically, and observing the subsequent constraint satisfaction of the controller. Results proved the hypothesis, showing that the RL controller was able to better account for process stochasticity and satisfy operational constraints compared to an NMPC controller (Yoo et al., 2021).

2.4 Solving current challenges in process industry RL applications

The inherent suitability of RL for process control has generated interest in industrial applications as research has accumulated in the field. However, persistent challenges exist concerning specific algorithm implementations and RL as a whole, hindering the adoption of RL-based APC as an industry standard. Nonetheless, constant improvements and iterations to RL control strategies are being developed, motivated by the potential to outperform mature technology and give an edge in the highly competitive contemporary market.

A significant challenge in deployment of RL control solutions in industry involves facilitating state-space exploration and learning through feedback, while satisfying safety critical constraints in a model-free implementation. In practice, a generic, randomly initialised RL algorithm cannot be deployed to an industrial process control application. The deployed controller needs to have some satisfactory base-line performance from the beginning, and exploration of the state space is to be bounded by relevant constraints during online training. These challenges are exacerbated by the fact that stock data-driven solutions and their dedicated infrastructures tend to be less reliable than traditional process control strategies and their systems (DCS). Furthermore, there is a general lack of understanding in the scientific community of how the components of DRL algorithms impact agent performance, training process and overall controller behaviour (Mowbray et al., 2022).

If controller initialisation is addressed using a simulated training environment, plant-model mismatch creates a potential challenge in deployment of the controller at a plant site. The experience-based learning framework needs to have been exposed to state transitions and process operating points in the training phase to be able to handle such situations after deployment. This further increases reliability risks and the need for constraint handling under changing process dynamics. It is worth noting, however, that changing process dynamics and parameters pose challenges also for legacy model-based optimal control solutions, where the underlying model used for feed-forward control needs to be recalibrated periodically to be kept up to date (Mowbray et al., 2021b).

At the algorithm implementation level, the RL training phase tends to be computationally expensive and data-hungry by nature. Although computational complexity is shifted offline compared to model-based strategies such as MPC which utilise online

optimisation, RL agents need significant amounts of data to converge on an optimal control policy. This issue further increases the need for an efficient initialisation strategy for RL controllers to reduce required online training. Additionally, RL algorithms have a high number of hyperparameters ranging from ANN architecture to exploration noise and experience replay mini-batch size. The relationship between these hyperparameters and agent performance is highly abstract and difficult to optimise, as effects are strongly dependent on application. Furthermore, ANNs are often poorly suited for handling raw unnormarlised data as inputs and outputs, requiring preprocessing and conversion to relevant scaling respectively (Sola and Sevilla, 1997). This can be difficult in the case of RL, where data is generated dynamically with no exact knowledge of the eventual distribution. As a result, the development of RL controllers can be tedious and unintuitive, requiring extensive trial-and-error, especially for those inexperienced in RL.

Depending on the process control objective, reward function engineering can be a challenging but critical aspect of a working RL controller, as it is the component that gives feedback to the agent on its performance. If there are multiple constraints, set points or economic variables, they all need to be accounted for in the reward function in a way that fulfills the control objective. Furthermore, nuances such as hierarchical objectives need to be considered if the optimal control solutions is required to satisfy compulsory safety constraints while optimising some less critical performance metrics.

During development of RL-based process control solutions, the aforementioned challenges need to considered at all levels of implementation design. The initialisation of the control agent can be done using a sufficiently realistic simulation environment or imitation learning using an existing legacy control solution. Regardless of approach, the deployment of the control scheme at a plant site requires a robust transfer learning framework to leverage the initialised agent effectively. The reliability and safety concerns in state-space exploration can be addressed with back-off-based bounding of feasible state space from the perspective of the agent to increase probabilistic safety margins. Alternatively, hard limits to agent actions can be programmed into control policy logic, based on defined limits to control actions. This is to prevent dangerous process operation even if the unbounded agent would be inclined to such behaviour during exploration of the optimal policy.

When initialising a preliminary controller, the design of the training scheme is critical for a robust controller capable of handling stochasticity and changing system dynamics after deployment. Since RL is based on learning through experience, the agent should be exposed to extensive experience during preliminary training. This includes comprehensive variation in operating points, system states and external independent variables, within the feasible ranges for the respective parameters. This way, in addition to hard coded operational constraints, the control policy would be better prepared for optimal control in a wider state-space envelope under changing external factors.

Solutions to algorithmic challenges are strongly based in theory. Although a wide range of open source RL algorithms exist, an effort should be made to understand comprehensively the underlying logic and structure of proven RL algorithms, since optimal implementation design is dependent on application domain and control

objectives. The theoretical understanding of RL algorithms also helps in comparing and evaluating data-efficiency and performance trade-offs between potential approaches. After settling on an implementation approach, the optimisation of a large number of hyperparameters should be done in a systematic framework based on a combination of logical reasoning and trial-and-error.

When applying an RL control algorithm to a specific control problem, it is critical to transform the process system into a corresponding MDP, requiring process knowledge as well as a good grasp on the desired control philosophy. Successful application of a control philosophy in an RL agent requires deliberative reward function engineering. Sub-reward weighting should be considered when there are multiple control and optimisation objectives. Additionally, the function governing the feedback reward of a single control variable should be thoughtfully constructed. For example, piecewise-defined functions can be a powerful tool for attaining desired reward behaviour when approaching optimality. The concept of reward function phasing can be expanded with respect to other domains such as time to account for process phases with different optimality criteria. Collectively, the discussed solutions to RL-based process control challenges highlight that although machine learning algorithms are considered black box solutions, their effective utilisation requires a strong grasp on theory, process knowledge and the underlying control philosophy.

3 Research material and methods

The development of an RL-based process control solution, as defined by the scope of this thesis, utilises a combination of proprietary tools available at Metso and open source tools popular in current RL state-of-the-art. The development process is iterative, gradually adding complexity to case simulations and the respective control problems. This way learning can be maximised during the development process while collecting useful results along the way.

3.1 HSC Chemistry

The training, deployment and testing of RL controllers in this project is carried out using HSC Chemistry software (Roine, 2023) (referred to as HSC in this thesis) in simulated chemical processes. HSC is a chemical calculation and process modelling platform, owned by Metso Oyj, that can be used for the simulation of a range of process applications. The current version of HSC is the culmination of decades of development that began in 1970s. In addition to featuring a large chemical property database on thousands of chemical compounds and minerals, the software is able to perform advanced thermodynamic calculations required for the process modelling of pyrometallurgy, hydrometallurgy and minerals processing. HSC chemistry has a total of 24 separate calculation modules with a range of functionalities ranging from process simulation to heat, material, and energy balance calculation (Roine, 2023).

HSC Sim is one of the available modules under the HSC umbrella, used for spread-sheet-based process flowsheet simulation. The module allows individual unit simulations to be connected via streams, enabling the virtual operation of complex full-scale processes. Individual process units have corresponding spreadsheets where process variables and calculations are set.

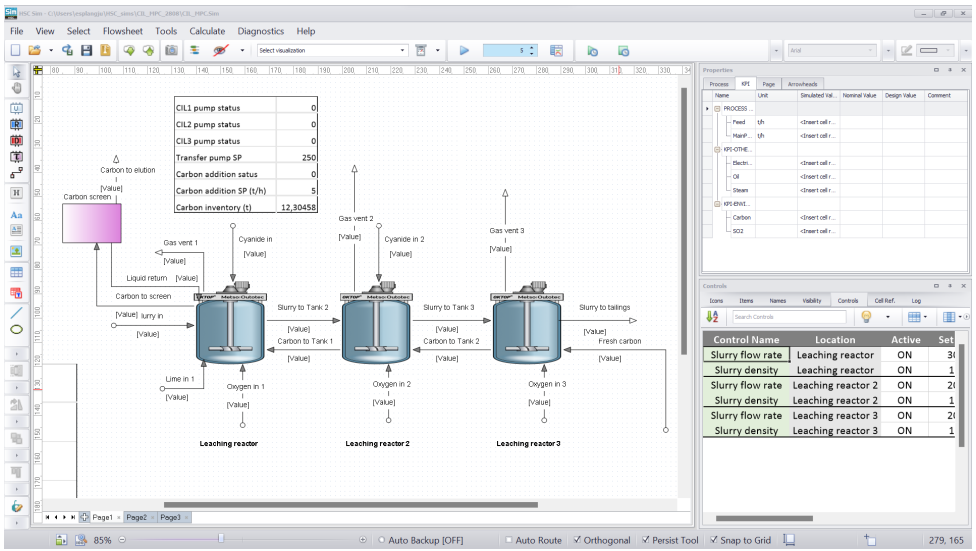

Figure 13: HSC Sim flowsheet user interface for the simplified CIL process case study.

In addition to static simulation units that consider processes at steady state, dynamic units based on differential equations exist in HSC for the construction of transient process simulations. In this thesis, dynamic simulations are used as environments for the developed RL controllers, since process control is concerned with dynamically changing process states. The software utilises numerical integration methods to solve differential equations of mass, energy, element, and fraction balances in simulated processes. HSC has a graphical user interface displaying process units as part of a flowsheet in addition to various dialogues where the user can observe important process data in numerical or graphical form. A screen capture of the user interface is shown in Figure 13. Furthermore, time-variant output data can be collected from the simulation into user-defined variables and time-variant input data can be defined through standard spreadsheet formulae all within the simulation file in separate spreadsheet tabs (Roine, 2023). The methodology with which HSC Sim is used to train RL agents and how trained agents are deployed to simulated processes are outlined in Section 3.3.

3.2 Machine learning with PyTorch

The RL process control algorithms developed in this thesis use Python (Van Rossum and Drake, 2009) as the programming language of choice and the PyTorch library (Paszke et al., 2019) for practical implementation of deep learning using ANN function approximators. PyTorch is an open-source machine learning framework developed by Facebook's AI Research lab (FAIR), currently known as Meta AI. In essence, PyTorch is a Python library designed to perform dynamic tensor computations for deep learning using GPUs and CPUs. Namely, it provides the user with multi-dimensional arrays called tensors, which are the key elements for creating and manipulating data in neural networks (Paszke et al., 2019).

In addition to tensors, key features of PyTorch are the dynamic computational graph and neural network modules, which enable developers to create neural network structures with ease. A wide range of additional features make PyTorch an excellent tool for deep learning, including Autograd, which facilitates gradient computation for optimising neural network parameters using techniques such as gradient descent (Paszke et al., 2019).

3.3 RL framework for agent training and policy deployment in process simulations

The developed RL algorithms in this project are designed such that during training they run locally in a Python IDE, communicating in real-time with a HSC simulation. In the training loop, the Python script, acting as the agent, sends control signals to the process simulation, acting as the environment, and receives a resulting state observation in exchange. The observation is used in conjunction with the reward function defined in the script to improve on the control policy. The training loop can be run indefinitely until satisfactory control performance is achieved with the iterated policy.

Reading of simulation data and writing of control output data inside HSC Sim is done in real time using a proprietary Python-HSC API (application programming interface). The API is able to read specific data variables defined in the simulation file, advance the simulation by a set time interval and overwrite control action variables using a set of functions accessible by the Python script. Using the features of the API, a control signal can be sent to the simulation, the process can be advanced in time and the effects of said control signal can be accessed by the RL agent in the form of a state observation. The Python-HSC API allows an RL agent implemented in an external Python script to learn an optimal policy to control a HSC simulation, forming the basis for the scope of this thesis. The trained agent is represented by a policy network in the implementations discussed in Chapter 4 (REINFORCE and DDPG). The policy network is an ANN which receives process state observations as inputs and produces either a corresponding stochastic control action distribution (REINFORCE) or a set of deterministic control actions (DDPG) as output.

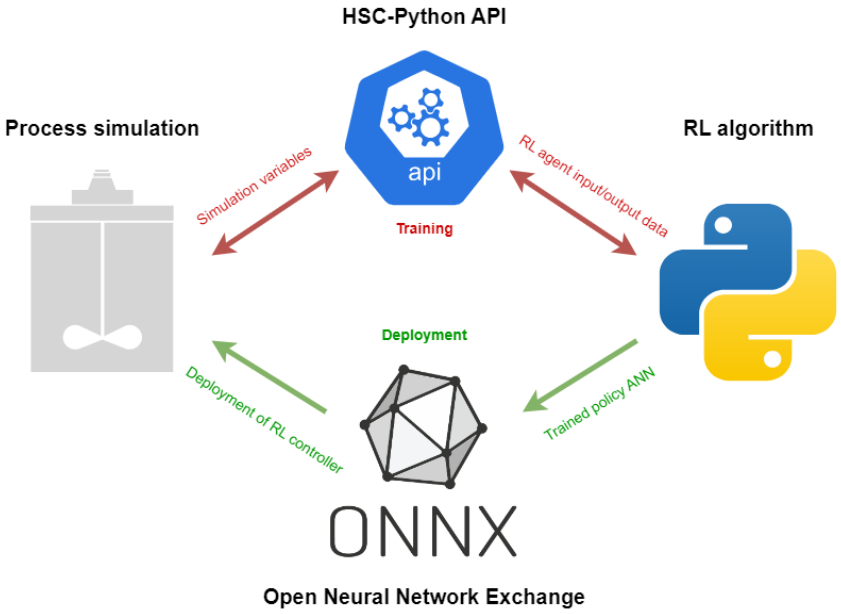

Figure 14: Developed framework for training and deployment of an RL process controller.

The deployment of trained policy networks to HSC simulations is done using the Open Neural Network Exchange (ONNX). ONNX is an open-source conversion format designed for interoperability between deep learning models and tools. It facilitates creation, sharing, and deployment of machine learning models across different platforms and hardware solutions, through a common format for representing trained models (Openja et al., 2022). The significance of ONNX in the context of this thesis, is that it enables deployment of an externally trained and saved control policy in the HSC simulation environment. ONNX machine learning model functionality in HSC has been developed at Metso to facilitate this kind of agent deployment. The ONNX ANN can be imported into a HSC spreadsheet, where dynamic data variables

corresponding to the state observation variables can be assigned as input nodes. The imported ANN calculates outputs into spreadsheet cells in real time, which can be used to overwrite desired control action variables in the dynamic simulation. A trained RL control policy can hence be integrated into a HSC dynamic simulation to control the simulated process. Control performance can be evaluated by observing process values and KPIs via the HSC Sim user interface. A schematic diagram of the developed RL framework for controller training and deployment can be seen in Figure 14.

3.4 Problem formulation

With the development framework in place, focus can be shifted towards algorithm development. The first step in the development process requires defining process control as a machine learning problem in the RL paradigm so that it can be solved with a suitable algorithmic approach. Refer to Section 2.2.4 for a more theoretical analysis of RL as a potential process control solution. The control task of interest has to be defined as a sequential decision making problem in the form of a MDP that maintains all relevant information from the chemical process.

The proof-of-concept experiments and case study in this thesis consider the regulation control problem, where process values are to be shifted to, and kept at defined set points in an optimal manner. In MDP terms, states in the process regulation problem are represented by state observations consisting of relevant process information, including but not limited to process values of controlled variables and respective set points. Actions in the MDP are represented by variables that the controller can manipulate in order to control the relevant process values. Definition of the aforementioned components in a problem-specific process control MDP requires process knowledge and understanding of the control philosophy.

The process simulation transitions the MDP system to a new stochastic state, determined by control actions chosen and process dynamics. The final component of the MDP is the reward received for taking a control action and observing the subsequent state. The reward is defined by a reward function, designed to reinforce desired actions and consequent states. In the process regulation problem, the reward function needs to be constructed such that higher rewards are given as process values approach set points. In the case of optimal control, a component needs to be added to the reward function which rewards desired objectives and penalises undesired outcomes. RL process control is effectively about developing an algorithm implementation that is able to solve this MDP.

The solution to the process control MDP is an optimal control policy which maps process states to control actions, maximising received reward. After setting up a framework that facilitates the training and deployment of a control policy, the remaining challenge is to implement an RL algorithm capable of iterating such a policy. With an abundance of open source solutions to choose from, features of existing algorithms should be considered in terms of suitability to the relevant control problem. Such features include the adaptability to continuous and multi-dimensional action spaces, data efficiency, and stability.

3.5 Development methodology

The beginning of the algorithm development phase is characterised by an initial lack of experience in RL control solutions, little available implementation documentation in literature, and an untested development framework. Consequently, an iterative development strategy is devised consisting of proof-of-concept experiments with gradually rising levels of complexity. The idea behind the strategy is to learn about the practical execution of the designed RL control framework, the simulation-algorithm interface, RL algorithm implementation, and hyperparameter optimisation in simple test cases before complicating the control problem.

In the first proof-of-concept test case (POC1), a simple simulation is constructed around the dissolution of NaCl in water. The single-input single-output (SISO) control of salt concentration in the tank is chosen as the regulatory control problem. This particular problem is chosen as the initial proof-of-concept experiment due to simple dynamics and a small number of variables, resulting in a fast simulation time and simple control philosophy. Combined with the resulting low required iteration count for convergence, POC1 is ideal for agile testing and debugging of integral solution features.

The Monte Carlo vanilla policy gradient method, known as REINFORCE, is tested as the algorithm for POC1. Reasons include the simple implementation of only one ANN for the policy approximator, successful trials in literature and the generally desirable features of policy gradient methods with respect to process control. The RL controller training and deployment framework is tested and validated using POC1. Furthermore, general algorithmic challenges and bugs are solved concerning simulation data normalisation, control action representation and scaling, as well as reward function design and agent training strategy.

After successfully deploying a control policy trained using REINFORCE, a new test case is explored. POC2 uses the familiar training and deployment framework from POC1 but incorporates a different RL algorithm for iterating the control policy. DDPG is implemented for learning a policy for controlling the SISO regulatory control problem as in POC1. DDPG is investigated because of potential for better data efficiency and handling of multi-dimensional and continuous action spaces simultaneously. The latter feature is especially relevant when considering future experiments with more complex MIMO control problems. Because of greater algorithmic complexity with the introduction of a critic and a more sophisticated training process, the focus in POC2 is to develop a working DDPG implementation in a simple test scenario. This is straight forward since the other components and interfaces of the RL framework would have already been familiarised in POC1.

After testing the feasibility of the DDPG process control approach in POC2, a second case process simulation is constructed in HSC, incorporating energy balance in the form of a heating coil used to adjust the temperature of the familiar NaCl dissolution tank. The updated control objective in POC3 is to control both salt concentration and temperature in the tank using a single RL control policy. The devised control philosophy is to control tank temperature by adjusting flow of heat-transfer fluid through the heating coil, while adjusting salt concentration through NaCl feed rate.

The endothermic property of the reaction inside the tank adds further complexity to the dual regulatory control problem since the rate of dissolution, and hence salt concentration, is correlated to temperature. After testing and validating MIMO control using a trained policy in POC3, the final case study is commenced.

While the POC experiments are used as stepping stones to develop implementation strategies, a final case study is designed to explore the potential of RL-based control in industry applications. The case process simulation is a carbon-in-leach (CIL) process for gold recovery from slurry in three sequential reactors. The main control objectives are to regulate dissolved oxygen in each reactor through individual oxygen feeds and remaining cyanide tailings in the last reactor through cyanide feed into the first and second reactor. In addition to being representative of a real industrial process (albeit in a smaller scale) and hence an interesting case study, the CIL simulation adds further complexity in terms of state variables, dynamics and controlled variables. The DDPG implementation from POC3 is adapted to facilitate the new process simulation and its characteristics. The development focus in the final case study is on algorithm optimisation and hyperparameter tuning because learning the more advanced control policy involves significantly higher computational complexity than in the POC experiments, requiring a more optimised algorithm.

4 Results

This chapter documents the progression of test cases and the associated algorithmic solutions, presenting the results obtained from agent training and policy deployment in process simulations. Results are analysed in terms of learning performance, describing the convergence of policy iteration, and control performance, describing how well the learned control policy fulfills the intended control objective.

4.1 Proof-of-concept 1: SISO control using REINFORCE

4.1.1 Case process 1: Mass balance regulation in NaCl dissolution

To test the feasibility of training an RL-based process controller in the devised implementation framework, a simple case process was constructed. Case process 1 is made up of a singular chemical reactor where dissolution of sodium chloride (NaCl) in water (H_2O) takes place. The ionisation of sodium chloride molecules in water can be expressed with the following equation:

$$NaCl_{(s)} \xrightarrow{H_2O} Na^+_{(aq)} + Cl^-_{(aq)}. \tag{31}$$

Water and sodium chloride are fed into the reactor and solution volume is kept constant using a proportional discharge rate. Water feed is an independent variable which fluctuates randomly, adding an element of stochasticity to the process. The control objective for POC1 is to regulate solution concentration, specifically by controlling the chloride ion (Cl^-) concentration (mol/L) inside the reactor through manipulating sodium chloride feed (ton/h).

4.1.2 Algorithm development

The policy gradient approach in the form of REINFORCE was used as the RL algorithm in POC1. The pseudocode (Algorithm 1) and algorithm description in Section 2.1.4 was used as a base for the implementation. The reader is advised to refer back to that section for a more in-depth explanation of algorithm logic.

The developed REINFORCE implementation consists of four Python files responsible for different functionalities. One file contains the class that defines the policy network and the class containing methods to select an action based on the output of the policy network, as well as update the network weights using backpropagation. Another file contains the reward function which returns the reward on each simulation time step, and a main function responsible for initialising and running the REINFORCE training loop. The main function utilises methods from the HSC-Python API, located in a third file to facilitate communication with the process simulation. The functionality to export a policy network in ONNX file format is implemented in a fourth file.

The policy network is designed to have four nodes in the input layer, consisting of process values of the two reactor input streams, the chloride concentration process value and the chloride concentration set point. These four variables represent a state observation received from the simulation. The state observation vector is normalised

using a heuristic standard score normalisation with estimated population parameters as it was noted during development that normalised inputs result in better learning performance. The output layer consists of two nodes, representing the probability distribution of the incremental change in reactor sodium chloride input in the form of mean and variance. The control action selection method samples randomly from this distribution, producing a stochastic policy which is suitable for dealing with continuous action spaces and satisfying the need for state space exploration during learning. The sampled control action from the policy output represents a normalised numerical increase or decrease to the sodium chloride input stream value of the previous time step. It was noted during development and testing that controlling normalised change in the manipulated variable produced the best policy iteration results, as opposed to controlling the normalised whole input stream value. In addition to the input and output layers, the policy network contains four hidden layers of 12 nodes each.

Some minor modifications were made to the original vanilla policy gradient algorithm in the POC1 implementation. The policy update rule was modified such that each reward in the return sequence of a trajectory was normalised before applying the policy update in Algorithm 1. Variance is reduced in the sampled trajectories as a result, resulting in better learning stability. The goal of this algorithm modification is closely related to REINFORCE with baseline, as discussed in Section 2.1.4, which also strives to reduce variance and hence stabilise learning.

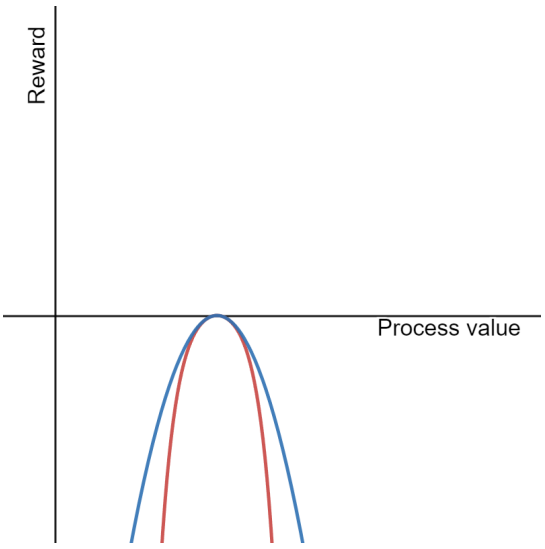

Figure 15: Comparison of a quadratic reward function (blue) and a hyperbolic sine reward function (red) where the set point of a hypothetical regulation problem is at the x-intercept.

As POC1 deals with a SISO regulatory control problem, the reward function should return a higher reward as the process value approaches the set point. The initial tested reward function was a negative squared error of controlled variable compared to set point. This gave a quadratic reward function, approaching zero as the maximum reward. In an attempt to punish undesirable process states and actions more, a negative

hyperbolic sine function of the quadratic error was explored. Two of the tested reward functions are visualised in Figure 15.

4.1.3 Agent training

The POC1 RL agent was first trained using a constant chloride concentration set point over each training episode. The goal was to investigate convergence properties of the algorithm under a constant set point and stochastic outside disturbance in the form of a fluctuating water feed. Episode length was set to 300 time steps, representing a single sampled trajectory. The results showed that the REINFORCE algorithm is able to learn a control policy to maintain a constant reactor concentration under stochastic process disturbance as seen in Figure 16. The corresponding learning curve, visualising policy convergence, can be seen in Figure 17.

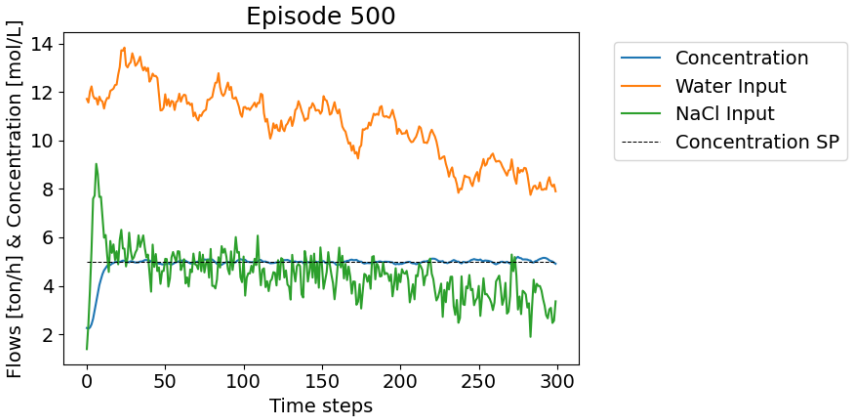

Figure 16: Control performance of REINFORCE agent for POC1 during training with static set point.

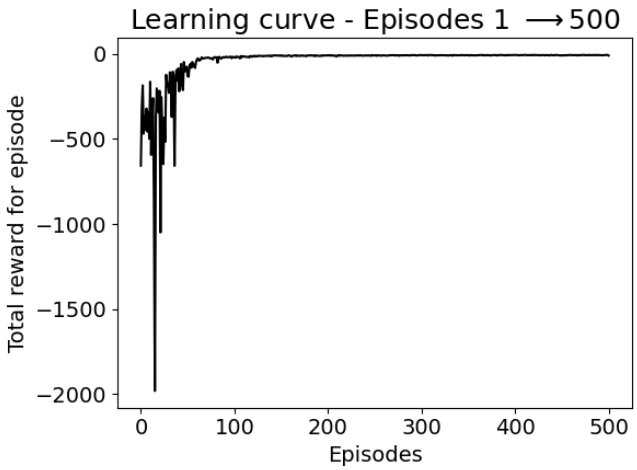

Figure 17: Learning curve of REINFORCE agent for POC1 with static set point.

With promising convergence properties observed in the initial static set point trials, a strategy was needed for training the control policy to handle shifting set points. A sinusoidal set point function was designed with a randomly varying amplitude, frequency and centre line between episodes. This way the RL agent would learn how to handle a range of states, transitions, and set points during training. Naturally, convergence became slower with the introduction of episodically changing sinusoidal set point functions but the POC1 algorithm was able to converge on an optimal control policy nonetheless as seen in Figure 18. The corresponding learning curve is displayed in Figure 19.

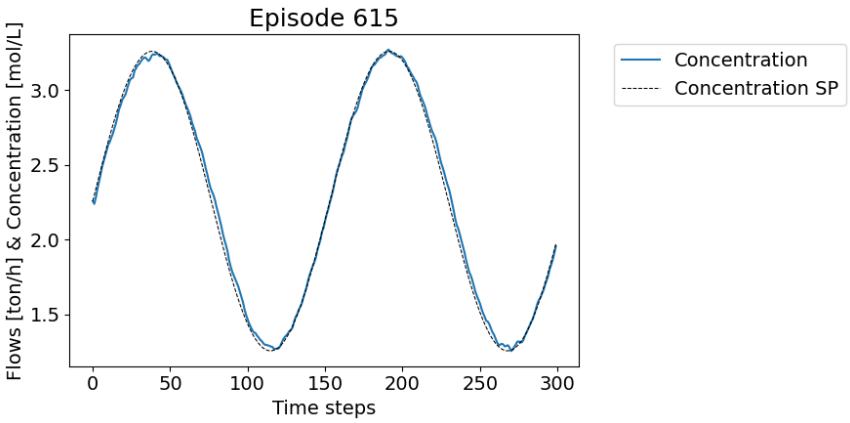

Figure 18: Control performance of REINFORCE agent for POC1 during training with shifting set point.

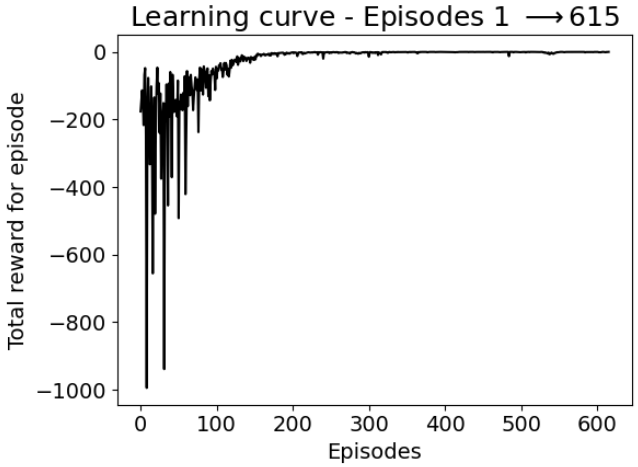

Figure 19: Learning curve of REINFORCE agent for POC1 with shifting set point.

4.1.4 Controller deployment in simulated environment

After convergence of the control policy during the training phase and the associated plateauing of episode scores, a trained policy network can be tested in the HSC

simulation environment as a deployed controller. The network is imported into the HSC simulation spreadsheet as an ONNX file. The input layer nodes are linked to state observation variables and the node representing the mean of the stochastic control action output is added to the current sodium chloride feed variable. This way the manipulated variable in the simulation is updated with a deterministic action selection, as exploration in policy behaviour is no longer required after ending the training phase. The performance of the controller can be tested by manually manipulating the concentration set point in the reactor and seeing the response of the process variable. The results in Figure 20 show stable and reliable regulatory control performance produced by the trained control policy.

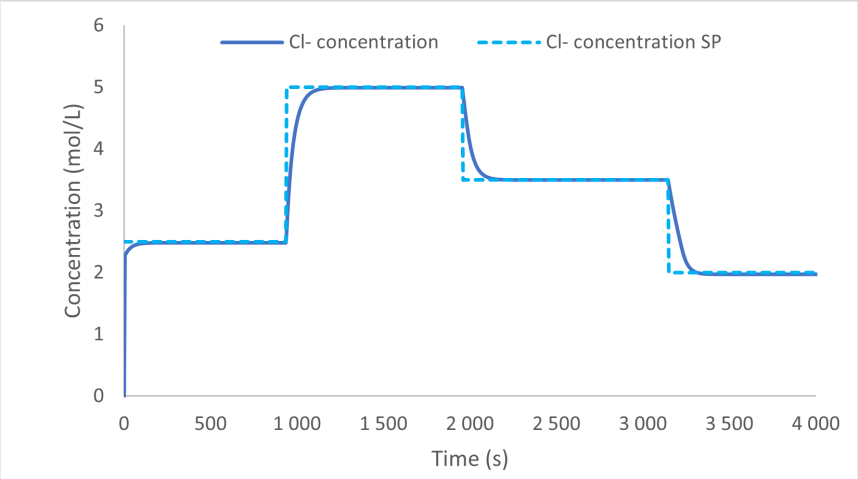

Figure 20: Control response of deployed POC1 policy in simulated environment.

4.2 Proof-of-concept 2: SISO control using DDPG

POC1 was able to demonstrate the potential of RL-based control combined with the devised training and deployment framework in a simple regulatory control problems using the vanilla policy gradient approach. POC2 proceeded to explore the actor-critic approach with DDPG, due to its potential in improving data-efficiency and addressing multi-dimensional continuous action spaces in more complex process control problems. The same case process from POC1 is used to test the feasibility of the DDPG algorithm for SISO process control. Case process 1 also allows benchmarking the performance of DDPG against REINFORCE in a common environment, before attempting more complex control problems.

4.2.1 Algorithm development

The original DDPG algorithm introduced in Section 2.1.5 and outlined in Algorithm 2 was used as the base for the POC2 implementation. Once again, for a more in-depth analysis of the algorithm, the reader is advised to refer back to the relevant section.

Compared to POC1, the POC2 algorithm is significantly more complex in terms of implementation, as is expected when examining the pseudocode representations of the two algorithms. The DDPG solution introduces the Q-network as a critic, requiring an additional set of software functionalities. The implementation consists of five core Python files with distinct roles. The actor and critic networks are defined as classes in one file. The DDPG agent is implemented in its own separate file as a class containing all required methods to run required steps in the algorithm. This includes the training loop, choosing the control action, and updating network parameters of the actor and critic. The replay buffer is also defined as a class in a separate file. The reward function and main function are defined in a file which is responsible for running the algorithm during training by defining class instances and calling on various algorithm functionalities in other files. The HSC-Python API file from POC1 is also included in the implementation to allow communication with the process simulation during training.

The actor network was designed like the policy network in POC1, receiving a state observation in the input layer and mapping it to an action in the output layer. The actor in POC2 differs by increasing the size of the state observation to 6 with the addition of outflow rate of the reactor and reactor mass. This was done to utilise all available information in the control problem. Furthermore, since DDPG is a deterministic policy gradient method, the output layer for the actor in POC2 contains only 1 node, representing the deterministic action given by the policy. The critic network has a 7 node input layer, receiving the state observation and the action given by the actor. The output layer consists of a singular node, giving the Q-value approximation of the action taken. The state observation data fed into the network and the control output received from the actor network are normalised and scaled with the same logic as in POC1.

Some minor changes were made to the original DDPG algorithm. The size of the two hidden layers in the actor and critic networks were decreased to 100 nodes, compared to 400 for hidden layer 1 and 300 for hidden layer 2 in the original implementation. The Ornstein-Uhlenbeck process noise, used for the exploration policy in the original DDPG algorithm, is replaced by a simple Gaussian random noise component in the action choice method which decays over episodes, emphasising exploration in the beginning. It was deemed that the inclusion of an Ornstein-Uhlenbeck process in a separate class would add unnecessary complexity to the implementation. The replay buffer size was set to 10 000 transitions and the sampled mini-batch for network updates was set to 120 transitions. Other hyperparameters were largely kept the same as in the implementation featured in the original DDPG paper (Lillicrap et al., 2016).

The reward function used in POC2 was improved from the POC1 implementation. It was noted that since the gradient of a quadratic or hyperbolic sinusoidal reward function approaches zero when nearing the set point, high accuracy is perhaps not rewarded sufficiently. A clipped reward function was introduced, which transitions from a quadratic reward to a linear reward towards a positive maximum, after process value crosses a threshold. This type of reward function introduces positive rewards close to the set point, incentivising precise control. The altered reward function is visualised in Figure 21.

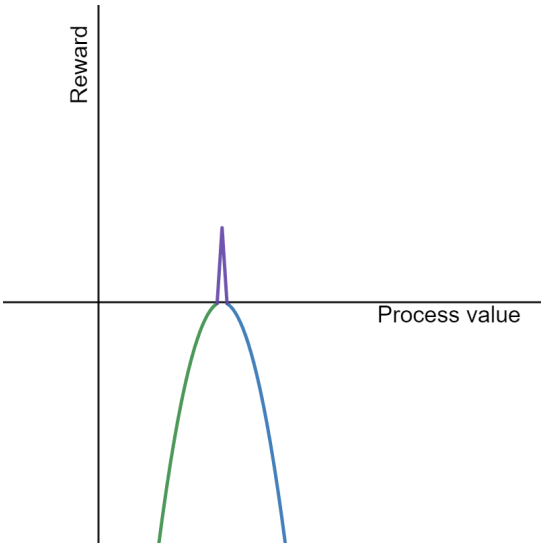

Figure 21: Clipped reward function. Function parameters can be tuned to change threshold and maximum reward.

4.2.2 Agent training

The DDPG agent was trained using a similar approach as in POC1 by utilising a sinusoidal shifting set point. A modification to the training scheme was made with a hard upper limit at 6 mol/L for the chloride concentration set point. This was done in consideration of the natural solubility of sodium chloride in water, to avoid unrealistic and hence irrelevant set points during training. The POC2 DDPG algorithm confirms the theoretical data-efficiency advantage over REINFORCE by converging significantly faster during training as seen in Figure 23. The utilisation of the clipped reward function in POC2 causes the different y-axis scaling and overall shape in the learning curve compared to POC1. However, a comparison can still be made between convergence rates by examining the episode count on the x-axes. The convergence of the POC2 agent is corroborated by examining the policy performance during the training episode 50 in Figure 22.

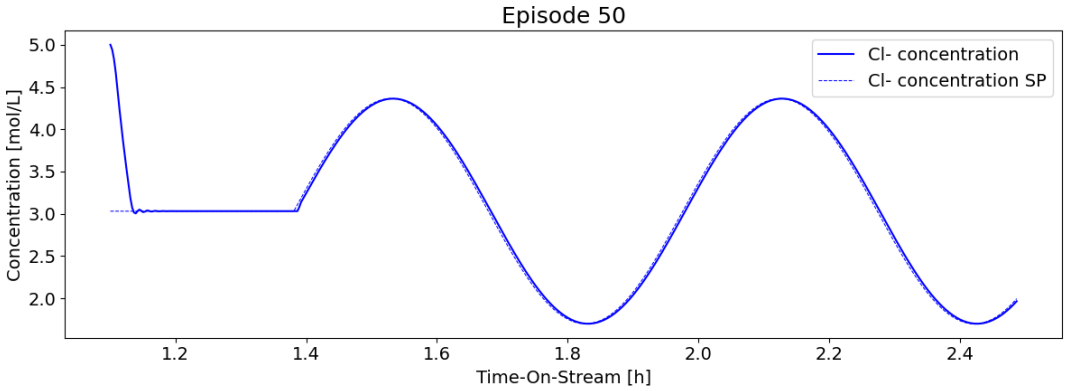

Figure 22: Control performance of DDPG agent for POC2 during training.

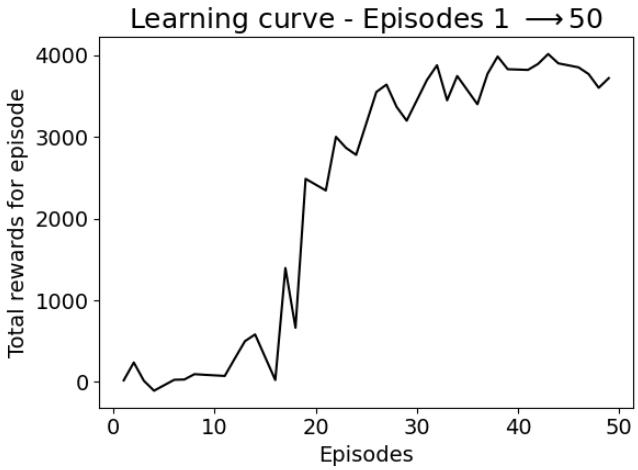

Figure 23: Learning curve of DDPG agent for POC2.

4.2.3 Controller deployment in simulated environment

To test the performance of the trained POC2 control policy, it was deployed in the HSC simulation using the same procedure as in POC1. The results are comparable, showing a robust and stable controller with rapid response time. It is worth noting that the deployed policy exhibits some oscillation in the response and overshoot in the first set point shift as seen in Figure 24. This behaviour is likely and artefact of the particular policy iteration used and further training could produce better results.

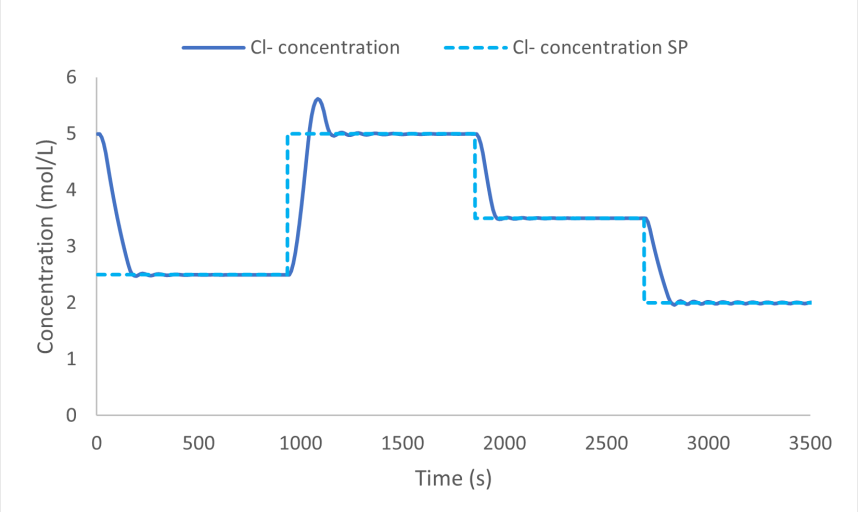

Figure 24: Control response of deployed POC2 policy in simulated environment.

4.3 Proof-of-concept 3: MIMO control using DDPG

POC2 results demonstrated the DDPG algorithm as a valid approach to training an RL policy for simple regulatory process control, outperforming the classic REINFORCE

algorithm in both precision and convergence rate. POC3 continues development of process control solutions using DDPG, applying the actor-critic algorithm to a multi-dimensional action space in the form of a MIMO control problem. In practice, this is done by introducing a second parallel control objective, consisting of separate controlled and manipulated variables. A new process simulation is required to set up the MIMO control problem designed to test more advanced capabilities of DDPG.

4.3.1 Case process 2: Mass and energy balance regulation in NaCl dissolution

The new test simulation uses case process 1 as a starting point but new dynamics and variables are added. In addition to sodium chloride dissolution and associated reactor input feeds, energy balance is introduced in the form of a heating coil used to raise temperature of the reactor. Temperature is controlled through manipulation of heat-transfer fluid flow in the coil. The endothermic dissolution reaction taking place in the reactor lowers solution temperature without the introduction of heat energy from an outside source.

The control objective in POC3 is to regulate both chloride ion concentration inside the reactor and reactor temperature according to arbitrary set points using a single RL control policy trained using the DDPG algorithm. Due to the endothermic nature of the dissolution reaction, temperature, and ion concentration are correlated, adding a level of difficulty to the control problem.

4.3.2 Algorithm development

Due to the modularity and generic nature of the solution developed in POC2, the structure and implementation of the algorithm is kept almost identical in POC3. This includes algorithm hyperparameters, file structure, and logic. The actor and critic networks have been modified to accommodate the different control problem and application. Five new state observation variables are added as nodes to the input layer of both networks, comprising the heating coil temperature, heat-transfer fluid temperature, heat-transfer fluid feed rate, reactor temperature, and the reactor temperature set point. A node representing the second control action variable, responsible for increasing or decreasing heat-transfer fluid flow, is added to the output layer of the actor network and to the input layer of the critic network. Hidden layers dimensions are maintained from POC2.

The clipped reward function from POC2 is utilised again but this time in two sub-reward functions responsible for rewarding the two regulation problems separately. The two reward components are added together with equal weights to produce an overall reward function used to give feedback to the RL agent.

It was noticed during algorithm development that with two actor outputs, the algorithm performance is highly dependent on the appropriate normalisation and denormalisation of control action data. In order for the algorithm to learn optimal control inputs for two controlled variables simultaneously, the policy network outputs need to be scaled similarly. Without correctly tuned normalisation parameters, the

DDPG algorithm has a tendency to learn the regulation of only one of the controlled variables, due to inconsistently normalised policy outputs. The process of tuning normalisation parameters of network inputs and outputs involves examining value ranges of respective variables as well as trial-and-error.

4.3.3 Agent training

The now familiar training philosophy of using sinusoidally shifting set point functions, to expose the exploration policy to varying state transitions, was used again. The respective set point shift functions for the two process variables were customised to suit realistic value ranges with lower and upper bounds. Due to the correlation between controlled variables and more complex dynamics, the exploration policy was slower to converge compared to POC2. Nonetheless, the POC3 DDPG algorithm was able to learn a multi-action deterministic policy to regulate two correlated process variables, fulfilling the intended goal.

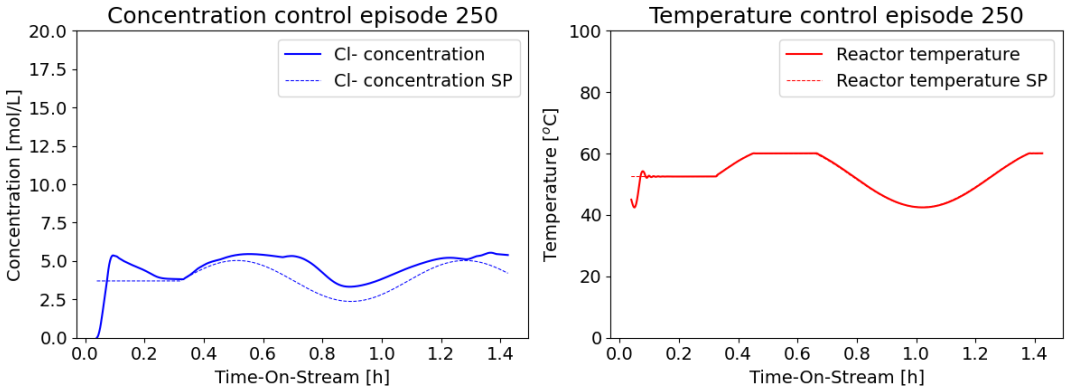

Figure 25: Control performance of partially converged DDPG agent for POC3 during training.

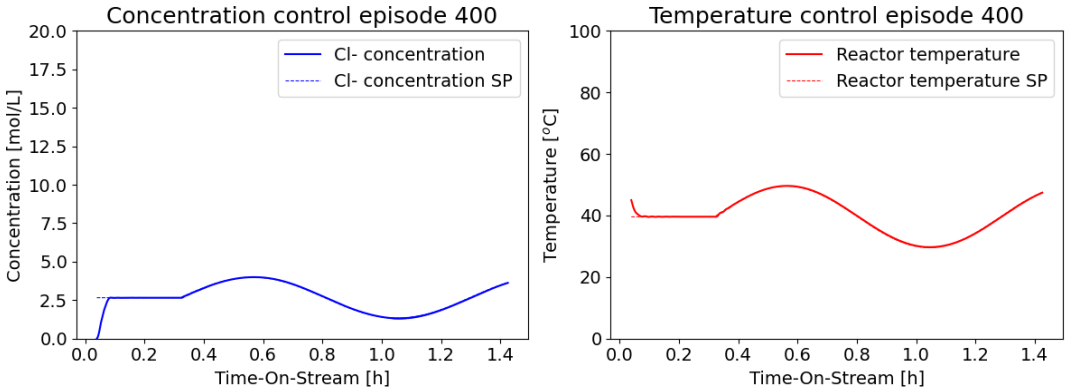

Figure 26: Control performance of fully converged DDPG agent for POC3 during training.

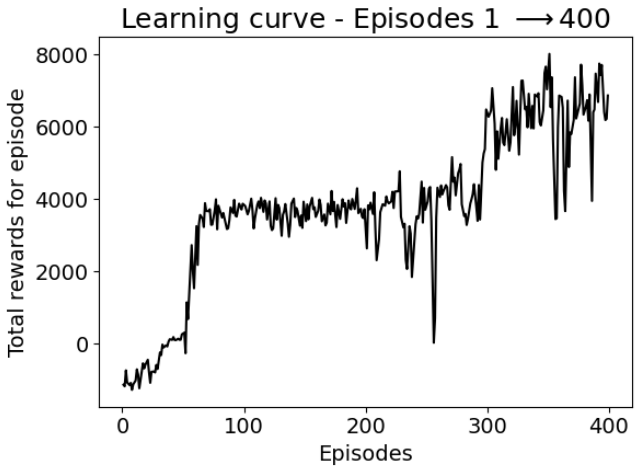

Figure 27: Learning curve of DDPG agent for POC3.

The POC2 algorithm seemed to learn the regulation of the two controlled variables in sequence. This is evident when examining the learning curve in Figure 27. The graph has two sections of rapid increase in total episodic rewards, corresponding to the algorithm first learning how to regulate temperature and then concentration. The performance during training of the fully converged policy can be seen in Figure 26. In comparison, the performance of the control policy during training before having learned concentration regulation can be seen in Figure 25.

4.3.4 Controller deployment in simulated environment

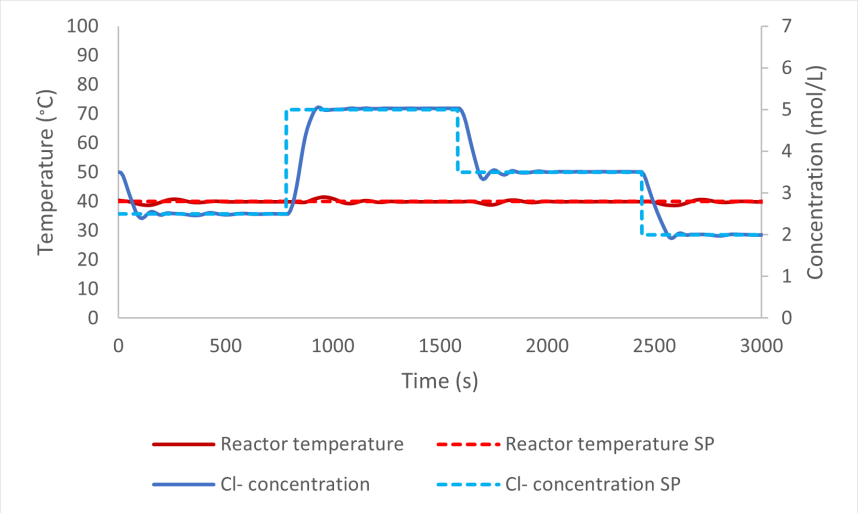

Figure 28: Control response of deployed POC3 policy in simulated environment with shifting concentration set point. Reactor temperature set point is kept constant.

The deployment of a trained control policy was done using the same procedure as in POC2 previously. The additional output layer node in the policy network is used to

update the heat-transfer fluid flow at each simulation time step. Through examining control response in HSC, it becomes evident that the trained control policy is able to successfully regulate two controlled variables simultaneously. Additionally, the controller is capable of reliably compensating for a shift in set point of a correlated process variable to keep the process value of the other controlled variable constant. The control responses of the deployed POC3 policy can be examined in Figures 28 and 29.

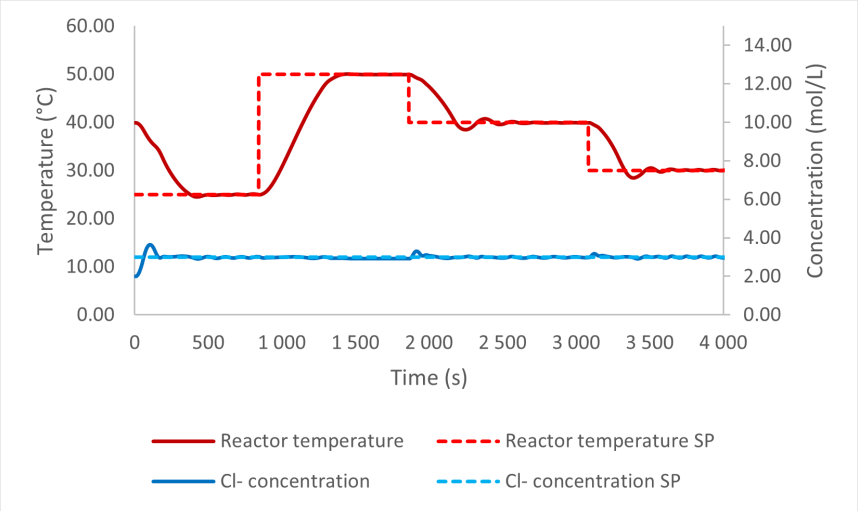

Figure 29: Control response of deployed POC3 policy in simulated environment with shifting reactor temperature set point. Concentration set point is kept constant.

4.4 Case study: Control of carbon-in-leach process using DDPG

POC3 proved the capability of the DDPG algorithm for controlling multiple regulation problems simultaneously. The associated development process and collected experimental results gave the implementation experience and assurance needed to consider applications in more complex simulations of existing industrial processes. The final case study of this thesis involves the optimal control of a carbon-in-leach (CIL) process used commonly for gold recovery in industry. An existing process simulation in HSC is used as the test environment during development. The simulation has previously been used as a testbed for MPC-based optimal control solutions.

4.4.1 Case process 3: Carbon-in-leach regulation and optimisation

The process simulation used in the CIL case study is significantly more complex than the test environments used in the POC experiments. Process slurry flows through three consecutive unit functions in the form of leaching reactors via connected streams. Each reactor has a number of input feeds which are manipulated to control process state.

The exact chemistry of the CIL process is outside of the scope of this thesis, so focus is on controlled variables and their interactions. The control objective in the CIL process is concerned with the amount of oxygen (O_2) and cyanide (CN^-) present in the functional units. Oxygen is fed into each reactor and the resulting dissolved oxygen in slurry needs to be regulated individually in all three. Cyanide is fed into the first two reactors and is consumed throughout the process. The second objective is to regulate the free cyanide left over in the last reactor to avoid overconsuming the expensive reactant while still maintaining desirable process conditions for the extraction of gold.

The optimal control of the CIL process is hampered by latency in concentration control of cyanide in the last reactor, and the chemical interaction between cyanide and oxygen. The addition of cyanide in the first two sequential reactors, and the subsequent measurement and control of cyanide concentration in the last reactor causes feedback delay in the control loop and a noticeable deadtime as a result. Long deadtime in a control loop causes grief for simple PID controllers and needs to be accounted for by a more advanced control solution. An additional challenge is introduced by the chemical interaction between oxygen and cyanide in the reactors to form fulminate ions (CNO^-) (Schubert, 1992):

$$2CN^- + O_2 \longleftrightarrow 2CNO^-. \tag{32}$$

The interaction between the manipulated variables causes correlation between the controlled variables, making independent regulation more difficult. Conventional approaches to optimal control of the CIL process include MPC, relying on first principles modelling of dynamics. The goal of this case study is to implement the mentioned control functionality in the CIL simulation as a singular RL control policy trained using the DDPG algorithm.

4.4.2 Adaptation of algorithm from POC3

The overall algorithm structure and a large portion of the implementation were kept unchanged from POC2 due to the successful results obtained. Naturally, the actor and critic networks were modified to suit the new control problem formulation. The more complex simulation and control problem requires a larger state observation vector which was translated to an equivalent number of nodes in the input layers of the two networks. The action space in the CIL simulation contains three oxygen inputs and two cyanide inputs. Five nodes, representing respective feed increments, were defined in the output layer of the actor network and added as inputs to the critic network. The previously developed piecewise-defined reward function was used again for the separate controlled variables. The four sub-reward functions were weighted equally in the collective reward function used to give feedback to the agent from state transitions.

The importance of careful state observation and action increment normalisation in MIMO RL control applications was realised in POC3 and applied accordingly in the CIL control implementation. The significant escalation in RL problem complexity from POC3 meant that a greater effort was required in hyperparameter tuning and algorithm optimisation. The dimensions of both hidden layers in actor and critic

networks were increased to 600 nodes, almost doubling the number of hidden nodes from the original DDPG implementation with 400 and 300 nodes respectively in the first and second hidden layers (Lillicrap et al., 2016). The reasoning behind growing the complexity of the function approximators was to increase deep learning performance and manage the larger input and output layers. The size of the replay buffer used during training was increased by a factor of 50 to 500 000 transitions. Using an episode length of 500 time steps, the buffer used in the POC experiments would only able to store 20 episodes worth of transitions. It was hypothesised that this would be an insufficient buffer to sample from for robust and efficient convergence in the higher dimensional state and action spaces. In performance tests, the hypothesis was supported, showing better convergence properties with the larger network architecture and replay buffer size. Due to the slower convergence in the CIL application, the Gaussian noise incorporated in the behaviour policy to address the explore-exploit trade-off was modified slightly. The decay rate of the exploration noise was decreased to facilitate more thorough exploration of the state and action spaces, matched with the increased dimensionality of the problem.

During initial tests with an unchanged algorithm, the dynamic regulation of dissolved oxygen in reactors was learned effectively by the RL agent. However, the deadtime in the cyanide control loop caused difficulties in learning cyanide regulation as part of the control policy. To enhance predictive capabilities of the algorithm during policy iteration, data from previous time steps was added to the state observation vector and hence as inputs to the actor and critic networks. It was hypothesised that this would give the RL algorithm more insight into the dynamics and rate of change of controlled variables in the third reactor and hence compensate for the system dead time. Similar logic involves increasing the derivative term in a PID controller to improve performance and decrease response oscillations in control loops with long deadtime (Eder, 2011). The idea of using data from multiple consecutive states as ANN inputs in RL has been explored in the past, for instance in the context of Atari games with the DQN algorithm (Mnih et al., 2015).

4.4.3 Agent training

The dynamic set point training scheme was utilised to learn a policy for optimal regulatory control of the CIL process. The training results showed that dynamic cyanide regulation improved noticeably after the introduction of previous measurements to the actor and critic input layers. Using a changing sinusoidal set point function during training, the cyanide concentration response converged to a stable sine wave which closely mirrored the set point function in terms of magnitude and wavelength, only with a slight phase lag. This phase lag corresponds to the characteristic latency in system dynamics. When a regulatory control system registers a change in set point, response time can only be as rapid as the system dynamics allow, meaning that processes with dead time inevitably exhibits some time delay in response.

The DDPG algorithm was able to learn the dynamic regulation of cyanide in the last reactor and dissolved oxygen individually in all three reactors using a single control policy in the developed RL training framework. The successful regulation

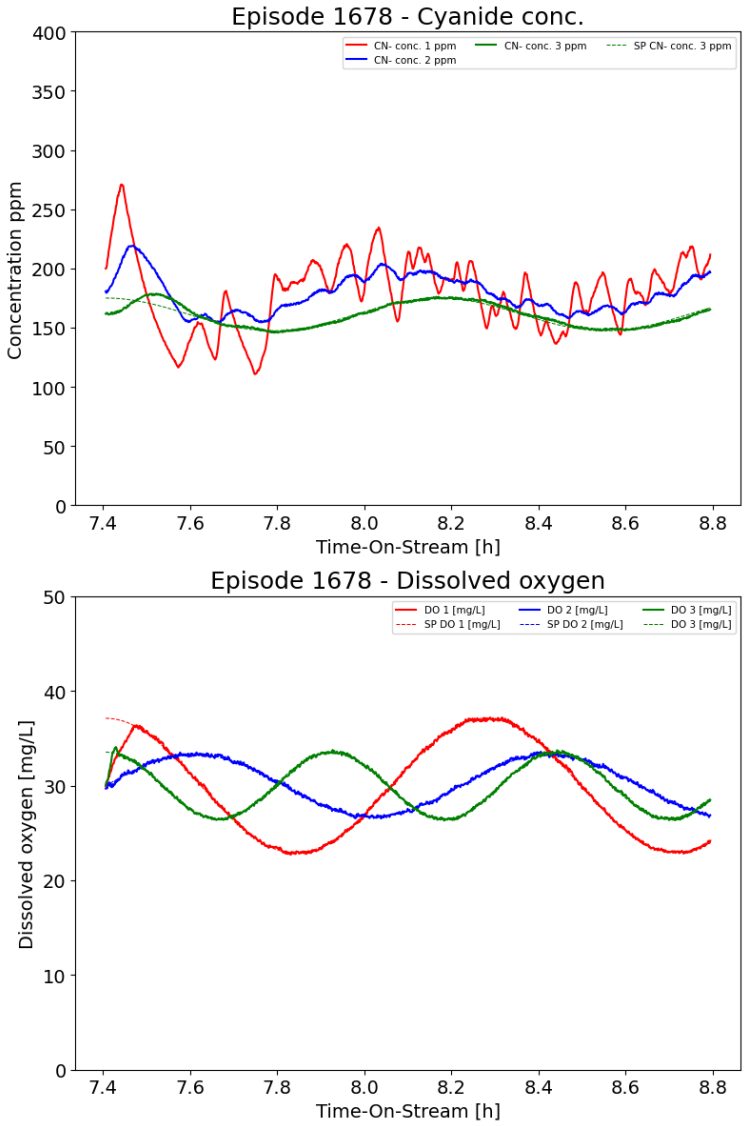

Figure 30: Control performance of fully converged DDPG agent for CIL case study during training.

of controlled variables according to sinusoidal set points after convergence can be seen in Figure 30. As expected, the higher complexity of the RL problem resulted in a higher iteration count required for convergence. Convergence is visualised in Figure 31. Similarly as in POC3, the algorithm learned the regulation of individual variables separately one after the other, eventually converging fully on a policy capable of controlling all desired process variables.

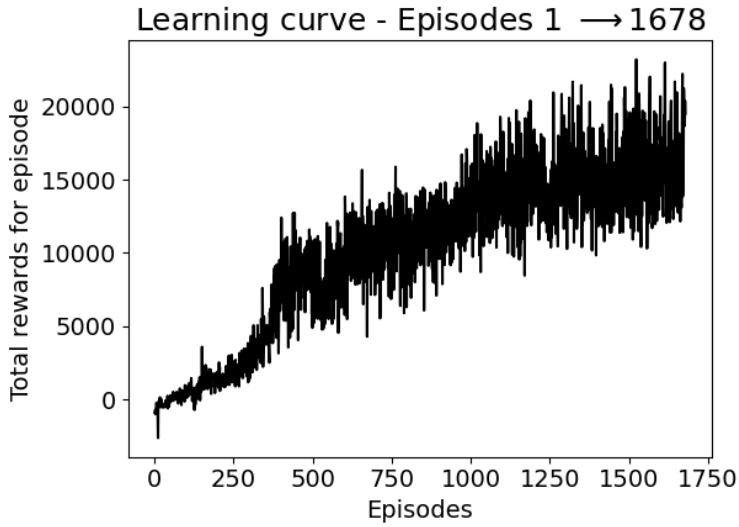

Figure 31: Learning curve of DDPG agent for CIL case study.

4.4.4 Challenges in controller deployment

Validation of the trained control policy requires deployment into the HSC simulation environment as done with previous POC experiments. This involves mapping all relevant input and output variables to the input and output layer nodes in the simulation file spreadsheet. The CIL case study algorithm requires the integration of historical process data in the policy network. Utilising the ordered data from previous time steps in ANN calculations is a feature that is yet to be released in HSC. This makes the integration of the trained policy network into the simulation via ONNX unfeasible with the current HSC build, limiting the local deployment of the trained agent. This feature is under development and may be available in upcoming releases of the HSC Sim module.

5 Discussion and conclusions

The optimal control of chemical processes has been a crucial aspect of process engineering for decades, motivated by potential improvements to operational KPIs. Established industry standard APC methods such as MPC rely on first principles modelling and feed forward control to optimise process control trajectories. Drawbacks of the model-based APC approach include computationally expensive online optimisation, complicated process modelling, required periodic recalibration, and mediocre handling of stochastic disturbances. Rapid development in process simulation technology, industrial digitalisation and machine learning has created opportunities in a wide range of data-based industrial applications over recent years. This includes optimal process control that leverages RL.

The aim of this thesis was to develop an RL-based APC solution to optimally control simulations of industrial chemical processes. Three research objectives were derived from the aim. The first research objective was to evaluate and select an appropriate RL algorithm for the task based on compatibility to problem formulation and effectiveness. The second objective was to apply the chosen algorithm and successfully compromise APC performance in a stochastic and continuous proof-of-concept control problem with acceptable data-efficiency and computational complexity. The final research objective was to effectively utilise a developed RL control policy in a simulated industrial process with comparable control objectives to current legacy APC solutions.

To fulfil the aim and research objectives defined for this thesis, a systematic problem solving and development methodology was devised. First, a comprehensive literature review was conducted to establish a knowledge base concerning fundamental theoretical concepts in RL and state-of-the-art in RL-based process control. This knowledge base could subsequently be leveraged in solution design and implementation. After completion of the literature review, an RL training scheme was established as an architecture consisting of the RL algorithm runtime environment, the process simulation file, and a proprietary API facilitating communication between the two. In the next phase, the process control problem was expressed in terms of a MDP and formulated as a sequential decision-making problem able to be solved by an RL agent. After this, the algorithm development phase began, consisting of four stages responsible for fulfilling the research objectives. Each stage introduced a new algorithm or control problem, gradually increasing in implementation complexity, forming an iterative development strategy.

POC1 involved the SISO control of a simple process simulation to test the performance and potential of the policy gradient algorithm REINFORCE. POC2 was an experiment designed to compare the actor-critic algorithm DDPG against REINFORCE using the same control problem. After selecting DDPG as the better suited algorithm, MIMO control was tested in a slightly more complex simulation in POC3 to validate algorithm performance. Finally, in the case study, the optimal control of a CIL gold recovery process simulation was undertaken as the concluding development stage.

Results from POC1 and POC2 experiments demonstrated the potential of RL agents to learn control policies for simple regulatory control problems in processes

with stochastic disturbances. Figure 16 visualises the ability of the REINFORCE algorithm to handle stochastic process disturbances while maintaining robust set point regulation performance. Figures 20 and 24 present the control responses of trained and deployed REINFORCE and DDPG control agents respectively in the same process control scenario. The results indicate robust and stable control performance after a relatively low number of training episodes.

After comparing and evaluating the two algorithms tested in POC1 and POC2, it was observed that DDPG outperformed REINFORCE in terms of data efficiency, converging significantly faster. The lower number of episodes required by DDPG for convergence can be visualised when comparing learning curves in Figures 19 and 23. The data efficiency advantage with DDPG can be explained partially by its use of a replay buffer and conversely the slower convergence rate of REINFORCE can partially be explained by it being a Monte Carlo method, only updating the policy after complete episodes instead of after every time step. In addition to a difference in algorithm efficiency, the inherently better simultaneous handling of continuous and multi-dimensional action spaces made DDPG the better choice for optimal process control. Hence, the first research objective was completed through POC1 and POC2. A significant finding during POC2 development was that a piecewise defined reward function can be designed to work better than a standard quadratic or hyperbolic sine function with process values close to the set point.

POC3 results demonstrated that the developed DDPG agent is able to learn precise and reliable control policies efficiently for MIMO problems. The responses for two controlled variables regulated by the same policy can be seen in Figures 28 and 29. The solution and results for POC3 acted as a precursor to more complex optimal control problems and achieved the second research objective. Key findings during the development of POC3 included the importance of consistent normalisation and scaling of ANN outputs in RL algorithms applied to problems with multiple controlled variables. An interesting observation from the results was the tendency of the algorithm to sequentially learn the regulation of multiple variables, as seen from the two steep Section in the learning curve of Figure 27, representing the convergence of the two controlled variables.

The training results of the final case study demonstrated the ability of the developed algorithm to handle optimal control of an industrial process simulation by learning the simultaneous regulation of multiple interacting process variables. The case study was able to meet the final research objective during online policy iteration with robust process control performance as seen in Figure 30. The obtained results show that RL can be used as a tool to optimally control industrial process simulations based on an aptly defined reward function governing the control objective. This is accomplished without model-based real time optimisation or accurate information on process dynamics and stochastic disturbances. Instead, the iteratively updated control policy can learn through experience to handle stochastic system disturbances, which are often a challenge for legacy APC solutions. On a more practical level, the algorithm development process has produced significant insights regarding hyperparameter tuning, reward function engineering and the general adaptation of RL policies to process control problems. These insights have been invaluable in helping understand

the fundamental complications and solutions in applying RL to optimal process control in the virtual absence of available supporting documentation.

The aforementioned findings and conclusions help motivate and also facilitate future research and development of more sophisticated RL optimal control implementations. Possible avenues of future research include studying algorithm implementation strategies to improve performance. Possibilities include alternative actor-critic algorithms such as TRPO and PPO, experimenting with ANN architectures, hyperparameter optimisation, and considering additional algorithmic features like hindsight experience replay. A limitation of the scope of this thesis but an interesting investigation would be the quantitative benchmarking of an RL control policy against a legacy APC solution using the same control objective. Furthermore, to showcase and test the customisability of RL problem formulation, it would be interesting to apply RL to higher level process optimisation. This could involve the inclusion of economic KPIs in the reward function to optimise control strategies against more abstract criteria.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference on Machine learning - ICML '04*, Banff, Alberta. ACM Press.

Chai, T., Qin, S. J., and Wang, H. (2014). Optimal operational control for complex industrial processes. *Annual Reviews in Control*, 38(1):81–92.

Demidov, D. and Al-Refai, R. (2021). *Deterministic Policy Gradient Algorithms Overview*. Mohamed bin Zayed University of Artificial Intelligence, Masdar, United Arab Emirates.

Deng, J., Sierla, S., Sun, J., and Vyatkin, V. (2022). Reinforcement learning for industrial process control: A case study in flatness control in steel industry. *Computers in Industry*, 143:103748.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. In *33rd International Conference on Machine Learning (ICML)*, New York, New York. JMLR: W&CP.

Eder, H. H. (2011). *Dealing with long deadtime – a comparison between PID, PI, Smith Predictor and Model Based Control (MBC)*. ACT.

Faria, R. D. R., Capron, B. D. O., Secchi, A. R., and De Souza, M. B. (2022). Where Reinforcement Learning Meets Process Control: Review and Guidelines. *Processes*, 10(11):2311.

Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). *An Introduction to Deep Reinforcement Learning*. Now Foundations and Trends.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In *35th International Conference on Machine Learning*, Stockholm, Sweden. Proceedings of Machine Learning Research.

Görges, D. (2017). Relations between Model Predictive Control and Reinforcement Learning. In *20th IFAC World Congress*, volume 50, pages 4920–4928. IFAC-PapersOnLine.

Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019). Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7:133653–133667.

Jiang, Y., Yin, S., Li, K., Luo, H., and Kaynak, O. (2021). Industrial applications of digital twins. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2207):20200360.

Kim, J. W., Park, B. J., Yoo, H., Oh, T. H., Lee, J. H., and Lee, J. M. (2020). A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system. *Journal of Process Control*, 87:166–178.

Langley, P. (2011). The changing science of machine learning. *Machine Learning*, 82(3):275–279.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

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.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *33rd International Conference on Machine Learning*, pages 1928–1937, New York, New York. Proceedings of Machine Learning Research.

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., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mowbray, M., Savage, T., Wu, C., Song, Z., Cho, B. A., Del Rio-Chanona, E. A., and Zhang, D. (2021a). Machine learning for biochemical engineering: A review. *Biochemical Engineering Journal*, 172:108054.

Mowbray, M., Smith, R., Del Rio-Chanona, E. A., and Zhang, D. (2021b). Using process data to generate an optimal control policy via apprenticeship and reinforcement learning. *AIChE Journal*, 67(9).

Mowbray, M., Vallerio, M., Perez-Galvan, C., Zhang, D., Del Rio Chanona, A., and Navarro-Brull, F. J. (2022). Industrial data science – a review of machine learning applications for chemical and process industries. *Reaction Chemistry & Engineering*, 7(7):1471–1509.

Nicolae, N., Popescu, M., and Patrascioiu, C. (2019). Implementation of Advanced Process Control in Refineries. In *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, pages 95–100, Sinaia, Romania. IEEE.

Openja, M., Nikanjam, A., Yahmed, A. H., Khomh, F., and Jiang, Z. M. J. (2022). An Empirical Study of Challenges in Converting Deep Learning Models. In *2022*

IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 13–23, Limassol, Cyprus. IEEE.

Pan, E., Petsagkourakis, P., Mowbray, M., Zhang, D., and Rio-Chanona, E. A. d. (2021). Constrained model-free reinforcement learning for process optimization. *Computers & Chemical Engineering*, 154:107462.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Zeming, L., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Soumith, C. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8024–8035, Red Hook, New York. Curran Associates Inc.

Peroni, C., Kaisare, N., and Lee, J. (2005). Optimal control of a fed-batch bioreactor using simulation-based approximate dynamic programming. *IEEE Transactions on Control Systems Technology*, 13(5):786–790.

Petsagkourakis, P., Sandoval, I., Bradford, E., Zhang, D., and del Rio-Chanona, E. (2020). Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering*, 133:106649.

Petsagkourakis, P., Sandoval, I. O., Bradford, E., Galvanin, F., Zhang, D., and Rio-Chanona, E. A. d. (2022). Chance constrained policy optimization for process control and optimization. *Journal of Process Control*, 111:35–45.

Plathottam, S. J., Richey, B., Curry, G., Cresko, J., and Iloeje, C. O. (2021). Solvent extraction process design using deep reinforcement learning. *Journal of Advanced Manufacturing and Processing*, 3(2).

Rawlings, J. B., Mayne, D. Q., and Diehl, M. M. (2019). *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, Santa Barbara, California, 2 edition.

Roine, A. (2023). HSC Chemistry [Software]. Metso, Pori, Finland. Software available at www.metso.com/hsc.

Russell, S. J., Norvig, P., and Davis, E. (2010). *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, 3rd ed edition.

Sachio, S., Mowbray, M., Papathanasiou, M. M., del Rio-Chanona, E. A., and Petsagkourakis, P. (2022). Integrating process design and control using reinforcement learning. *Chemical Engineering Research and Design*, 183:160–169.

Santander, O. (2015). *Economic Model Predictive Control of Chemical Processes*. PhD thesis, University of Waterloo, Waterloo, Ontario.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Schubert, J. H. (1992). Dynamic simulation of the leaching and adsorption sections of a gold plant. Master's thesis, University of Cape Town, Cape Town, South Africa.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs].

Shah, H. and Gopal, M. (2016). Model-Free Predictive Control of Nonlinear Processes Based on Reinforcement Learning. In *4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS*, volume 49, pages 89–94, Tiruchirappalli. IFAC-PapersOnLine.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. In *31st International Conference on Machine Learning*, volume 32, pages I–387–I–395, Beijing, China. JMLR.org.

Sola, J. and Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468.

Sutton, R. S. and Barto, A. (2020). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts; London, England, 2 edition.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *30th AAAI Conference on Artificial Intelligence*, Phoenix, Arizona. AAAI Press.

Van Rossum, G. and Drake, F. L. (2009). *Python reference manual*. CreateSpace, Scotts Valley, California.

Watkins, C. (1989). *Learning From Delayed Rewards*. PhD thesis, King's College, London, England.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Woolf, P. (2009). *Chemical Process Dynamics and Controls Book 1*. Open Michigan.

Yoo, H., Kim, B., Kim, J. W., and Lee, J. H. (2021). Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Computers & Chemical Engineering*, 144:107133.

Zhou, K., Taigang Liu, and Lifeng Zhou (2015). Industry 4.0: Towards future industrial opportunities and challenges. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2147–2152, Zhangjiajie, China. IEEE.