Reinforcement Learning based Controller Design for Nonlinear Process Control

by

Hareem Shafi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Process Control

Department of Chemical and Materials Engineering
University of Alberta

Abstract

Reinforcement learning (RL) has received wide attention in various fields lately. Model-free RL brings data-driven solutions that learn the control strategy directly from interaction with process data without the need for a process model. This is especially beneficial in the case of nonlinear processes where the process model might not be readily available or accurate. It circumvents the need for a model identification step. It is also able to re-train in the case of process shifts or process noise to improve performance. In contrast, traditional model-based control methods require an explicit process model, and the performance of parametric models deteriorates over time in case of process shifts or unmeasured disturbances. However, despite learning schemes like deep deterministic policy gradient (DDPG), deep Q-networks (DQN) or actor-critic, convergence to an optimal policy in process control remains a persistent challenge.

This thesis focuses on the integration of RL based methods into the process control domain. The first part of the thesis addresses the multivariate control of continuous state and action spaces of chemical processes. A parallel learning architecture is utilized to improve the control quality and convergence to an optimal policy through a better exploration of the state and action space. A centralized RL agent is able to successfully learn an effective policy for servotracking control of a quadruple tank system as an example. It can learn directly from interactions with the process while ensuring that the process remains operational.

The second part of the thesis deals with developing a hierarchical RL based constrained controller for a higher-level optimization of the Primary Separation Vessel (PSV). A supervisory RL agent is concerned with improving the bitumen recovery rate through interface level setpoint manipulation. A lower-level RL ensures control of the froth-middlings interface level. It does so despite the nonlinear nature of the process and the unpredictability of the ore composition of the slurry fed into the PSV. The unpredictability also necessitates regulation of the tailings density below a sanding threshold. It is carried out through manipulations of the tailings flowrate by a non-interacting sanding prevention RL agent. In the interface level control loop, behavioral cloning based two-phase learning scheme to promote stable state space exploration is also proposed. Based on simulation results, the behavioral cloning scheme ensured improved convergence to the near-optimal policy. The proposed hierarchical structure successfully demonstrates improved bitumen recovery rate by manipulating the interface level while preventing sanding, demonstrating the feasibility of such approaches to chemical processes.

Preface

The major work of this thesis is contained in two chapters. The first part of the work was done on developing a reinforcement learning based control strategy for multivariate control of a nonlinear process, and it is contained in Chapter 3. The contributions of this part have been submitted in:

Referenced Conference Publication:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Asynchronous Reinforcement Learning Based Continuous Space Control for Multivariable Process*, International Federation of Automatic Control, 2020 [submitted].

The contributions of this part have also been presented in:

Conference Presentation:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Reinforcement Learning Based Controller for Setpoint Tracking with Illustrative Application in a Quadruple Tank System*, 69th Canadian Chemical Engineering Conference, Nova Scotia, 2019.

Chapter 4 contains the second part of the work. It proposes a hierarchical architecture based reinforcement learning control scheme for economic optimization of an oil sands gravity separation vessel. Its contributions have been submitted to:

Referenced Journal Publication:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Hierarchical Constrained Reinforcement Learning for Optimization of Bitumen Recovery Rate in a Primary Separation Vessel*, IEEE Transactions on Control Systems Technology [submitted].

The idea generation and its materialization into code was completed in collaboration with Dr. Kirubakaran Velswamy. I was responsible for the mathematical derivation, execution of the experiments, collection and analysis of the data to compile them into representative results, and composing the manuscripts for the referred publications. Dr. Biao Huang was the supervisor involved in supervision, discussion and critique.

Dedicated to Mama and Baba

Acknowledgements

Thank you, Dr. Biao Huang, for allowing me the privilege of exploring this novel research area under your guidance and support. Your vision continues to inspire me as it has many before me. I am grateful for all that I have learned from you, and I am grateful for the patience I have been extended throughout this learning process.

Thank you, Dr. Kirubakaran Velswamy, for being the mentor very few are lucky enough to have, in research and in life. I could not have done this without you. Thank you for the unconditional support at every step, and thank you for expanding my understanding of the domain. I carry forward all that I have learned from you in these two years.

I would also like to thank my research group for fostering an encouraging learning environment and lending ears for my hour long presentations every once in a while. It has been a pleasure to be a part of this dynamic group. A special thanks to Dr. Jinfeng Liu and Dr. Fadi Ibrahim for their help with the PSV model.

Thank you to my family for always believing in me and ensuring I had the opportunities and experiences that brought me where I am today. Baba, thank you for inspiring me to be a better version of myself every day. Bhai, thank you for teaching me that everything is possible if I put my mind to it. Mama, I hope I have made you proud.

Contents

List of Tables

List of Figures

List of Symbols

$s_t \in S$: Set of observable states

$a_t \in A$: Set of actions

μ_t, σ_t: Mean and standard deviation of the action

$\pi(a_t \mid s_t)$: Policy

r_t: Instantaneous reward

γ: Discount factor

R_t: Returns

$\hat{R}_t$: Returns prediction

$V(s)$: Value function

$Q(s, a)$: Action-value function

E_π: Expectation from following policy π

θ_A: Actor network parameters

$\pi_{\theta_A}(a_t \mid s_t)$: Policy with parameters θ_A

θ_C: Critic network parameters

$Q_{\theta_C}(s_t, a_t)$: Action-value function with parameters θ_C

$V_{\theta_C}(s_t)$: Value function with parameters θ_C

$J(\theta_A)$: Actor loss function

$d\theta_A$: Actor gradient

$J(\theta_C)$: Critic loss function

$d\theta_C$: Critic gradient

$H(\pi)$: Shannon's entropy

β: Bias correction

$A_{\theta_C}(s_t, a_t)$: Advantage function with parameters θ_C

Chapter 1

Introduction

1.1 Industrial background

The advances in technology in recent years ushered in what is known as the fourth industrial revolution. The ability of countries to infuse technology-driven changes to improve the productivity and efficiency of all industrial processes has the potential to transform their economies. Industry 4.0 is a subset of this technological revolution, and it entails the switch towards automation of industrial operation. It aims to work towards smart factories with the capability of self-monitoring their processes and making intelligent decisions. Such a factory relies on the deployment of sensor networks, their efficient operation, and relaying the information obtained to a decentralized computing unit that can use this information to take meaningful decisions towards maximizing its economic objectives.

Data analytics and artificial intelligence are important pillars of Industry 4.0. Cognitive computing is employed in the processing of the signals acquired from the sensor networks. The industrial internet of things (IIOT) enables the transfer of data and cloud computing allows storage of relevant data. Employing machine learning then enables reasoning for efficient decision making using this data. Strides are being made in each of these domains and their fusion is fundamentally changing the way the industries work.

The manufacturing industry is a front-runner in the businesses that have leveraged data science, communication, and robotics to achieve Industry 4.0 goals. With access to more data, Industry 4.0 converts regular machines to self-aware and self-learning machines to improve their performance. Real-time status monitoring and product tracking, as well as the control of production processes, are the pillars of Industry 4.0.

1.1.1 Process control industry

The process control industry is a continuous production manufacturing industry. To ensure the overall efficacy of the industrial processes, not only the controlled variable is to be maintained at defined setpoints, but intermediate variables also have to be regulated. These have to be done while accounting for sources of uncertainty regarding the input variables as well as disturbances introduced into the process.

As is true for any industry, this low-level control and regulation is done to achieve an overall economic objective. The economic objective is achieved through the optimization of qualitative and quantitative requirements. As such, the problem statement in process control falls in one of three cases:

1. Servotracking

   Tracking setpoint changes of the process variable in the presence of disturbances.

2. Regulation

   Maintaining the state of the system when disturbances are introduced to the system.

3. Optimization

   Minimizing a loss function of the dynamical system or maximizing economic returns.

The study of these cases falls under the purview of this thesis, and it focuses on finding novel solutions for these cases in the process control and oil sands industry.

1.1.2 Oil sands industry

Canada has the third-largest proven oil reserves in the world. These oil reserves exist primarily in the form of oil sands; a loose formation of sand grains or solid sandstone with clay, interspersed with bitumen, a heavy and viscous form of crude oil. The bitumen can be extracted and processed to produce synthetic crude oil [2]. The Canadian Oil Sands industry has a capacity of producing 166.3 billion barrels of crude oil products [3]. Sales from oil sands producers alone added up to CAD \$40bn in 2016 [4]. The revenue and employment opportunities generated contributes significantly to the national economy.

The oil sands industry is divided into the upstream, midstream, and downstream segments. Extraction and production of crude oil and natural gas falls under the purview of the upstream segment. The midstream industry deals with the processing, storing, marketing, and transport of the upstream products. Finally, the downstream industry includes the final processing industries such as oil refineries and petrochemical plants, and their retail and distribution.

In the upstream industry, the production process starts with surface mining or in-situ production. One-fifth of the total production is based on open-pit ore extraction. It starts with the mining phase where oil sands are shoveled out of the ground. The mined ore is then crushed and transported for the extraction phase. For extraction, heat and chemicals are added to the crushed ore to form a slurry mixture. This mixture is then sent to a gravity separation vessel from which the bitumen rich froth is sent to froth treatment for extraction, while the solids settle down to form the tailings layer which is then sent for processing before disposal to tailings ponds.

Four-fifths of the bitumen reserves lie too deep to be extracted through open-pit mining. In-situ bitumen extraction is used in that case. It relies on bitumen extraction through steam injections into well pads dug into the ground. The bitumen reservoir is heated through the steam pumped into the injection wells. Bitumen's viscosity reduces and it liquefies forming a water-bitumen emulsion which is pumped to the surface where separation is carried out. The dilbit obtained from the separation is sent for further processing to obtain bitumen.

Starting from the ore quality, several uncertainties are involved in the bitumen extraction processes. Control of the quality of froth or dilbit from the gravity separation processes can decrease the load on the downstream processes, resulting in tremendous economic benefit to the oil sands producers and service companies.

1.2 Motivation and objectives

As motivated previously, the general objectives of process control in the industry are regulation, servotracking, and optimization. These objectives are met through controllers designed specifically for these goals.

1.2.1 Current practices

Conventionally, the control of nonlinear processes in the industry is model-based or optimization-based. Model-based controllers require the first principles or empirical model of the system to derive the parameters of the controller while the optimization-based controllers attempt to minimize a certain loss function of the dynamical system to derive a control law.

1.2.2 Limitations and challenges

Development of the first principles models requires specific domain knowledge and is time-consuming for complex nonlinear systems. Moreover, the information required for the model derivation of the system might not be readily available or accurate. Empirical models come with their own set of assumptions. Meanwhile, optimization-based controllers have the drawback of increasing complexity with scale. Considerations such as the prediction horizon and control horizon add more complexity to optimization-based controllers. Additionally, shifts in operating conditions reduce the effectiveness of these controllers, so periodic tuning is required.

1.2.3 Scope for RL

It would be beneficial given the aforementioned limitations and challenges with conventional controllers to propose a self-learning model-free controller. Such a controller should scale well with multiple inputs and outputs and adapt to changing operating conditions. Since the use of neural networks as function approximators has enabled continuous state and action space control using RL, controllers based on RL emerge as suitable candidates for the control of nonlinear multiple-input multiple-output (MIMO) processes. They learn directly from interaction with the process data and require minimal human intervention. They are also able to adapt to shifts in operating conditions and take instant actions. Since the RL agents learn by minimizing a loss function, RL based control strategies are also ideal as an alternative for optimization-based controllers.

1.3 Thesis contribution & outline

There are two major contributions presented in this thesis:

1. **RL for Multivariate Control & Study on Degree of Asynchronous Learning**
   We developed an asynchronous RL based control strategy for effective multivariate control of a nonlinear continuous chemical process that learns from direct interaction with the process data. We also investigated the effect of the degree of parallel learners on the convergence and optimality of the RL based control.

2. **Economic Objective Optimization using Hierarchical Constrained RL**
   We developed a hierarchical RL scheme to optimize the bitumen recovery rate of a gravity separation vessel through manipulation of its interface level while regulating the process vari-

ables within their operational range. The RL scheme is able to utilize a two-phase learning strategy to leverage the existing conventional control and later improve on it.

The rest of the thesis is organized as follows: Chapter 2 provides the theoretical background of machine learning and in specific reinforcement learning. Chapter 3 discusses the work done on RL for multivariate control and presents the results of an investigation on the effects of parallel learning on RL control. Chapter 4 presents the hierarchical RL strategy for the economic optimization of a gravity separation vessel. Chapter 5 presents the conclusions and possible future opportunities to take forward from this thesis work.

Chapter 2

Machine Learning Fundamentals

The main contribution of this thesis is leveraging the advances in machine learning for improving the performance of controllers in continuous chemical processes. This chapter, therefore, provides an overview of the machine learning theory that would enable readers to understand the work done in the thesis. For in-depth understanding of the sections in this chapter, [5] and [6] are recommended,

Artificial intelligence (AI) is used as an umbrella term to refer to the scientific study and development of machines that can perform tasks that are considered characteristic of human beings e.g. planning, reasoning, learning, and speech recognition. Some of the oft-mentioned major goals that the artificial intelligence community is focused on achieving are related to:

- Artificial General Intelligence

  The subject of many speculations and garnering much interest from the science fiction genre, artificial general intelligence (AGI) is the domain that is concerned with the development of machines that are capable of performing all cognitive tasks that human beings can perform. The AI community is divided on the matter, but predictions have been made about when and if that would be achieved [7], and a number of tests have been prescribed for assessing AGI including the Turing test. The other domains, however, have more specific objectives.

- Natural Language Processing

  Natural language processing (NLP) is a subfield of artificial intelligence and linguistics and is concerned with the interactions between humans and machines. It focuses on processing natural language data to achieve objectives such as speech recognition and interpretation and generation of natural language [8]. Advances in NLP are all around us in the form of customer services like chat boxes and phone lines. Apple Siri, Amazon Alexa, and Google Assistant are also at the forefront of the commercial NLP AI currently in use.

- Robotics

  The field of robotics has also been of much interest as a commercial application of AI. AI robots focus on learning through sensory inputs to perform a series of mechanical manipulations to achieve defined goals. The sensory inputs could also be in the form of images or videos in which case it becomes an amalgamation of computer vision with robotics. AI in robotics has focused on path-planning and space reasoning among other things [9]. Applications of AI to industrial robots, humanoid robots as well as surgical robots have made their way into the mainstream.

- Computer Vision

  Computer vision is concerned with enabling machines to analyze and interpret images. With vision being one of the most abundant sensory resources, it is being used concurrently with other branches of AI to contribute to applications such as autonomous vehicles, facial recognition, and manufacturing robots [10].

- Machine Learning

  Machine learning (ML) is the body of scientific work aimed at imparting to machines the ability to learn without being explicitly programmed. Learning, in this case, is specifically defined as improving performance on given tasks, judged by a defined criterion. This thesis employs ML to improve controller performance on continuous chemical processes.

This chapter focuses on the ML theory and algorithms. With the common goal to improve performance on a specific task with minimal human intervention and without explicit programming, ML algorithms are divided into 3 main categories depending on the type of information available to the machine.

- Supervised Learning

  Used to develop a predictive model when the data available is labelled.

- Unsupervised Learning

  Used to group and interpret data when the data available is unlabelled.

- Reinforcement Learning

Used to learn a policy to achieve a defined goal when reinforcement for actions taken is available.

These categories are described in subsections 2.1 to 2.3.

2.1 Supervised learning

Supervised learning (SL) is employed when labelled data is available, i.e. the data available has both the input and the corresponding output available, and a predictive model has to be developed that optimally maps the input data to the output data. Mathematically, the relationship between the labelled data can be interpreted in terms of a function f, where X represents the input data, Y represents the corresponding output, and ϵ is the uncertainty as shown in equation 2.1.

$$Y = f(X) + \epsilon \tag{2.1}$$

SL is concerned with algorithms to create a model $\overline{f}$ to estimate the function f. Derivation of the predictive model is carried out in two steps: training, and testing. The data available is divided into two subsets (training-test split) to find $\overline{f}$: the (larger) training set $[X_{training}, Y_{training}]$ and the (smaller) test set $[X_{test}, Y_{test}]$. First, the training data is used to create a model. The test data is then used to assess the performance of the model. To ensure an unbiased estimate of the model performance, before the training-test split, the available data is shuffled and the test data is kept aside, i.e. the model does not see it during training.

When the output is continuous, the SL problem can be expressed as a regression problem. On the other hand, if the output is discrete, it would be a classification problem. For each case, the algorithm would attempt to map features in the input data X to the output Y. This mapping is performed under supervision. The output data available provides supervision to the model in the form of a cost function. The cost function contains the deviation of the model's predicted output $\hat{Y}$, obtained as presented in equation 2.2, from the actual output Y (equation 2.3).

$$\hat{Y} = \overline{f}(X) \tag{2.2}$$

$$L(X, \overline{f}(X), Y) = \hat{Y} - Y \tag{2.3}$$

The model $\overline{f}$ is learned through iterative optimization of this loss function. To conceptualize the working of SL algorithms, first a regression and a classification problem will be explained in the following subsections.

Observation	**Years of Higher Education**	**Monthly Income (CAD)**
1	4	6500
2	5	7625
3	0	4250
4	2	4600
5	4	6200
6	6	7500
7	3	5000
8	2	6000
9	1	4000
10	5	7650
11	4	6000
12	8	10000
13	7	8750
14	5	9500
15	6	8500

Table 2.1: Predicting monthly income based on years of higher education

2.1.1 Regression

Problem statement:

Given the number of years of higher education, derive a model to predict the average monthly income for individuals.

The data available is represented in table 2.1, and they are illustrated in figure 2.1. The next step is to divide the data randomly into training and testing sets. Dividing the data across a 2:1 ratio into training and data set yields sample training and test sets shown in table 2.2 and are illustrated in figure 2.2.

For a univariate input case like this problem statement where the input data can be presented as in equation 2.4, the simplest linear regression model would look like the one presented in equation 2.5.

$$X = [x_1] \tag{2.4}$$

$$\overline{f}(X) = \alpha_1 x_1 + \alpha_0 \tag{2.5}$$

The model parameters α_1 and α_0 are determined using different optimization algorithms presented in the later subsections. Univariate linear regression leads to the fit shown in figure 2.3 where the values of α_1 and α_0 are shown in table 2.3.

Now, if the input dimensions were to increase up to N, say if years of relevant job experience were also a consideration, the regression would now consider input X to be represented as given in equa-

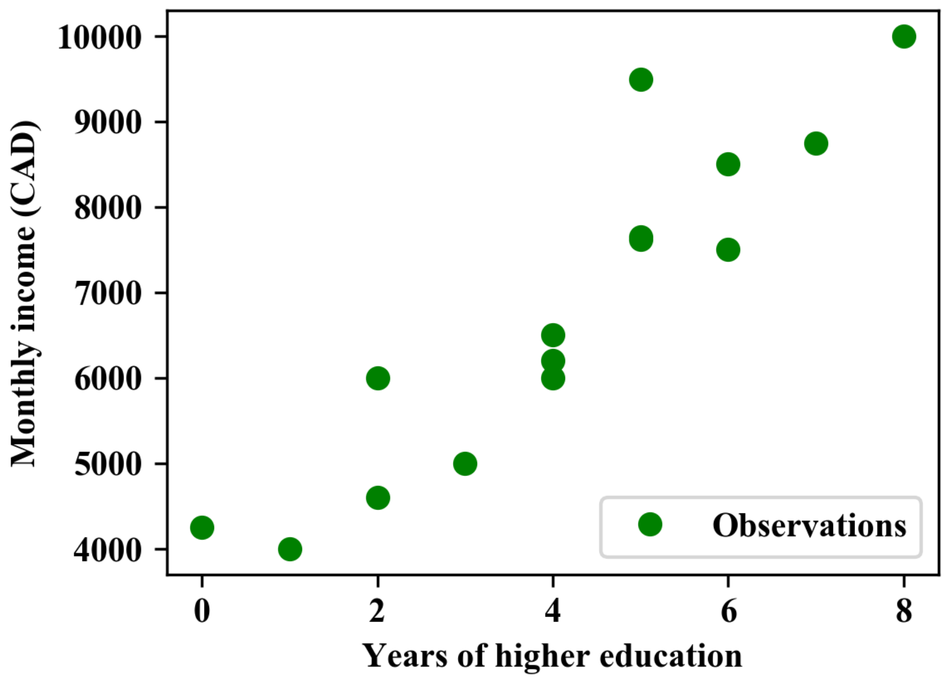


Figure 2.1: Predicting monthly income based on years of higher education

Training Data Set

Observation	Years of Higher Education	Monthly Income (CAD)
1	4	6500
2	0	4250
3	2	4600
4	4	6200
5	3	5000
6	1	4000
7	4	6000
8	7	8750
9	5	9500
10	6	8500

Test Data Set

Observation	Years of Higher Education	Monthly Income (CAD)
1	5	7625
2	6	7500
3	2	6000
4	5	7650
5	8	10000

Table 2.2: Training-test data

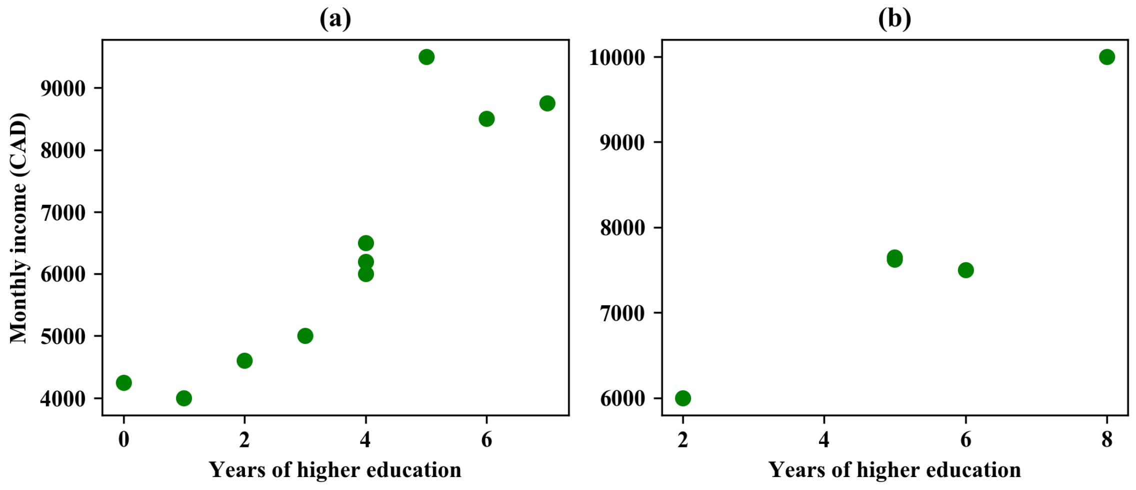


Figure 2.2: Training-test data (a) training data, (b) test data

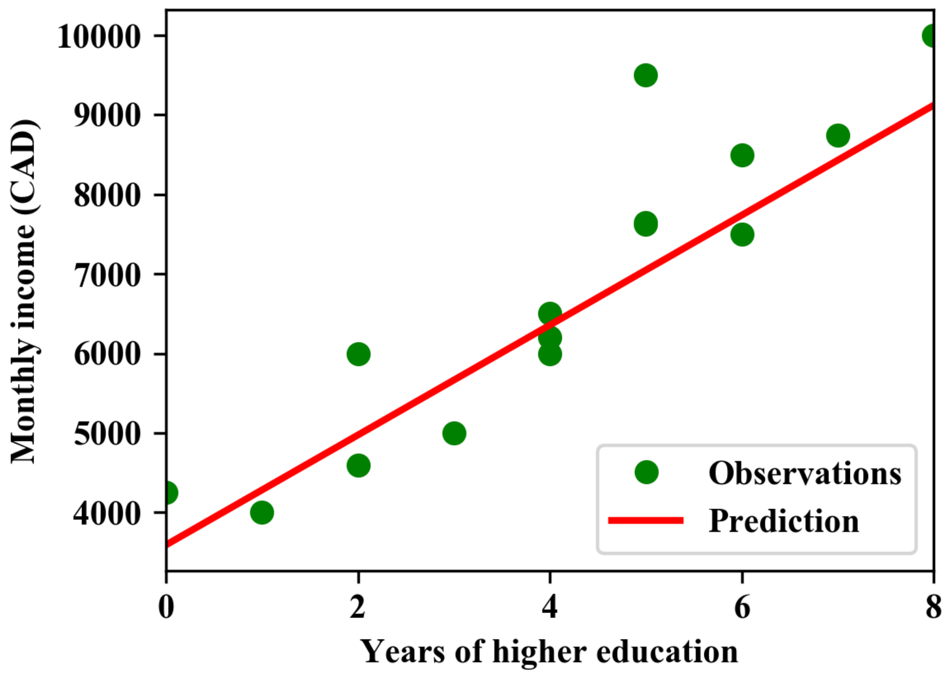


Figure 2.3: Univariate linear regression

Parameter	Value
α_1	777.462
α_0	3591.491

Table 2.3: Univariate linear regression results

Number of Petals	**Length of Petals (inches)**	**Flower**
5	6	Rose
10	3	Rose
...	...	...
7	5	Rose
3	5	Lily

Table 2.4: Flower classification data

tion 2.6. The model would not be presented as in equation 2.7. This would now be a multivariate linear regression with the optimizer solving for $[\alpha_N, ..., \alpha_0]$.

$$X = [x_N, x_{N-1}, ..., x_2, x_1] \tag{2.6}$$

$$f(X) = [\alpha_N, \alpha_{N-1}, ..., \alpha_2, \alpha_1, \alpha_0][x_N, x_{N-1}, ..., x_2, x_1, 1]^T \tag{2.7}$$

If the relationship between the input and output data cannot be described by a linear relationship, nonlinear regression is employed.

2.1.2 Classification

Problem Statement: Given the number of petals and petal lengths for various flowers, determine if each flower is a rose or a lily.

The problem statement makes it clear that a classifier is required to distinguish between roses and lilies given their input features (petal length and number). The dataset would look like the one presented in table 2.4. The same procedure for splitting the dataset into training and test data would be followed. However, in contrast to regression, the optimizers would result in a classification line as shown in figure 2.4 where red represents roses and cyan represents lilies. Parametric classification methods include using support vector machines (SVM) or applying nonlinear activation functions to the regression sum. Non-parametric methods include k-mean clustering and decision trees. These methods will be briefly discussed in later sections.

2.1.3 Cost function

The cost function is a tool that incorporates the supervision in the form of a deviation of the prediction from the actual output. The objective of a supervised algorithm is to minimize the cost function. Hence the objective can be presented as in equation 2.8.

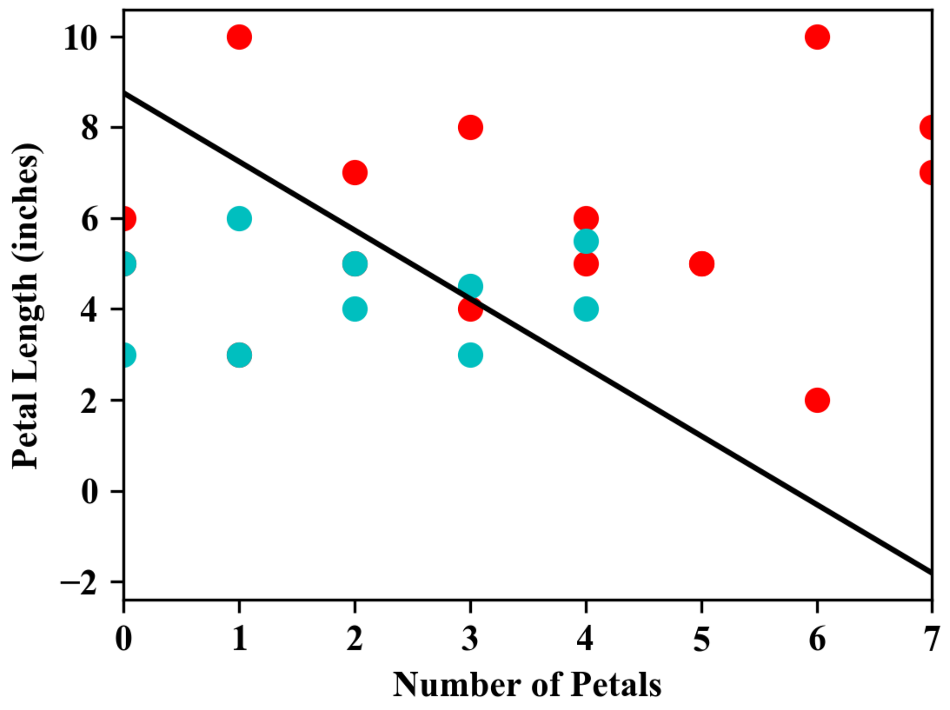


Figure 2.4: Flower classification

$$\min_{\overline{f}} L(X, \overline{f}(X), Y) \tag{2.8}$$

The cost function L for a regression task looks at the distance between the prediction made by the SL model and the actual output in the training set. The two cost functions that are commonly used are given below:

Mean Absolute Error (MAE):

It is important to record the magnitude of distance between the actual and predicted output in the cost function to make it monotonic. To ensure this, the MAE records the mean of absolute error as given in equation 2.9.

$$MAE = \frac{1}{N} \sum_{N} \mid \hat{y} - y \mid \tag{2.9}$$

where N is the total number of samples, and $\hat{y}$ is the predicted output and y is the actual output at each sample.

Mean Squared Error (MSE):

Another way of ensuring monotonicity is by taking the square of the distance between the predicted and actual output. This is done in MSE which calculates the average squared distance between the actual and predicted output as given in equation 2.10.

$$MSE = \frac{1}{N}\sum_{N}(\hat{y} - y)^2 \tag{2.10}$$

MSE is used more often in optimization compared to MAE as it is easier to calculate its derivative since it uses a squared function while MAE uses the absolute function. MSE also ensures that as the distance increases, a greater penalty is applied.

2.1.4 Optimization

Several optimization algorithms are employed to solve for model parameters that minimize the associated cost function. Two of the most commonly used optimization algorithms are defined here.

Gradient Descent:

The fundamental concept behind the gradient descent algorithm is minimizing the cost function through iterative updates of the model parameters in the direction of the steepest descent. The direction of steepest descent is found as the negative of the gradient of the cost function with respect to the parameters. The algorithm starts with initial parameter values and converges to the parameters which provide a minimum value for the cost function as shown in figure 2.5.

As an example, consider the regression problem statement in subsection 2.1.1. We will now look at how the values of the model parameters α_1 and α_0 were reached. Taking MSE as the cost function, and some initial values of the parameters α_1^0 and α_0^0, the gradient of the loss function would then be calculated for each parameter. Rewriting the cost function as a function of the parameters, we obtain equation 2.11.

$$L(\alpha_1, \alpha_0) = \frac{1}{N}\sum_{N}((\alpha_1 x + \alpha_0) - y)^2 \tag{2.11}$$

where x is the input as each sample, y is the actual output, and N is the total number of samples. The gradient of the loss function with respect of each parameter would then be calculated as given in equations 2.12 and 2.13.

$$\frac{\partial L}{\partial \alpha_1} = \frac{2}{N}\sum_{N}(-x(y - (\alpha_1 x + \alpha_0))) \tag{2.12}$$

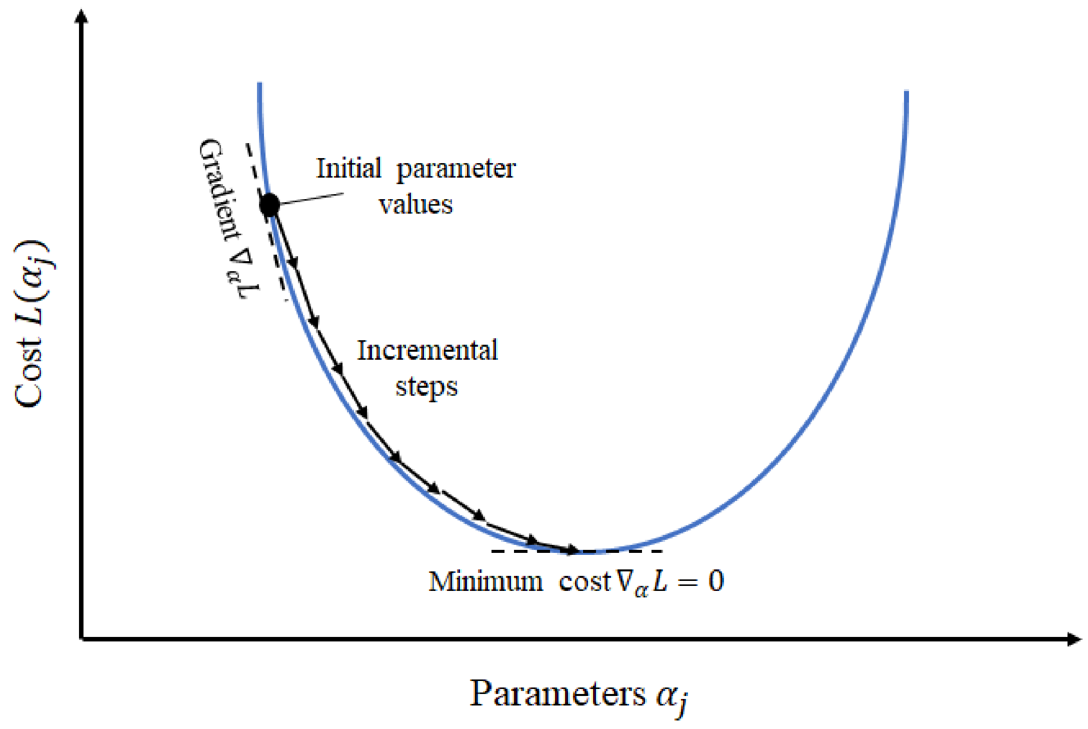


Figure 2.5: Gradient descent algorithm

$$\frac{\partial L}{\partial \alpha_0} = \frac{2}{N}\sum_{N}(-(y-(\alpha_1 x+\alpha_0))) \tag{2.13}$$

A learning rate ($\beta \mid 0 \leq \beta \leq 1$) is used to set the size of the update to the j^{th} parameter as shown in equation 2.14.

$$\alpha_j = \alpha_j - \beta\frac{\partial L}{\partial \alpha_j} \tag{2.14}$$

The iterative updates at each step thus update the parameters to yield a different fit and can be illustrated as shown in figure 2.6 to converge to the best fit.

Adaptive Gradient (AdaGrad):

As the name suggests, AdaGrad is an adaptation of gradient descent optimizer with an adaptive learning rate β. It maintains a cumulative sum S of squared gradients as shown in equation 2.15.

$$S(t) = S(t-1) + \frac{\partial L}{\partial \alpha_j}^2 \tag{2.15}$$

The learning rate in each parameter update is then amended as shown in equation 2.16.

$$\frac{\beta}{\sqrt{S(t)+\epsilon}} \tag{2.16}$$

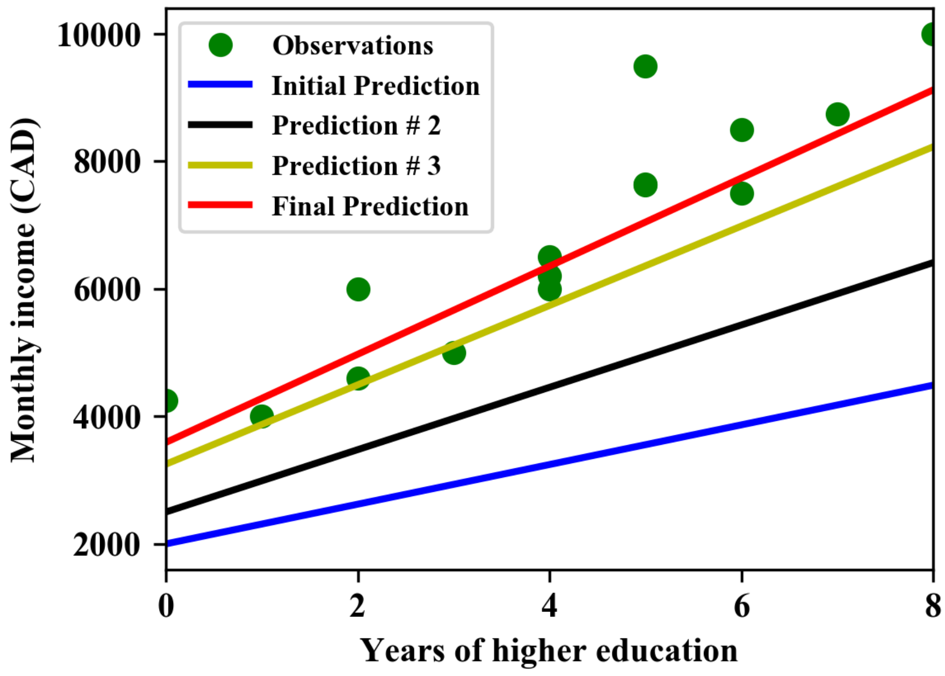


Figure 2.6: Gradient descent at work

where ϵ is a small floating-point number to ensure that division by zero does not occur. Compared to gradient descent, AdaGrad has the benefit of adjusting the learning rate to ensure that it is not too fast, to skip over the minima, and not to slow to take too long to converge. It also adjusts the rate for each parameter individually.

Root Mean Square Prop (RMSProp):

RMSProp is a variant of AdaGrad where instead of maintaining the cumulative sum of the squared gradients, an exponential moving average is maintained as shown in equation 2.17, where γ is the scaling factor.

$$S(t) = \gamma S(t-1) + (1-\gamma)\frac{\partial L}{\partial \alpha_j}^2 \tag{2.17}$$

RMSProp leads to faster convergence as compared to AdaGrad where the gradient gets smaller over time because of the cumulative term.

2.1.5 Deep learning

Inspired by the human brain, deep learning (DL) is a class of SL algorithms aimed at automatically discovering the representations in the underlying raw data. It has dominated accuracy benchmarks

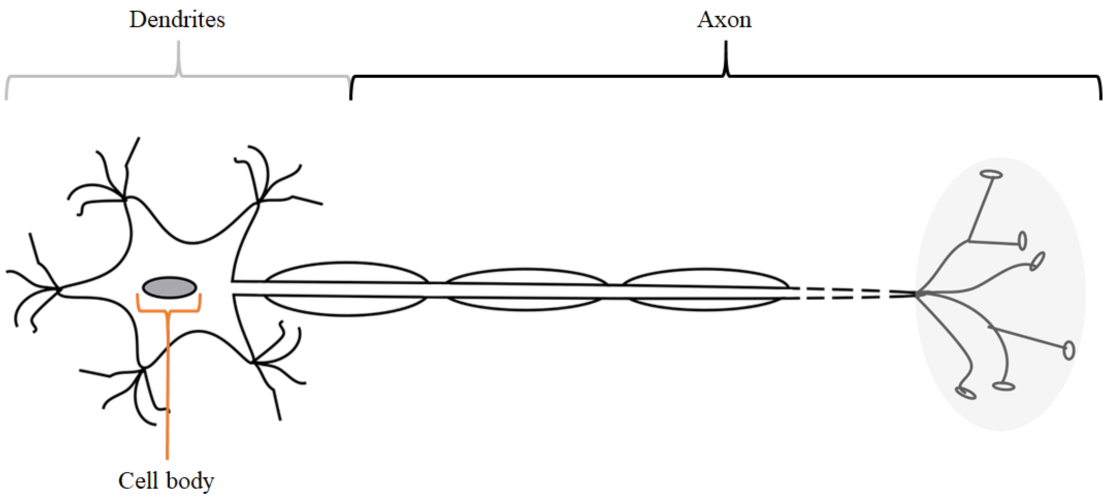


Figure 2.7: Biological neuron

over many fields especially in computer vision, medical diagnosis and data mining [11]. It builds on the model of the human brain to allow automatic learning from data the way human beings do. The basic building block of the human brain is a neuron as shown in figure 2.7. The dendrites are responsible for bringing the information to the cell body. The axon then carries the processed signal away from the cell body and out of the neuron.

Like the brain, a neural network is also an arrangement of neurons. Each node in the neural network is an artificial neuron such as the one illustrated in figure 2.8. Like the biological neuron, an artificial neuron receives information $[x_1, ..., x_N]$ from the neurons preceding it. The information received from each node is attenuated by its corresponding weight $[\alpha_1, ..., \alpha_N]$ and a summation is carried out over all the inputs as shown in the figure. A bias term is added and a nonlinear activation function is then applied to the sum resulting in the output of the neuron y as presented in equation 2.18.

$$y = f(\sum_{j=1}^{N} a_j, x_j + b) \tag{2.18}$$

The addition of the nonlinear activation function gives neural networks their approximating properties. A number of nonlinear functions have been used as activation functions including, most popularly, sigmoid, ReLU and tanh depending on the properties required. These are touched upon in subsection 2.1.5 as each activation function has its unique pros and cons.

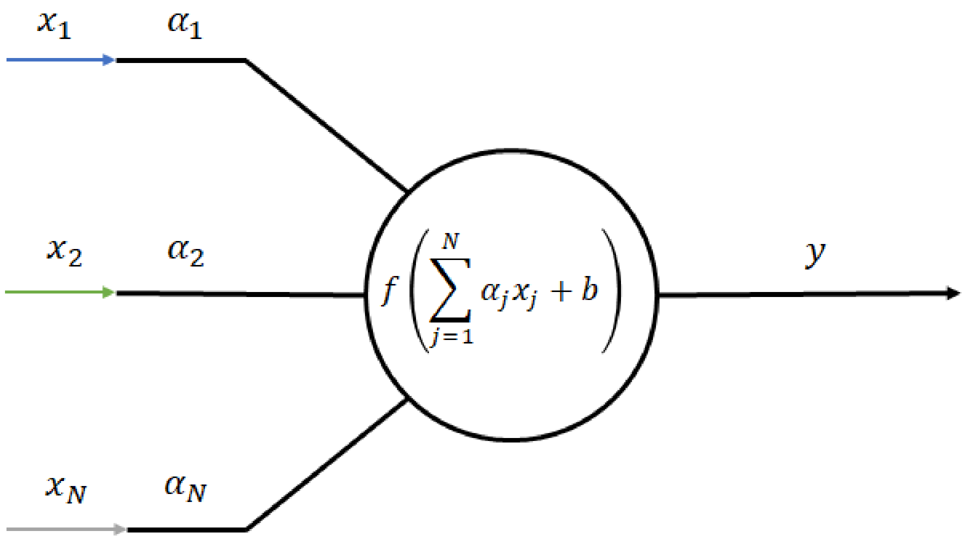


Figure 2.8: Artificial neuron

Neural Networks

Artificial neurons are joined to form what is known as a neural network. The universal approximation theorem states that feed-forward neural networks with a single hidden layer containing a finite number of neurons can represent a variety of continuous functions given the appropriate parameters [12]. To elaborate, the neural network structure is explained with the help of figure 2.9.

Each neural network has an input layer with the number of nodes equal to the input dimensions N. Each input node is connected to every node in the next hidden layer. There is at least one hidden layer in a neural network. Each node in the hidden layers is a neuron with its weights and bias. Each node then connects to each node in the next layer until the output layer is reached. The output layer has nodes equal to the dimensions of the output M.

Activation functions: The activation function applied to each layer in the neural network has to be chosen with regard to the overall objective of the neural network. Each node in the input layer only carries forward a single input. The activation function used in the output layer is determined by whether it is a regression or classification task. For the hidden layers, the choice of activation function depends on the understanding of the task. Activation functions used most commonly are described below with their pros and cons.

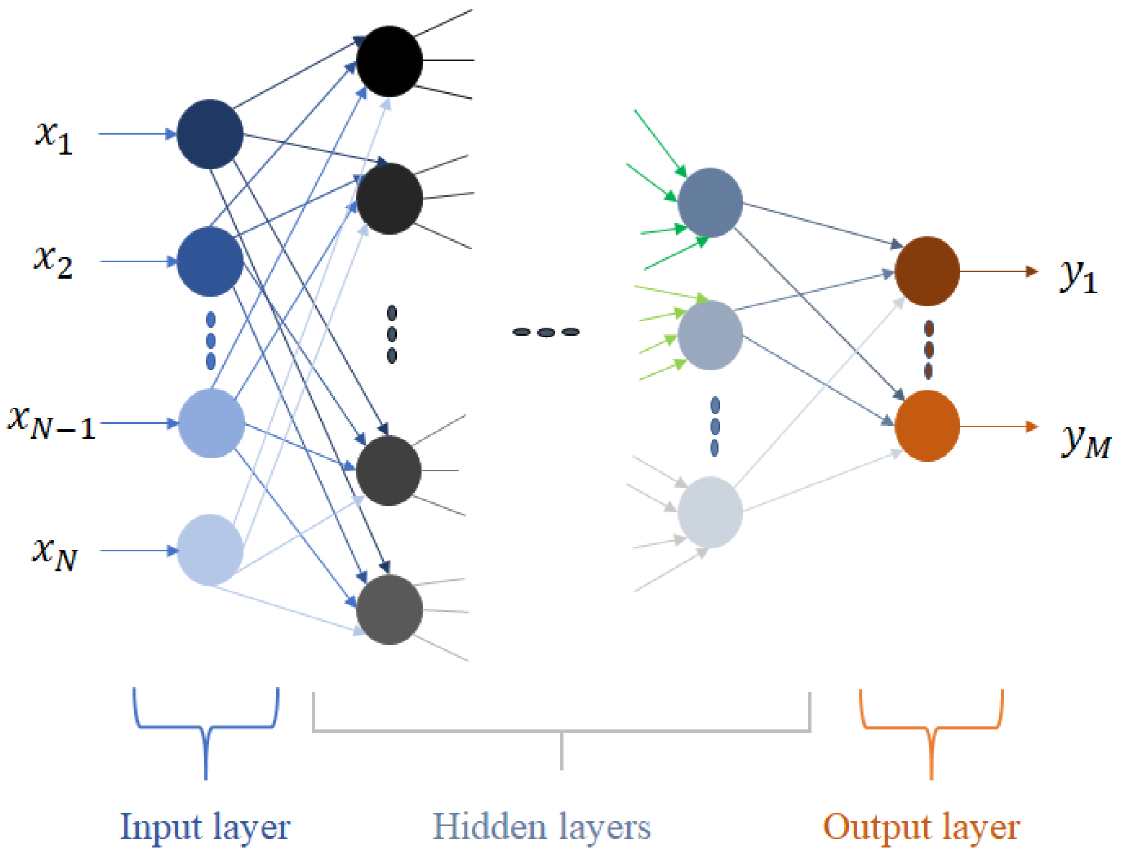


Figure 2.9: Neural network architecture

Step function: Step function is the simplest activation function which is described by equation 2.19.

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.19}$$

The step function is an excellent candidate for the output layer in classification cases where 1 represents a positive and 0 represents a negative for the class. The step function, however, does not present a derivative and should not be employed in the hidden layers as no learning occurs.

Sigmoid: Sigmoid function was the first function that was able to demonstrate universal approximation in a neural network and is still used extensively. It is represented by equation 2.20

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.20}$$

From the equation and the corresponding plot of the sigmoid function given in figure 2.11, it is evident that the output of a node with a sigmoid activation is scaled between 0 and 1. This presents

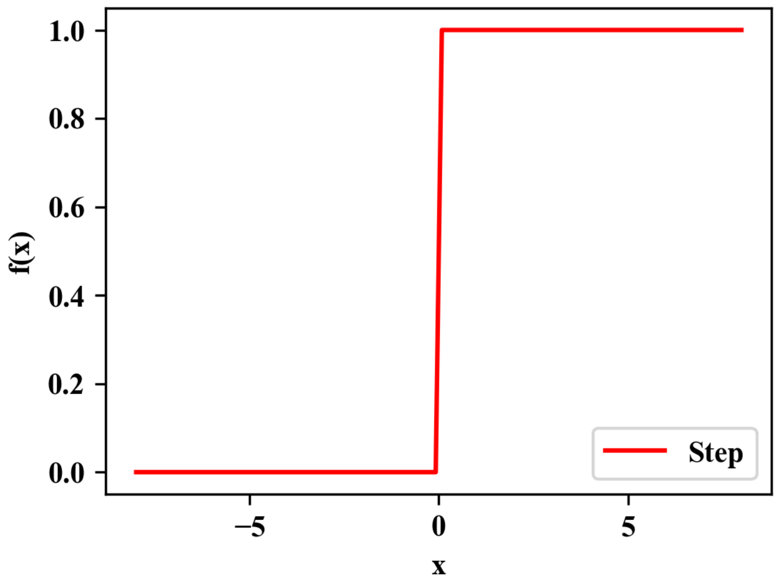


Figure 2.10: Step activation function

the problem of vanishing gradients as the output value changes very little when the input values move from 0 to $\pm\infty$, so learning is minimized. This will be explained later when backpropagation is introduced.

Tanh: Tanh is represented simply as the name suggests by equation 2.21. Its structure is similar to the sigmoid function but with a greater range of -1 to 1 and a steeper derivative as shown in figure 2.12.

$$f(x) = tanh(x) \tag{2.21}$$

It is thus more efficient in terms of learning because of the wider range and steeper gradient. Eventually, the same problem of vanishing gradient still applies to tanh, however, as the output does not change much with the change in input as the inputs move towards $\pm\infty$.

Rectified Linear Unit (ReLU): ReLU is also able to successfully approximate functions. It is represented by equation 2.22.

$$f(x) = max(0, x) \tag{2.22}$$

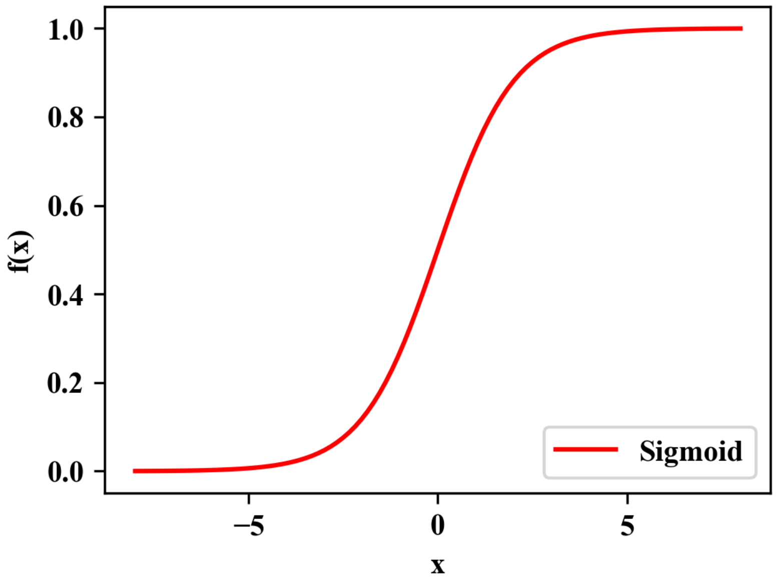


Figure 2.11: Sigmoid activation function

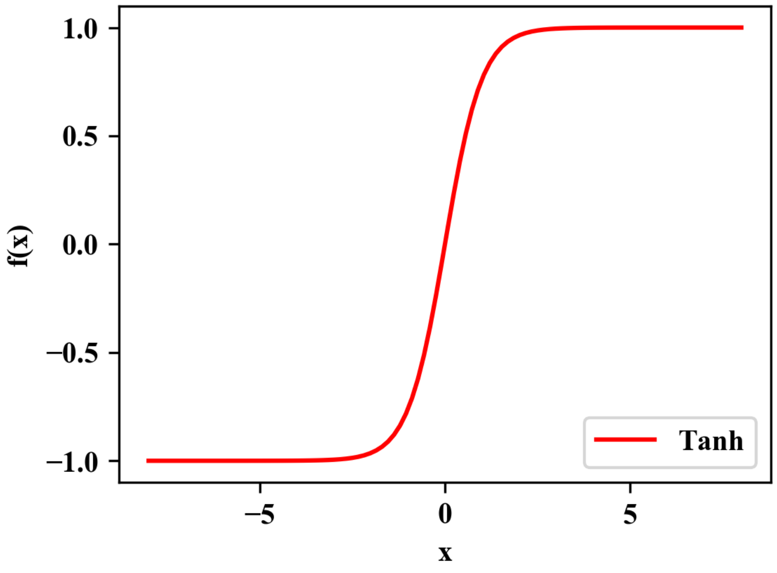


Figure 2.12: Tanh activation function

ReLU follows the linear function for values from 0 to $+\infty$. However, it saturates all input less than zero as shown in figure 2.13. This means that no learning occurs in that area. ReLU, however, remains popular because it has a wider range for positive input values and it reduces the computational load on the neural network due by switching off the nodes with a sum less than zero.

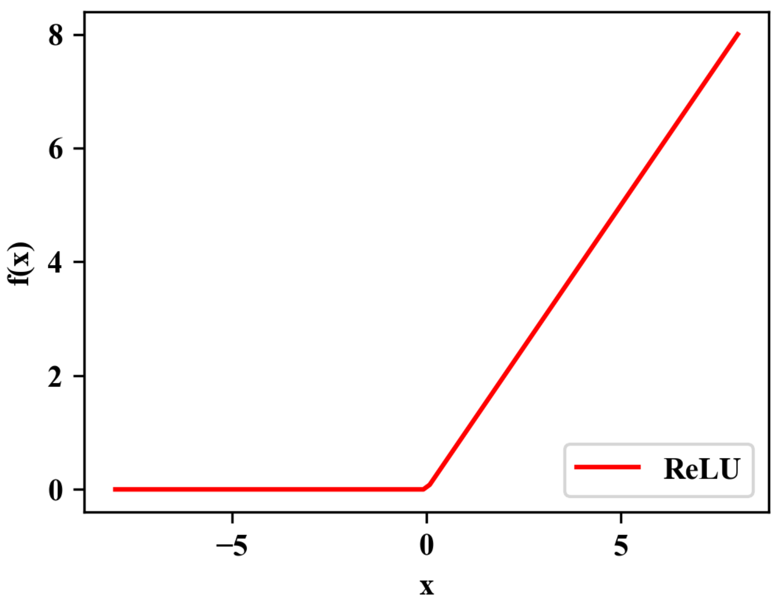


Figure 2.13: ReLU activation function

Neural network training: The training of a neural network follows the steps represented below:

1. Pre-processing data

   As is the standard for all SL algorithms, data is divided into training and test datasets after scaling and normalizing the whole data set. This ensures that no bias is introduced. As deep learning requires more data, and training data is divided into mini-batches for the training episodes.

2. Initialization of neural network parameters

   The neural network parameters, weights corresponding to each node-node connection and biases corresponding to each node, are randomly initialized. The learning rate is also set.

3. Feed-forward pass

   The inputs from the mini-batch are provided to the input layer, and forward calculations are performed from the input to the hidden layer and so on, until the predicted output is obtained from the output layer and stored.

4. Cost function calculation

   After each mini-batch, also known as a learning iteration or epoch, the cost is calculated as the deviation of the predicted output (obtained from the output layer) from the actual output in the mini-batch.

5. Backpropagation

   After calculating the cost, the neural network calculates the backward propagation of errors, i.e. the derivative of the error (cost function) with respect to the parameters of the neural network. Starting with the weights of the connections and the biases to the output layer and then going back to the input layer.

6. Optimization

   The designated optimization algorithm such as gradient descent (explained in subsection 2.1.4) is then employed to update the neural network parameters (weights and biases).

Steps 3 to 6 are repeated for all mini-batches or until the neural network parameters converge at the minimum cost. To put this all together mathematically, a simple 3 layer neural network presented in figure 2.14 below will be considered.

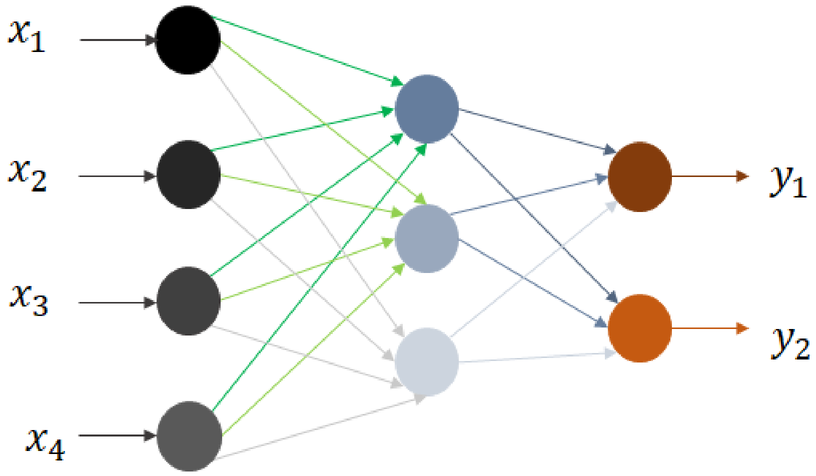


Figure 2.14: Example neural network

Step 1: Preprocessing data Assuming a data set of 1000 input-output pairs with input $X = [x_1, x_2, x_3, x_4]$ and output $Y = [y_1, y_2]$, the data will be normalized with their mean and standard deviation. It will then be shuffled and split into training and test data with a split of

70% to training and 30% to test. With 700 samples for training, the data will then be divided into mini-batches. Assuming a mini-batch of 100 samples, there will be 7 mini-batches available for training.

Step 2: Initialization of neural network parameters In this 3-layer network, there will be two sets of neural network parameters that will be randomly initialized to values between zero and one:

1. Weights for the connections between the nodes of the input and hidden layer ($\alpha_{ij}^h \mid i \in 1,2,3,4$ and $j \in 1,2,3$ and the bias term of each node in the hidden layer $b_j^h \mid j \in 1,2,3$, where h denotes the hidden layer, i denotes the node in the input layer and j denotes the node in the hidden layer.

2. Weights for the connections between the nodes of the hidden and output layers ($\alpha_{ij}^o \mid i \in 1,2,3$ and $j \in 1,2$ and the bias term for each node in the output layer $b_j^o \mid j \in 1,2$, where o denotes the output layer, i denotes the node in the hidden layer and j denotes the node in the output layer.

The activation functions $f(x)$ for each layer would also be chosen to be a sigmoid, tanh, or ReLU.

Step 3: Feed-forward pass The input from the mini-batch will be fed to the neural network for the feedforward pass. In this step, the forward calculations will be carried out to obtain the outputs of the neural network. The forward pass will occur sequentially. The inputs would be passed to the hidden layer to produce the outputs of the hidden layer z_j^h, where $j \in 1,2,3$ represent the node in the hidden layer, as shown in equation 2.23.

$$z_j^h = f(\sum_i \alpha_{ij}^h x_i + b_j) \tag{2.23}$$

Sequentially, this output will be fed to the next layer to obtain the output as shown in equation 2.24, where now $i \in 1,2,3$ represents the nodes in the previous layer (input to this layer) and $j \in 1,2$ represent the nodes in the current layer.

$$y_j = f(\sum_i \alpha_{ij}^o z_i^h + b_j) \tag{2.24}$$

Cost function calculation: The cost function to be employed is determined from amongst MSE, MAE. After each mini-batch, the cost is calculated by taking the deviation of the predicted output value $\hat{Y} = [y_1, y_2]$ from the corresponding actual output value $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}]$. The cost over the entire mini-batch is calculated. Taking MSE as example, this cost calculation is presented in equation 2.25.

$$L(X, \alpha, b) = \sum (\hat{Y} - \mathbf{Y})(\hat{Y} - \mathbf{Y})^T \tag{2.25}$$

Backpropagation: Backpropagation finds the contribution of each parameter (weights and biases) towards the final cost function. Starting from the output layer and propagating backward and solving using chain and product rules, backpropagation solves for the partial derivatives of cost w.r.t. each parameter:

$$\frac{\partial L(X,\alpha,b)}{\partial \alpha_{ij}^h}, \frac{\partial L(X,\alpha,b)}{\partial b_j^h}, \frac{\partial L(X,\alpha,b)}{\partial \alpha_{ij}^o}, \frac{\partial L(X,\alpha,b)}{\partial b_j^o}$$

Optimization: The final step in training is optimization. Using the gradient descent optimizer as an example, the parameters of the neural network are updated with a fixed learning rate β as given in the following equations:

$$\alpha_i j^h = \alpha_i j^h - \beta \frac{\partial L}{\partial \alpha_i j^h} \tag{2.26}$$

$$b_j^h = b_j^h - \beta \frac{\partial L}{\partial b_j^h} \tag{2.27}$$

$$\alpha_i j^o = \alpha_i j^o - \beta \frac{\partial L}{\partial \alpha_i j^o} \tag{2.28}$$

$$b_j^o = b_j^o - \beta \frac{\partial L}{\partial b_j^o} \tag{2.29}$$

Steps are repeated from feed-forward pass to optimization for all mini-batches or until convergence occurs.

Recurrent Neural Networks

While neural networks are able to approximate a wide variety of continuous functions, they only learn the relationship between the output and the input at an instant. This is due to the assumption that the current output is an explicit function of the current input. Moreover, they accept a fixed size input vector and produce a fixed size output vector. This is often not the case in the real world where systems and functions have sequential or temporal relationships and the output often depends on past values.

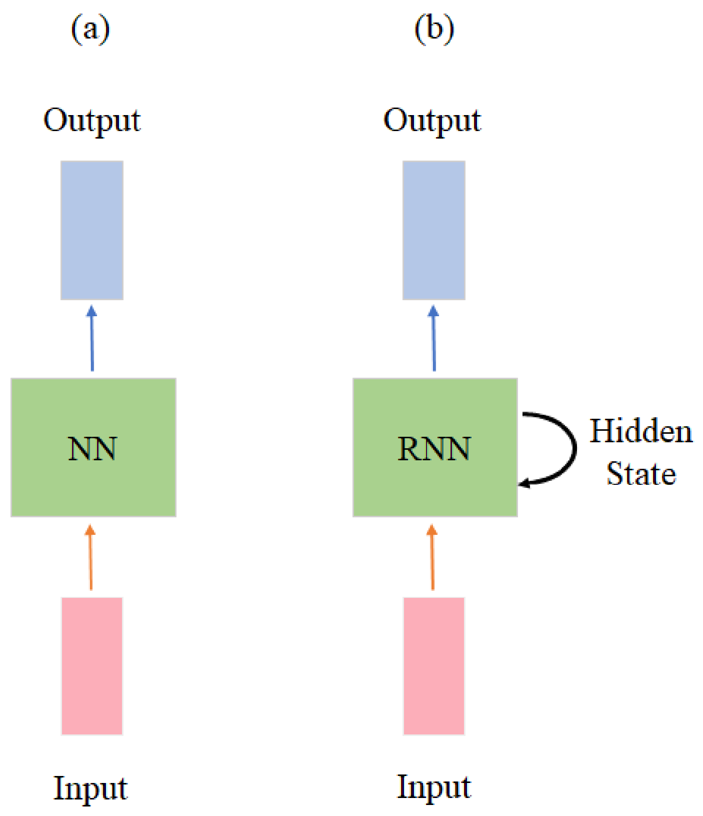


Figure 2.15: (a) Neural network (b) Recurrent neural network

Recurrent neural networks (RNN) are a variant of neural networks that incorporate information from the past states into the output estimated for the current state. This is done by means of a hidden state in the RNN that is fed back as an input as shown in figure 2.15. As seen, while a (vanilla) neural network takes an input at every instant and produces an output, the RNN has a hidden state that it updates at every time instant and it is considered along with the input at that instant to produce the output. The vanilla neural network's output at an instant y_t can be presented as a function f_α of the input x_t as shown in equation 2.30,

$$y_t = f_\alpha(x_t) \tag{2.30}$$

where f_α represents a neural network with parameters α. A RNNs output is dependent on its hidden state h_t as shown in equation 2.31. The hidden state is updated at each time instant as shown in equation 2.32.

$$y_t = g_\alpha(h_t) \tag{2.31}$$

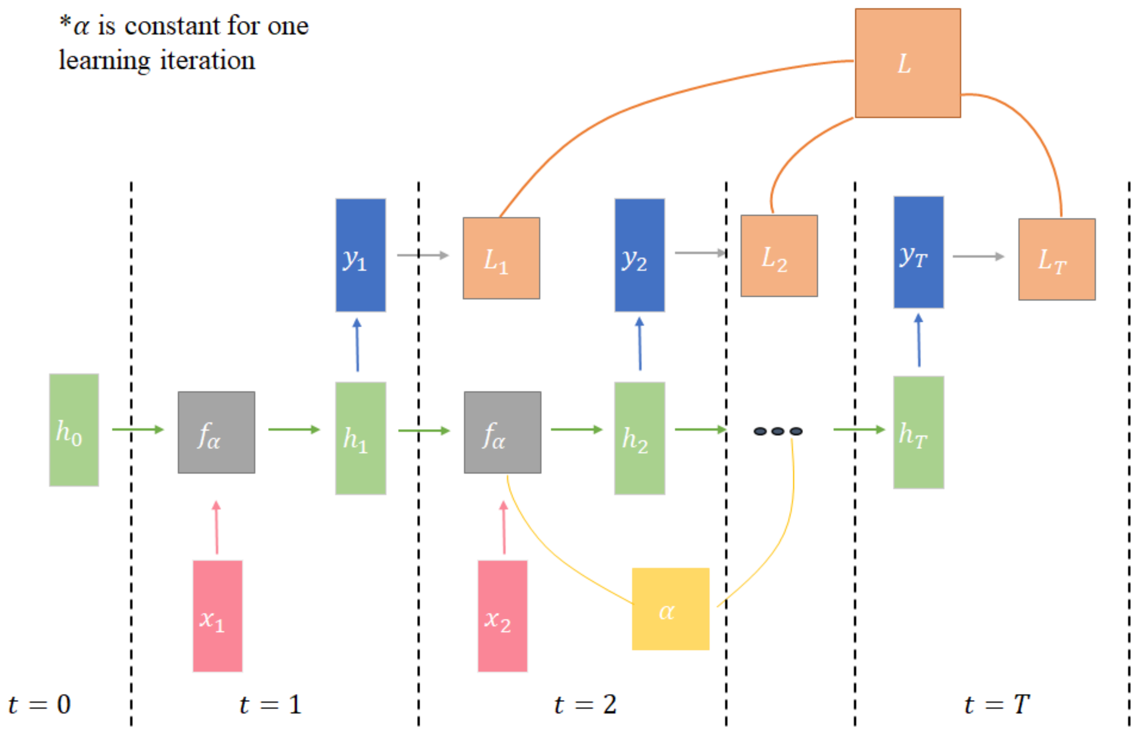


Figure 2.16: Working of a RNN

$$h_t = f_\alpha(h_{t-1}, x_t) \tag{2.32}$$

How a RNN processes sequential data with fixed size input and fixed size output is illustrated in figure 2.16. At time $t = 0$, the hidden state is initialized to h_0. Then at time $t = [1, .., T - 1]$, the neural network updates its hidden state h_t using its past hidden state and its current input x_t as given in equation 2.32. The output y_t is obtained from the hidden state as given in equation 2.31. This continues until the terminal output y_T is obtained. The loss L_t is then calculated from each output and summed (L) over the learning iteration. The RNN parameter α remain constant throughout a learning iteration. Backpropagation and optimization is then carried out as usual.

This also allows RNNs to be used for variable size input and variable size output problems. The difference is that the RNN reads the input (in variable-size input case) or produces the output (in the variable size output case) until the END token is parsed. Compared to a vanilla neural network, there are a number of cases that the RNN operates other than the fixed-size input to fixed-size output case:

1. Variable-size input to fixed-size output (Many to one):

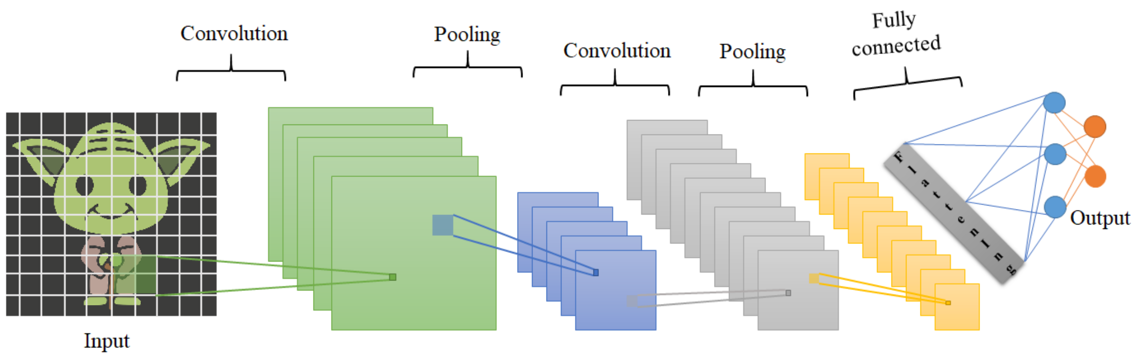


Figure 2.17: Convolutional neural network

Use cases like sentiment classification where a neural network might be fed a sentence of variable length and has to classify whether a specific sentiment (e.g. happiness) is portrayed in the sentence.

2. Fixed-size input to variable-size output (One to many):
   Cases like image captioning where images of fixed size are fed to the RNN, and it produces a caption describing the image with variable number of words.

3. Variable-size input to variable-size output (Many to many):
   Applications such as machine translation in which the RNN reads a sentence in one language and translates it into another language.

RNNs can also be cascaded to form multilayer RNNs. Some variants of RNN such as the Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) have gained popularity for their ability to capture temporal relations.

Convolutional Neural Networks

Convolutional neural networks (CNN) are a specialised kind of neural networks that are particularly skilled at feature extraction from data. Inspired by the organization of the visual cortex, they gained prominence for their image processing capabilities. CNNs make assumptions about the underlying data structure to extract useful trends. They are hence able to better capture the spatial and temporal relations in data such as images and time-series data through the application of relevant filters. A typical CNN is presented in figure 2.17. The layers that make up a CNN are:

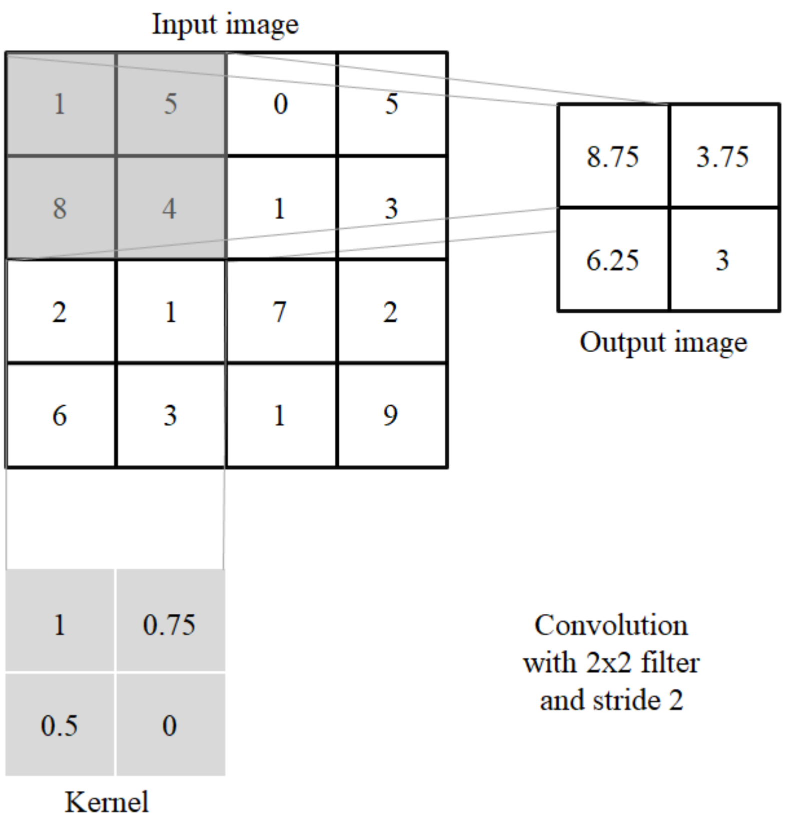


Figure 2.18: Convolution

Convolution layer: The convolution layer consists of kernels/filters. The number of kernels and their size is defined beforehand. In the case of images, for instance, the kernel weights convolved with the pixel values that the kernel covers provide one output pixel. Each kernel parses the whole image once to create an output image. Then the number of output images from the convolutional layer, therefore, equals the number of kernels. The size of the output image is also determined by the size of the kernel. The size of the output W_o depends on the size of the input W_i, the size of the kernel W_k and the stride S of the kernel (i.e. how many pixels the kernel moves). Convolution is presented in figure 2.18.

Padding layer: The padding layer is used to augment the image size to adjust the size of the output as required. Zero-padding, in which pixels of size 0 are added, is the most commonly used one. The output size hence can be presented as:

$$W_o = \frac{W_i - W_k + 2P}{S} + 1$$

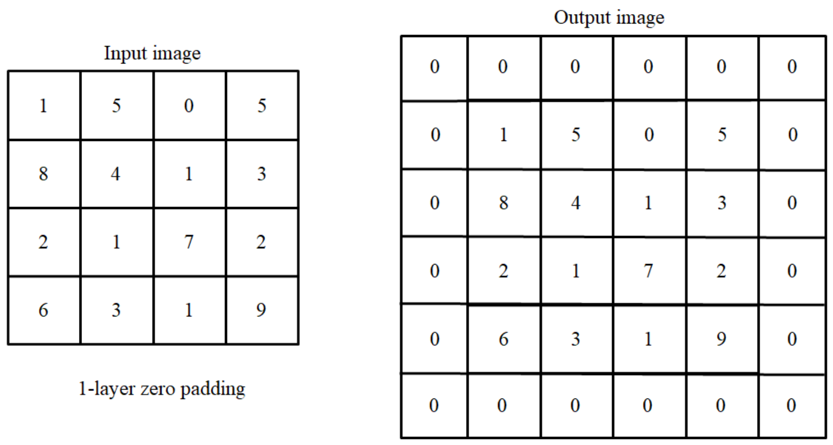


Figure 2.19: Padding

where P is the size of the padding. Applying 1-layer padding to the image from figure 2.18, results in figure 2.19.

Pooling layer: The pooling layer is used to extract the dominant features by reducing the dimensions of its input. They do this by parsing through the spatial or temporal data in blocks. Two types of pooling layers are commonly used: 1) max-pooling that outputs the maximum value from the block, and 2) average-pooling that outputs the average of the block. Both max-pooling and average-pooling are demonstrated in figure 2.20.

Dropout layer: Dropout layer is used to reduce overfitting the CNN. It does so by allowing different CNN architecture configurations during training. Some of the layer outputs are randomly dropped (made 0). This way the number of nodes and their connections to the previous layer appear different to the CNN. This, hence, reduces the chances of overfitting.

Fully connected layer: The features extracted from these layers are fed to fully connected layers after flattening so that non-linear combinations of the high-level features can be learned. They make up the final layers.

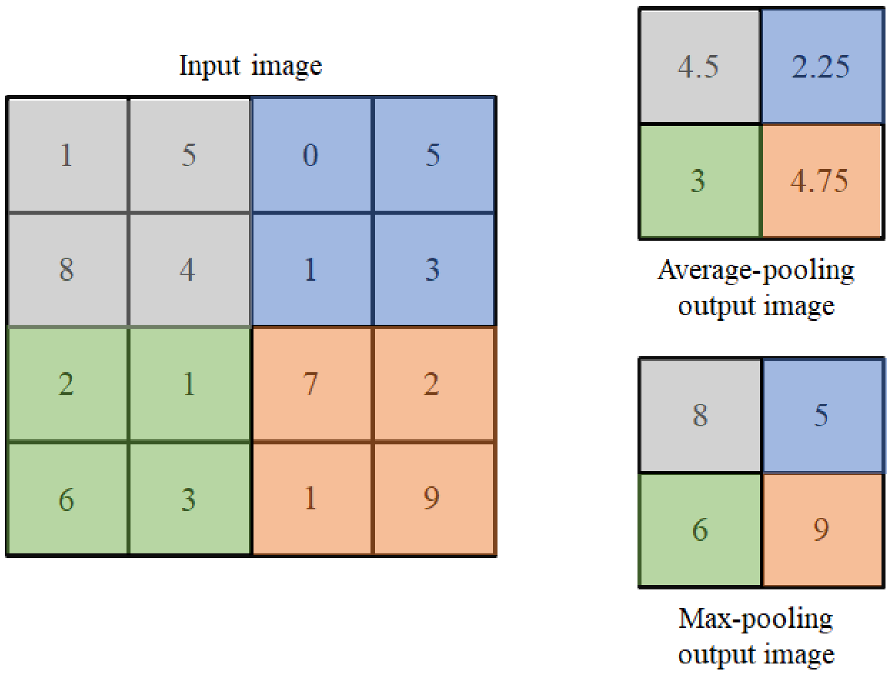


Figure 2.20: Pooling

2.2 Unsupervised learning

Unsupervised learning algorithms are used when unlabelled data is available and it needs to be grouped and/or interpretations have to be drawn from it. These algorithms attempt to uncover underlying trends in unlabelled data. They are also used to model the underlying probability distribution of the input data.

Semi-supervised algorithms lie between supervised and unsupervised learning. They use both labelled and unlabelled data to train. It is able to improve accuracy by using the unlabelled data to learn the underlying structure and the labelled data to improve accuracy.

2.2.1 Clustering

Unsupervised learning algorithms are used majorly for clustering. Unlabelled data is provided to an unsupervised learning algorithm that groups the data samples on the basis of the similarity between them. Some of the popular clustering algorithms are explained in the subsections below.

k-means clustering

This clustering algorithm is intended to divide the given input data $X = [x_1, ...x_N]$ into k clusters. Each data sample is assigned to the cluster with the closest mean value to the sample's value. This is done by dividing the data into k clusters ($C_i \mid i \in 1, ..., k$) and finding the clusters such that within each cluster, the sum of squares is minimized as given in equation 2.33.

$$\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} \mid\mid x - \mu_{C_i} \mid\mid^2 \tag{2.33}$$

Two steps that are followed iteratively until the objective function in equation 2.33 converges: 1) assignment, in which each sample is assigned to the cluster with the closest mean value, and 2) in which the means of the new clusters are calculated.

Hierarchical clustering

Hierarchical clustering has two ways to go about it: 1) Agglomerative, where each sample point starts as an individual cluster, and 2) Divisive, where the whole dataset starts as a single cluster.

Both of these are iterative processes that are the opposite of each other. In agglomerative hierarchical clustering, at every time step, the proximity is calculated between each cluster and the two closest clusters are merged. After the merging, the proximity is recalculated and merging reoccurs. This is repeated until only a single cluster remains.
In divisive type, at every time step, the dissimilar samples are removed from the cluster until every data sample is an individual cluster. The results of hierarchical clustering are obtained in the form of a dendrogram that records the sequence in which the merges or the splits occurred.

2.2.2 Neural networks

Neural networks have also been used in unsupervised learning, the most popular example of which is self-organizing maps (SOM). The underlying principle of SOMs is that if two inputs cause similar outputs from the same neurons, they must lie close to each other spatially. SOMs are used for dimensionality reduction by taking higher dimension inputs that are together spatially and by maintaining that topology in a lower-dimensional representation.
SOMs operate in two modes: 1) training, in which nodes compete for the right to represent certain subsets, and 2) mapping, in which data is then mapped to its lower-dimensional 'neighborhood'.

2.3 Reinforcement learning

RL is formulated differently from supervised learning and unsupervised learning in the sense that it does not look at labelled/unlabelled data. It is goal-based learning that occurs over the experience gained through an RL agent's interaction with an environment. RL algorithms learn by interacting directly with the environment to sample the optimal actions in order to achieve a specified goal [13].

Inspired by animal psychology, the RL agent learns the actions to take, based on the cumulative long-term rewards achieved over a defined period. Initially, the breakthrough in RL came from the successful integration of optimal control theory with animal psychology. RL borrows its formal structure from optimal control where the objective is to design a controller to minimize a measure of a dynamical system's behavior over time [14]. The approach towards solving this problem considers the state and value function of the dynamical system and is known as the Bellman optimality equation. The solution to this equation was the purview of dynamic programming, and it is from there that the discrete stochastic version of the optimal control structure, MDP, comes from.

Although the structure is taken from optimal control, the learning in RL is borrowed from animal psychology. Temporal-difference (TD) learning combined trial and error learning in computing with psychology to emerge as a feasible approach for large systems in the 1970s [15]. This idea was developed further by merging it with optimal control in the 1980s to form the classical conditioning model based on TD learning of the optimal policy as the archetype of reinforcement learning as we now know it [13].

2.3.1 MDP formulation

The formal structure of RL is represented by a Markov decision process (MDP). An MDP is assumed to follow the Markov property which states that the current state of the environment is sufficient to take the optimal action, i.e. the optimal action to be taken in this state does not depend on the past states of the environment. The MDP structure is illustrated in figure 2.21.

MDP comprises of an RL agent that is the learner in the process and the environment with whom the agent interacts to optimize a certain facet of its behavior. The MDP encapsulates the agent-environment interaction in discrete time steps within the length of the learning episode $t \in N$. It is assumed that at each time instant t, there is a set of states $s_t \in S$ that the environment can assume. There is also a set of actions $a_t \in A$ the agent can choose given the state observation s_t according to its policy $\pi(a_t \mid s_t)$. By virtue of the action a_t, the environment transitions to a new

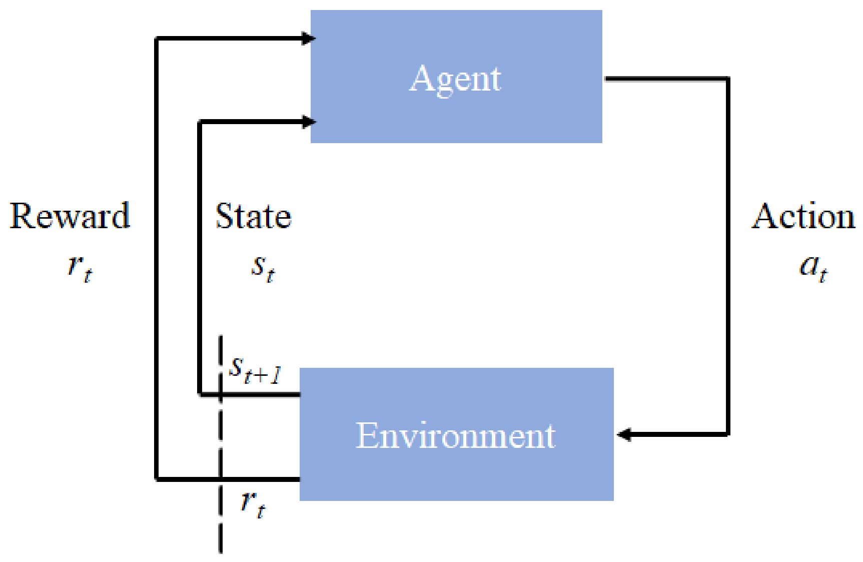


Figure 2.21: Markov decision process

state s_{t+1} and emits a scalar reward r_t associated with being in that state. The reward hence is the means through which learning occurs, and reward shaping is an important part of the RL problem [13]. These rewards accumulated over time by following the policy π and corrected by a discount factor $\gamma \in (0, 1)$ that determines the relative importance of future rewards are known as returns R_t (equation 2.34). Returns are direct feedback on the agent's performance at each state, and their estimate is known as the value function $V(s)$ as shown in equation 2.35.

$$R_t = \sum_{j=1}^{k} \gamma^j r_j \tag{2.34}$$

$$V(s) = E_\pi(R_t \mid s) \tag{2.35}$$

where E_π is the expectation from following the policy π. If the action taken is also considered, they constitute the action-value function $Q(s, a)$, as shown in equation 2.36.

$$Q(s, a) = E_\pi(R_t \mid s, a) \tag{2.36}$$

The transition probability function $P(s', r \mid s, a)$ represents the probability of the environment transitioning to a new state s' and emitting a reward r if action a is taken by the agent in state s. The core objective of an RL algorithm is then to find the optimal policy $\pi^*(a \mid a)$ that maximizes the overall returns as given in equation 2.37,

$$\max_{\alpha} J(\alpha) = E_{\pi_\alpha}(R_t \mid \pi_\alpha) \tag{2.37}$$

where π_α represent the policy with parameters α.

Although various schemes are available to optimize the RL objective, the following are widely utilized: model-based, value-based, policy gradient and actor-critic [16]. Model-based algorithms focus on learning the transition probability function $P(s', r \mid s, a)$ of the environment and use that to find the optimal policy. The focus of this thesis is on the model-free algorithms. The other three algorithm classes fall under the model-free category. With the framework in place, subsections 2.3.3 to 2.3.5 focus on these three algorithm classes to optimize the objective function presented in equation 2.37.

2.3.2 Generalization

The concept of generalization is integral to RL as much as it is to SL. While in SL, generalization refers to the SL algorithm's ability to perform well on regressing/classifying previously unseen data. In RL, it means finding a policy that is able to perform well in an environment where limited training was carried out as well as in environments similar to the one it was trained on [17]. This is also important for cases such as when noise is introduced to the state observations or a process shift happens in the environment. The ideas that are important for successful generalizations are presented in this subsection.

Exploration-exploitation

A concept central to RL is that of the exploration-exploitation trade-off. Exploration is taking an equal probability random action to explore the action space while exploitation is taking the action that has previously resulted in the highest reward of all the actions that have been attempted in that state.

The benefit of acquiring more information about the state and action space that might lead to better rewards through exploration has to be balanced with the possible loss of rewards that could have otherwise been gained by taking the best decision with the current information, i.e. exploitation. The possible loss of rewards is termed cumulative regret. The way each algorithm class handles the exploration-exploitation trade-off is detailed in their own subsections.

Bias-overfitting

The bias-overfitting trade-off is tied to feature selection in RL. If a high number of features are observed by the RL agent, it might base its policy on spurious correlation between features that might not represent a causal relationship leading to overfitting. On the other hand, omitting features that might have a causal relationship with the rewards can lead to a bias factor. The bias-overfitting trade-off can also be managed through the inclusion of appropriate factors in reward shaping and cost function.

On-policy & off-policy algorithms

On-policy RL algorithms are algorithms that interact with the environment using the same policy that they update as they learn, i.e. there is only one policy. On the other hand, off-policy algorithms have a separate update policy, that learns the optimal policy, and a separate behaviour policy, that determines the action the RL agent takes to interact with the environment.
So while both on-policy and off-policy algorithms are learning to find the optimal policy, the latter does not take the policy that is being updated into account while interacting. These concepts will be elaborated further as specific algorithms are discussed in the later subsections.

2.3.3 Value-based algorithms

Value-based algorithms focus on accurately estimating the optimal value function $V^*(s)$ or the optimal action-value function (Q function) $Q^*(s, a)$ which automatically takes into consideration the transition probability model. Once the optimal value/action-value function has been determined, the optimal policy $\pi^*(a \mid a)$ is selecting the actions for a given state with the highest values.

Q-Learning

The most popular value-based algorithm is Q-learning which was initially introduced in 1989. It was instrumental in the initial popularity of RL [18]. Q-learning was initially suggested for a discrete state-action space where a Q-matrix can be constructed using state-action tuples. The Q-matrix is meant to store the action-value function, i.e. the expected returns from taking that action in that state. The Q-matrix is initialized to zero, and then the RL-agent follows an ϵ-greedy policy to interact with the environment. The ϵ-greedy policy caters to the exploration-exploitation trade-off by setting a probability ϵ that the RL agent will take an equal probability random action in the given state, while it will take the action with the highest Q-value with a probability of $(1 - \epsilon)$.

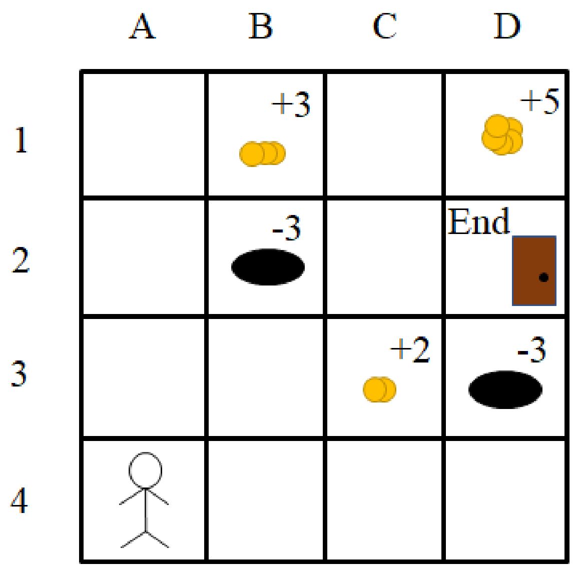


Figure 2.22: Q-learning problem

To learn further about Q-learning, and RL in general, an illustrative example is employed. The environment is the grid environment presented in figure 2.22, and the RL agent is the stick figure at block A4. The states of the environment are the blocks of the grid as shown in equation 2.38.

$$s_t \in S|S = [A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, D1, D2, D3, D4] \tag{2.38}$$

The actions that the RL agent can take are presented in equation 2.39.

$$a_t \in A|A = [up, down, left, right] \tag{2.39}$$

except at the borders where the actions going into the border are illegal. The reward function is based on the points accumulated in the game (indicated on the blocks in the figure) before reaching the terminal state $D2$. The reward function can then be represented as in equation 2.40. Moreover, each action taken results in a reward of -1 until the terminal state is reached, so at each time instant, a second step is added to the reward calculation (equation 2.41).

$$r_t = \begin{cases} +3, & s_t = B1 \\ -3, & s_t = B2 \\ +2, & s_t = C3 \\ +5, & s_t = D1 \\ -3, & s_t = D3 \end{cases} \tag{2.40}$$

$$r_t = \begin{cases} r_t - 1, & s_t \neq D2 \\ r_t, & o.w. \end{cases} \tag{2.41}$$

Table 2.5: Q-matrix

Q-value	up	down	left	right
A1	Q(A1, up)	Q(A1, down)	Q(A1, left)	Q(A1, right)
A2	Q(A2, up)	Q(A2, down)	Q(A2, left)	Q(A2, right)
A3	Q(A3, up)	Q(A3, down)	Q(A3, left)	Q(A3, right)
A4	Q(A4, up)	Q(A4, down)	Q(A4, left)	Q(A4, right)
B1	Q(B1, up)	Q(B1, down)	Q(B1, left)	Q(B1, right)
B2	Q(B2, up)	Q(B2, down)	Q(B2, left)	Q(B2, right)
B3	Q(B3, up)	Q(B3, down)	Q(B3, left)	Q(B3, right)
B4	Q(B4, up)	Q(B4, down)	Q(B4, left)	Q(B4, right)
C1	Q(C1, up)	Q(C1, down)	Q(C1, left)	Q(C1, right)
C2	Q(C2, up)	Q(C2, down)	Q(C2, left)	Q(C2, right)
C3	Q(C3, up)	Q(C3, down)	Q(C3, left)	Q(C3, right)
C4	Q(C4, up)	Q(C4, down)	Q(C4, left)	Q(C4, right)
D1	Q(D1, up)	Q(D1, down)	Q(D1, left)	Q(D1, right)
D2	Q(D2, up)	Q(D2, down)	Q(D2, left)	Q(D2, right)
D3	Q(D3, up)	Q(D3, down)	Q(D3, left)	Q(D3, right)
D4	Q(D4, up)	Q(D4, down)	Q(D4, left)	Q(D4, right)

Now that the environment is defined, the Q-matrix $Q(s, a)$ can be described as shown in table 2.5. On initialization, this table would then be as presented in table 2.6, where na denotes the illegal actions, and t denotes the terminal state.

Now, this would be a finite horizon problem where the RL agent would learn until reaching the terminal state. At each time step, the agent takes action a_t using the ϵ-greedy policy, the environment transitions to a new state s_{t+1} and the agent receives a reward r_t. This whole tuple $[s_t, a_t, s_{t+1}, r_t]$ is then used to update the Q-matrix. The update to the Q-matrix is derived using gradient descent and is an approximate solution to the Bellman optimality equality as given in equation 2.42.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s, a)) \tag{2.42}$$

where α denotes the learning rate of the Rl agent, and γ represents the discount factor, i.e. the relative importance of future rewards. As this is repeated over time, the RL agent updates the Q-matrix and gains experience. For instance, with α and γ both set at 0.9 if the first action taken by the RL agent is *right*, the agent causes the observed state of the environment to transition from $A4$ to $B4$, the update would then be:

$$Q(A4, right) = 0 + 0.9(-1 + 0.9(0) - 0) = -0.9$$

where the term $Q(s_{t+1}, a_{t+1})$ represents the highest Q-value in the row for state $B4$. This is repeated until the Q-matrix is populated. As more experience is gained, Q-learning improves its knowledge

Table 2.6: Initialized Q-matrix

Q-value	up	down	left	right
A1	na	0	na	0
A2	0	0	na	0
A3	0	0	na	0
A4	0	na	na	0
B1	na	0	0	0
B2	0	0	0	0
B3	0	0	0	0
B4	0	na	0	0
C1	na	0	0	0
C2	0	0	0	0
C3	0	0	0	0
C4	0	na	0	0
D1	na	0	0	na
D2	t	t	t	t
D3	0	0	0	na
D4	0	na	0	na

of the environment and its returns. An n-step Q-learning also exists which updates the Q-matrix after every n-steps instead of after every 1-step.

2.3.4 Policy gradient algorithms

Policy gradient methods solve directly for the optimal policy $\pi^*(a \mid s)$. They first estimate the value function through summation of the observed rewards r_t. Then the value/action-value function is used to approximate the returns to provide the gradient of the objective function that the RL tries to optimize that was described in equation 2.37 to update the policy parameters using gradient ascent. The update in the policy parameters is being represented as given in the REINFORCE algorithm [13] shown here in equation 2.43.

$$\nabla_\theta V^{\pi_\theta}(s) = E_{\pi_\theta}[\nabla_\theta[\log \pi_\theta(s,a)][Q\pi_\theta(s,a)]] \tag{2.43}$$

Hence, policy gradient methods learn in two steps: 1) policy evaluation, in which the agent interacts with the environment to gain rewards and learn the value/action-value function, and 2) policy improvement, in which the learned cost function is used to improve the policy.

2.3.5 Actor-critic algorithms

This brings us to the third category, the actor-critic algorithms. Closely related to the policy gradient methods, the actor-critic methods combine the benefit of both value-based and policy

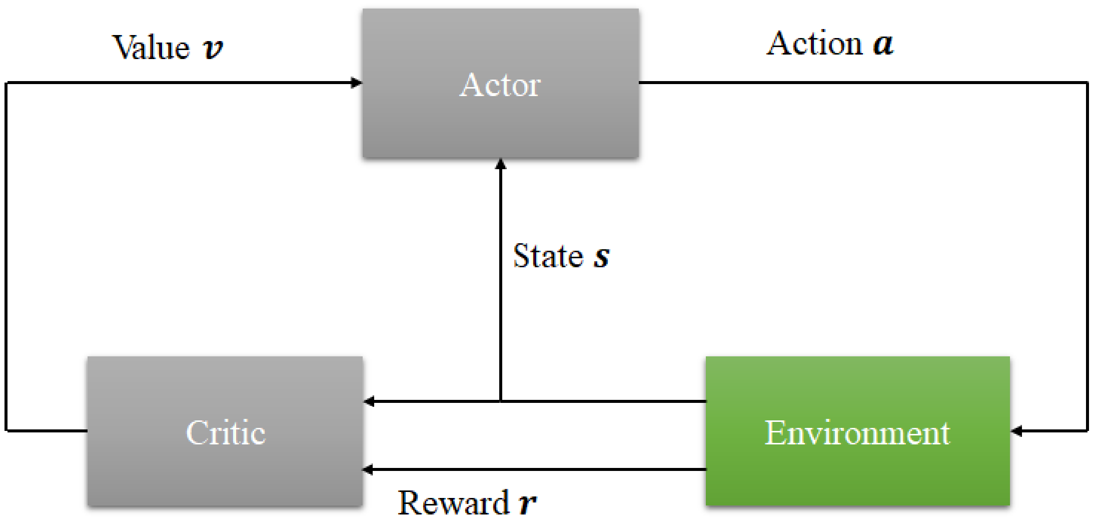


Figure 2.23: Actor-critic

gradient algorithms to represent the RL agent as a two-part learner consisting of a parameterized actor (the policy) and a parameterized critic (the value/action-value function) [17]. The actor-critic setup is represented in figure 2.23. They employ a Monte-Carlo kind of scheme where learning occurs over experience. Experience is gained from multiple repeated episodes to record an approximation for the action-value function[19]. This allows the actor-critic algorithms to be model-free, that is it does not require that the dynamics of the environment be known.

The actor observes the state s_t of the environment and selects an action a_t according to the policy $\pi(a \mid s)$. The critic observes the state s_t (and action a_t in the case of action-value function) and produces a value function output. The environment transitions to the new state s_{t+1} and emits a reward r_t. The actor-critic employs a storage buffer known as the experience replay which stores the $[s_t, a_t, r_t, s_{t+1}]$ tuple for a fixed length of time $t \in N$ in the case of infinite horizon problems or until the terminal state is reached in a finite horizon problem. After the experience replay buffer is filled, the returns R_t are calculated using the rewards stored, and the critic is then regressed to estimate the returns using MSE as criterion from the value/action-value function with the critic objective function given in equation 2.44.

$$\min_{\alpha_c} L(\alpha_c) = \sum_{t=0}^{N} \mid\mid R_t - Q_{\alpha_c}(s_t, a_t) \mid\mid^2 \tag{2.44}$$

where α_c represents the parameters of the critic. This episodic update method is also known as bootstrapping. This is where the last sample time reward is assumed to be its return and the returns for previous sample times are obtained by back-calculation from there. After the critic update, the actor policy is updated by means of the actor gradient as in the REINFORCE algorithm given in equation 2.43. The actor-critic thus follows this sequential update to find the optimal policy.

2.3.6 Deep reinforcement learning

The initial popularity of RL following the success of Q-learning was limited by the curse of dimensionality, i.e., its inability to solve high-dimensional problems because the number of calculations increases drastically with the number of states and actions, such as in Q-matrix calculations. A solution to this problem was proposed in the form of function approximators. With the universal approximation theorem proving the ability of neural networks to approximate continuous functions, neural networks were suggested as function approximators for use in RL problems, resulting in deep reinforcement learning (DRL).

They were initially employed to approximate the action-value function (Deep Q-Networks) for higher dimensional state spaces in RL problems [20]. This led to an RL breakthrough as it enabled control of continuous state spaces. It also popularized the concept of experience replay where the states, actions, and rewards are stored in memory to update the policy at a later time. Another important concept that was popularized after its use in DQN was that of a separate target network, hence promoting off-policy learning algorithms.

Continuous action space control was enabled with the introduction of DDPG algorithm which employed an actor-critic model-free architecture for control of more than 20 simulated physics tasks [21] in which both the actor and the critic were represented by neural networks.

Chapter 3

Reinforcement Learning for Multivariate Control

The contributions of this chapter have been submitted in:

Referenced Conference Publication:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Asynchronous Reinforcement Learning Based Continuous Space Control for Multivariable Process*, International Federation of Automatic Control, 2020. [submitted].

The contributions of this part have also been presented in:

Conference Presentation:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Reinforcement Learning Based Controller for Setpoint Tracking with Illustrative Application in a Quadruple Tank System*, 69th Canadian Chemical Engineering Conference, Nova Scotia, 2019

RL has received wide attention in various fields lately. Despite learning schemes like DDPG, DQN or actor-critic, convergence to an optimal policy in continuous processes remains a persistent challenge. This is mainly attributed to the continuous state and action space nature of chemical processes. A parallel learning architecture can improve convergence through better exploration of the state and action space. In this chapter, an asynchronous advantage actor-critic (A3C) scheme, which combines the benefits of parallel learning with better exploration through asynchronous updates, is investigated. A centralized agent-based servotracking of levels in a nonlinear interacting quadruple tank process is considered as an example. Training under varying degrees of parallel learning consistently resulted in the optimal control policy. Based on numerical simulations, the servotracking based metrics indicate a near-optimal performance by all policy cohorts. Hence, with minimal hardware, the A3C can be an effective RL approach for control of continuous chemical processes.

3.1 Introduction

A control system is designed to lead the system to the desired state and sustain it in the presence of external disturbances [16]. A model of the process is usually required for controller design. The apparent uncertainties built into this model can deteriorate the design goals. Obtaining complex models for accuracy and utilizing them in control design can become cumbersome. The non-stationary nature of processes requires constant intervention to accommodate process variations in closed-loop control. A model-free process data-oriented approach can greatly improve the control design in accommodating nonlinearities.

Lately, machine learning has gained recognition for its versatile real-world applications in areas such as recommendation systems [22], medical diagnosis [23], and natural language processing [24]. Introduced in the 1980s, reinforcement learning (RL) has been employed in various fields with an encouraging degree of success. RL is a branch of machine learning that builds on the essence of human decision making towards achieving a defined goal. It can incorporate a model-free approach in its learning strategy [25]. Real-world problems requiring goal-based decision making are probable candidates for RL based solutions. An agent learns through interactions with the environment to take actions that maximize rewards. Each run on the environment is usually terminated based on achieving the goal [13]. Though various schemes are available to implement RL, the following are widely utilized: model-based, value-based, policy gradient and actor-critic [16].

The state-action space exploration strategy greatly influences the stability in convergence to the optimal agent (henceforth termed policy). Value-based approach deals with a discretized state-action space, making it ideal for low dimensional state space problems [13], [26], [27], [28]. In earlier works, discrete action space problems [29] as well as continuous action space problems [30] have been successfully addressed using onpolicy proximal actor-critic approach. Although the use of non-linear function approximators such as neural networks enable continuous control, it can impact the convergence to an optimal policy [31], [21]. Therefore, asynchronous schemes are recommended to improve convergence [1].

In this study, a multi input multi output (MIMO) quadruple tank (QT) process is considered as an example [32]. It constitutes a multidimensional continuous state/action space. This chapter investigates development of an off-policy, asynchronous advantage actor-critic (A3C) based control scheme for servotracking of interacting levels in QT [1].

The contributions of this chapter are: Development of a reinforcement learning based control strategy that learns directly from interaction with a multivariate continuous process using a rewarding structure compliant with process control objectives. In particular, this reward structure for process control distinguishes itself from that commonly used in the general literature by considering both the output performance and the input effort. These methods utilize process data based on interactions with the experiment in order to tune the function approximator weights. This control strategy was implemented on a nonlinear process and servotracking was demonstrated. Results obtained from the off-policy advantage actor-critic algorithms (A2C) and its asynchronous variant, A3C, are discussed. Based on this, a study on the impact of the degree of parallel learning amidst asynchronous policy update on convergence and optimality of RL control is conducted.

3.2 Asynchronous advantage actor-critic algorithm

Deterministic policy gradient algorithms have long served in learning the optimal solution for continuous space environments [33], [31]. Owing to its low sample efficiency, the convergence to the optimal policy is very slow [16]. Recently, actor-critic algorithms using neural networks as function approximators have received attention [1], [16]. This off-policy approach involves sequential learning of predicting the discounted rewards and policy, guaranteeing convergence.

A non-parametric, experience-based data is utilized for training the critic network to predict the quality of an action in the temporal scale [34], [20]. This data is collected from the process interactions of the actor. This action assessment scheme is termed as experience replay [34], [35]. Hence, this model-free approach improves the generalization of policy using real interactions.

3.2.1 Actor-critic setup

The actor-critic setup is accurately represented by MDP, refer to subsection 2.3.1 for elaboration. The MDP considers that the current state of the environment represents its temporal transitions in the past. The set of states of the MDP ($s \in S$) are mapped as measured variables of the QT. The set of actions ($a \in A$) represent the manipulations by the policy ($\pi(a \mid s)$). The set of instantaneous rewards ($r \in R$) represent the quality of an action. This also acts as the handle to address constraints. The learning algorithm optimizes the actor $\pi(a \mid s)$ to maximize the discounted returns predicted by the critic $Q_\pi(s, a)$. The control systems analogy of the data flow in an actor-critic setup is depicted in figure 3.1.

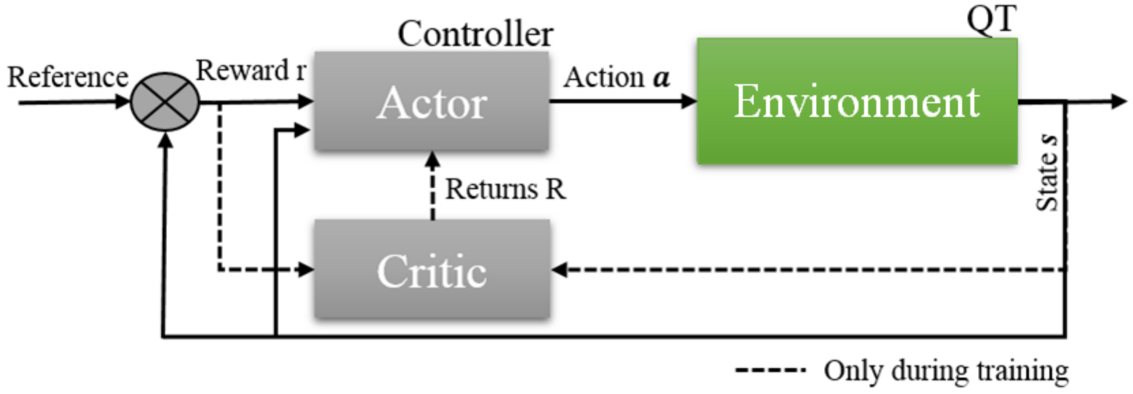


Figure 3.1: Actor-critic analogy to feedback control

As previously motivated (section 2.3), neural networks are often chosen as function approximators for the actor and the critic. The output of each node p_i in a neural network layer l with n nodes, amounts to a regression over the inputs $p_i^{t-1} : i \in 1 : n$ to that layer as shown in equation 3.1. The weights and biases (w_i, b_i) of each node make up the trainable parameters (θ) of the corresponding neural network. The neural network can hence be envisioned as a sum of nonlinear regressions.

$$p_i^{t-1} = f(w_i^l . p^{t-1} + b_i) \tag{3.1}$$

The actor's policy is modelled as a neural network with parameters $\theta \in \mathbf{R}^d$ where d represents the dimension of the network. Similarly, the critic (also called the action-value function Q_π) is also represented by a neural network with parameters $\theta_Q \in \mathbf{R}^{d_1}$ where d_1 represents the dimensions of this network. It provides the estimate of the policy returns E_π from taking actions according to the policy for a given state as shown in equation 3.2. The objective function (equation 3.3) maximizes the returns (discounted long-term rewards), calculated as given in equation 3.4 where β represents the discount factor (relative importance given to future rewards). In general, the action-value function predicts the discounted returns as represented in equation 3.5.

$$Q_\pi(s, a) = E_\pi(R_t \mid s_t, a_t) \tag{3.2}$$

$$\max_{\theta} J(\theta) = E(R_t \mid \pi_\theta) = E_\pi[\sum_k \beta^{k-1} r_{t+k} \mid s, a] \tag{3.3}$$

$$R_t = \sum_k \beta^{k-1} r_{t+k} \tag{3.4}$$

$$\hat{R}_t = Q_\pi(s_t, a_t) \tag{3.5}$$

The actor-critic scheme builds upon the policy gradient scheme that uses the REINFORCE algorithm [13] as mentioned in chapter 2.

3.2.2 Advantage actor-critic

In a Monte-Carlo setup, an improved means of exploration of the state/action space is acquired. The critic network, predicting the returns is optimized based on equation 3.6 where E_Q represents the action-value estimate of the total returns starting from state s and taking actions a from following the policy π.

$$\nabla_{\theta_Q} J(\theta_Q) = E_Q[\sum_{t=0}^{N} \nabla_{\theta_Q} \mid\mid R_t - \hat{R}_t \mid\mid^2] \tag{3.6}$$

where N represents the episodic length.

Further to this, the strategy unfolds into a two-step sequential optimization. The first step during each episodic update targets the critic's prediction error. The actual returns are calculated using the data from experience replay (equation 3.4). The temporal difference representing the prediction error between the actual returns (equation 3.4) and the current critic network's predictions (equation 3.5) is termed the advantage function $A(s, a)$. During learning, the advantage function can greatly stabilize the variance in the network gradients. The second step involves optimizing the actor network, based on minimizing the objective provided in equation 3.7. It is considered to comply with Bellman optimality [13], [31].

$$\nabla_\theta J(\theta) = E_\pi[\sum_{t=0}^{N} \nabla_\theta \log \pi(a_t \mid s_t, \theta)[\sum_{t=0}^{N} (R_t - \hat{R}_t)] \tag{3.7}$$

Using a stochastic gradient descent algorithm, (RMSPROP optimizer. TensorFlow), both the actor and critic are optimized. The two-step optimization is performed after a predetermined number of interactions of the actor with the environment. Both networks approach the optimal approximation irrespective of the initial weights in the actor and critic's networks. This sequence is depicted in figure 3.2 [13], where the optimal target policy (π^*) and the optimal target action-value function (Q^*_π) are obtained. The accumulated rewards from each episode are utilized to determine the status

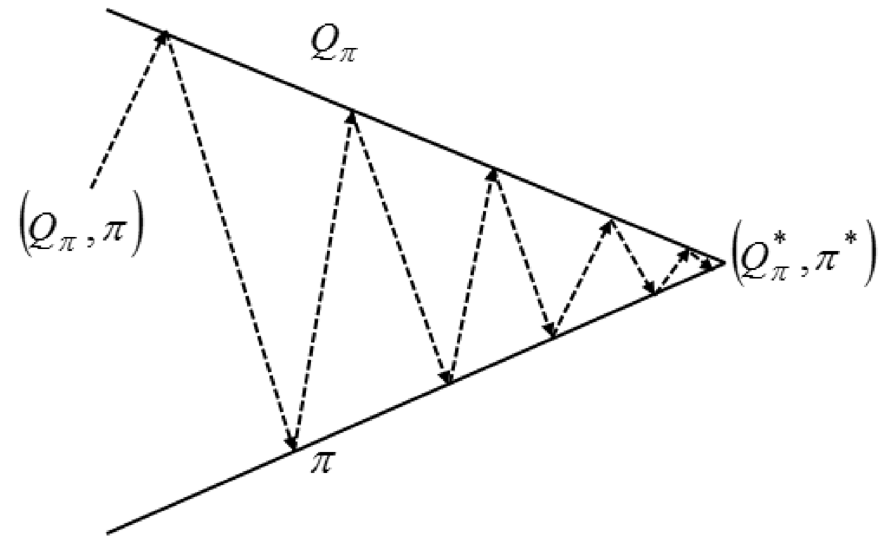


Figure 3.2: Optimal policy from random initialization

of convergence. Convergence, in the context of the action-value function, indicates the optimal prediction of the discounted returns (R_t), whereas optimal action selection for a given state vector is that of the actor's. The trajectories indicate a diminishing gradient to their respective network weights, whilst staying physically separate even after convergence.

3.2.3 Asynchronous advantage actor-critic

The coverage of a spatial range of state/action values and their combinations (in multivariate case) is termed as exploration. Exploration in a stable state space plays a key role in RL based control design for process control applications. Coverage of the state/action space becomes essential in providing a truly generalized solution for nonlinear systems. Asynchronous versions of the actor-critic schemes not only target maximal exploration but also redistribute the learning between multiple workers [1]. This eventually improves convergence in the premise of stability [16]. Parallel learning based off-policy schemes have been developed to address exploration and aid in asynchronous learning. Massively distributed architectures [36] to simple lightweight frameworks utilizing just the central processing unit (CPU) cores [1] have been proposed for parallel learning. In this study, the latter is utilized to design a control-oriented continuous space A3C scheme.

The A3C architecture for CPU based computation is provided in figure 3.3. A package constituting of an instance of the actor, advantage-based critic and the environment (in this case, encompassing the QT model and the rewarding mechanism) is termed a worker. One QT model is instantiated for each worker. Each worker individually interacts with its exclusive QT model whose data (based on the local policy interaction) is utilized in training the target network. A global actor-critic instance and multiple workers constitute the parallel learning infrastructure. Each worker's actor and critic neural networks are randomly initialized. In an episodic context, worker interactions are limited to the experiment duration.

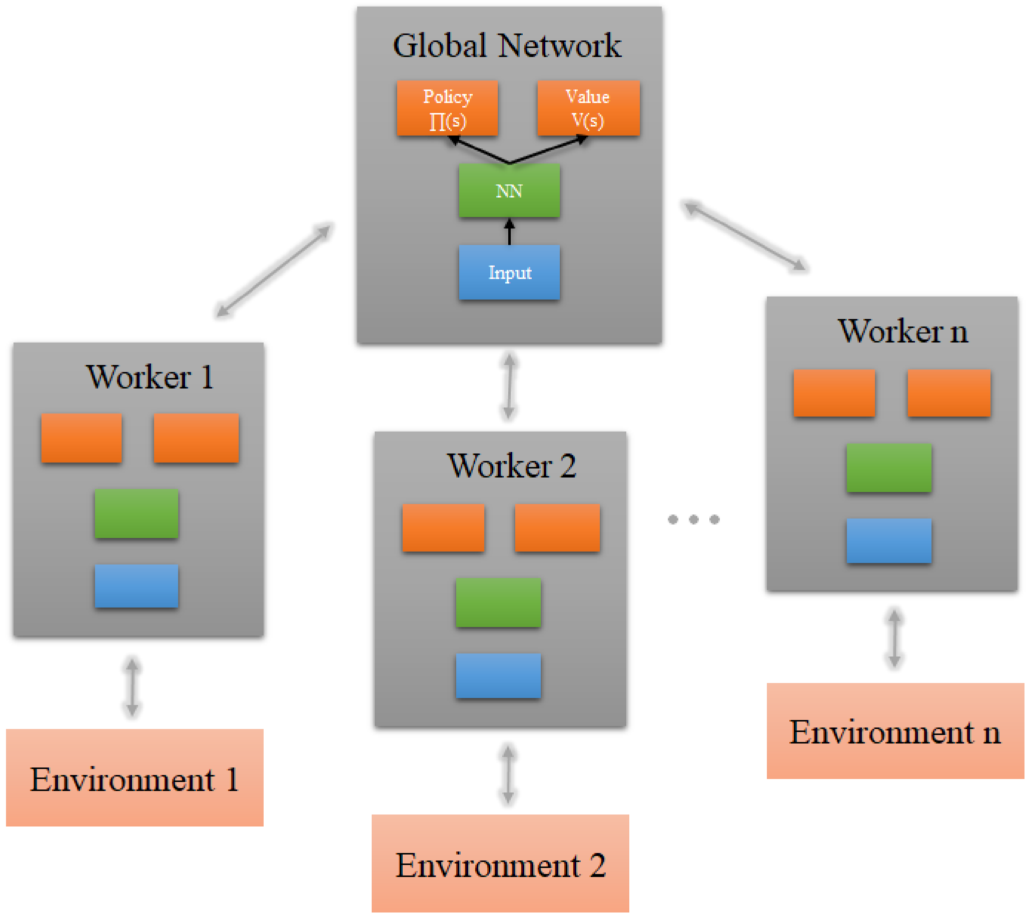


Figure 3.3: Asynchronous advantage actor-critic setup

During each sample time, the data (states, actions, and rewards) are stored in the experience replay buffer. A predetermined buffer length (in this report, 100 samples), once full, will be used to update the target network. This scheme ensures capturing actions that may result in higher rewards or penalties, which is not the case if the entire experiment is run before updating the target network. This episodic learning scheme is termed the temporal difference approach in RL. At the end of each update, the global actor-critic weights are pushed into the worker initiating the update.

Since workers use parallel computing, the updates on the global network are asynchronous, leading to learning from multiple workers' interactions. This improves exploration, which in turn can aid in convergence to an optimal policy. The episode is run its entire duration while intermediate updates are carried out as explained. Since each worker can be designed to work under varying conditions (setpoint sequence, measurement noise, etc.) the uniqueness of each worker network is preserved throughout the learning.

3.3 Quadruple tank process

The QT is an interacting nonlinear MIMO process. As illustrated in figure 3.4, the process constitutes of two interacting loops that impact the controlled variables (h1,h2), the levels in the bottom two tanks. The manipulated variables are two pump voltages (v_1, v_2). Each pump feeds to the bottom tank and the opposite top tank. Non-minimum phase transience is considered by controlling the flow ratio in both pumps $(\gamma_1 + \gamma_2 < 1)$ setting the manual valves V_1 and V_2.

The system of differential equations governing the transience in the levels of each tank is provided in equations 3.8 through equation 3.11. The model parameters involved are provided in Table 1 [32].

$$\frac{dh_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_3}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1 \tag{3.8}$$

$$\frac{dh_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_4}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2 \tag{3.9}$$

$$\frac{dh_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2 \tag{3.10}$$

$$\frac{dh_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1 \tag{3.11}$$

3.4 Results & discussion

In this study, numerical simulations were performed to obtain the reported results. The first principles model of the QT process with a sample time of 5 seconds was simulated and then controlled by RL. The duration of each episode in the experiments discussed was 2 hours and 30 minutes. The

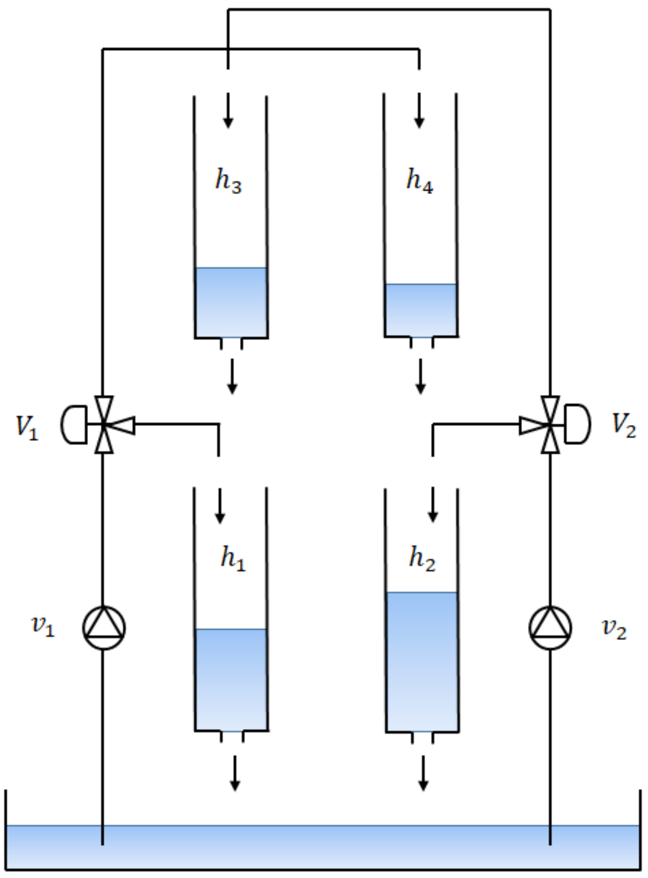


Figure 3.4: Quadruple tank process

Table 3.1: Quadruple tank model parameters

Parameter [unit]	**Description**	**Value**
$A_1, A_3 [cm^2]$	Tank surface area	28
$A_2, A_4 [cm^2]$	Tank surface area	32
$a_1, a_3 [cm^2]$	Outflow surface area	0.071
$a_2, a_4 [cm^2]$	Outflow surface area	0.057
$k_c [V/cm]$	Conversion constant	1
$g [cm^2/s^2]$	Acceleration due to gravity	981
$h_1^0 [cm]$	Steady-state level - Tank 1	10.43
$h_2^0 [cm]$	Steady-state level - Tank 2	15.98
$h_3^0 [cm]$	Steady-state level - Tank 3	6.6
$h_4^0 [cm]$	Steady-state level - Tank 4	9.57
$v_1^0 [V]$	Control voltage - Pump 1	3.15
$v_2^0 [V]$	Control voltage - Pump 2	3.15
k_1	Conversion constant	3.14
k_2	Conversion constant	3.29
γ_1	Flow ratio - Valve V_1	0.35
γ_2	Flow ratio - Valve V_2	0.35

differential equations were solved using Euler's method. The non-minimum nature of the QT was engaged by choice of setpoints through the mentioned period of the episode.

3.4.1 Infrastructure

The actor in the RL agent utilized a feed-forward network as function approximator. It considered input states X_A (given in equation 3.12) that are fed into two hidden layers of 200 neurons each (sigmoid activation). The outputs from the hidden layers were fully connected to a 4-neuron output layer with the output Y_A (given in equation 3.13).

$$X_A = [h_1, h_2, h_3, h_4, h_{1_{sp}}, h_{2_{sp}}]^T \tag{3.12}$$

$$Y_A = [\mu_{v_1}, \sigma_{v_1}, \mu_{v_2}, \sigma_{v_2}]^T \tag{3.13}$$

A centralized stochastic actor that sampled the actions $(\Delta v_1, \Delta v_2)$ from a Gaussian distribution obtained from Y_A was designed (given in equation 3.14 and 3.15).

$$\Delta v_1 \sim N(\mu_{v_1}, \sigma_{v_1}) \tag{3.14}$$

$$\Delta v_2 \sim N(\mu_{v_2}, \sigma_{v_2}) \tag{3.15}$$

The actor's input state vector was augmented with the stochastic actions from the actor to form critic's input state vector X_C, given in equation 3.16. A single hidden layer feed-forward network with 100 neurons (sigmoid activation) fully connected to a single neuron scalar output ($\hat{R}_t$) constituted the critic. A contribution of this work is proposing a reward structure that is compliant with the process control objective function (presented in equation 3.17) of reducing deviation of the process variable y from the setpoint y_{SP}, and reducing changes to the control action (u), i.e. the manipulated variable. The instantaneous reward structure considered for the experiments is hence presented in equation 3.18.

$$X_C = [X_A^T, \Delta v_1, \Delta v_2] \tag{3.16}$$

$$\min \mid y - y_{SP} \mid^2 + \mid \Delta u \mid^2 \tag{3.17}$$

$$r_t = \sum_{i=1}^{2} (\mid h_{i_{sp}} - h_i \mid + \lambda \mid \Delta v_i \mid) \tag{3.18}$$

where λ is an aggression tuning factor determining the importance of each objective. It was kept at 0.25 for all experiments reported.

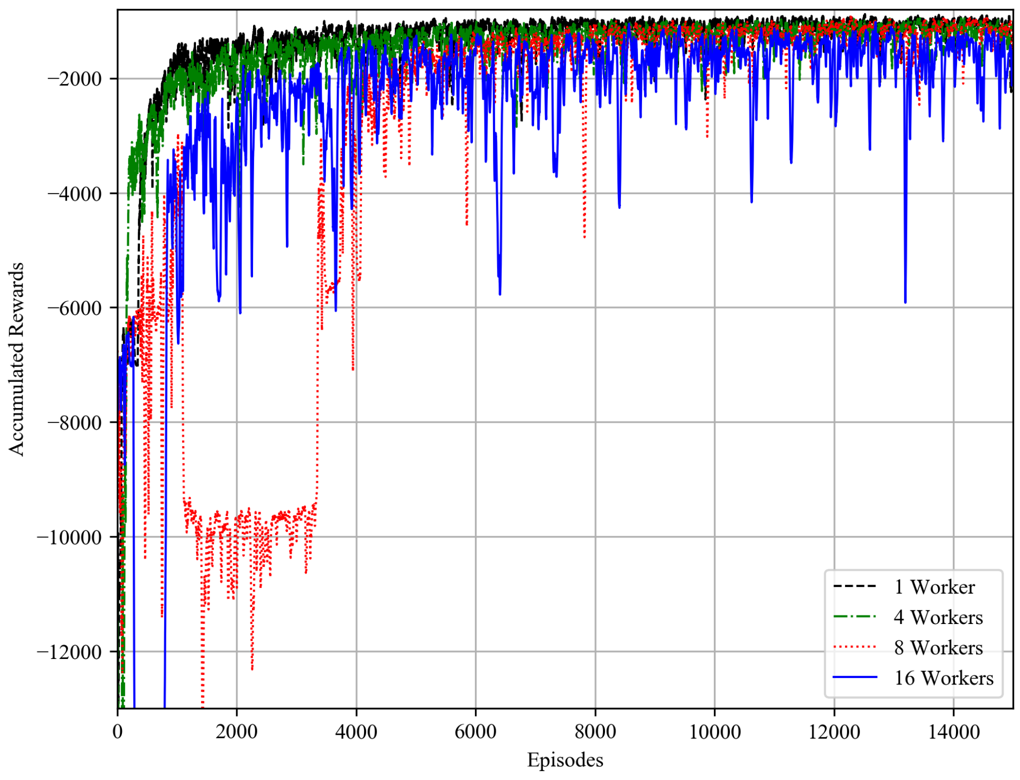


Figure 3.5: Parallel learning with varying worker cohorts

The software platform used constituted of TensorFlow version 1.9.0 used with Python 3.7.1 for both environment and A3C on Windows 10 64-bit operating system. A Lambda computer with an Intel i9-9820X processor with 20 threads was utilized for the numerical simulation.

3.4.2 Parallel learning and convergence

Multiple experiments were conducted with varying degrees of parallel learning (with multiple combination of worker cohorts). For this, 1, 4, 8, and 16 workers respectively were engaged in learning continuous servotracking of interacting levels in the QT. Each experiment constituted a Monte-Carlo approach with 15,000 episodes. Here, figure 3.5 illustrates the accumulated rewards obtained using varying degrees of parallel learning. The accumulated rewards from each episode are trended over the learning phase. The metrics to evaluate these trends are presented in table 3.2.

For each experiment, a convergence metric based on the number of episodes by which the accumulated rewards reach within 10% of its best-accumulated rewards per episode was considered. This is termed as the optimal range. Also, from each experiment, the variance in accumulated reward

Table 3.2: Parallel learning metrics

Cohorts	**Highest Reward**	**Convergence**	**Variance**
1	-878.9	5291	32341
4	-949.9	5394	37680
8	-917.8	8349	54854
16	-1016.9	8559	230400

after attaining optimal range was deduced. The number of episodes considered for this was based on the slowest converging experiment to maintain uniformity.

3.4.3 State/action space exploration

From table 3.2, it is inferred that the convergence is slowed down by an increase in the worker cohorts whilst, the variance in accumulated rewards increases exponentially (also observed in figure 3.5. This is an indication of the exploration being promoted by an increased number of worker cohorts. However, this exploration trend continues, despite the accumulated reward approaching within 15% of the single worker experiment.

This is effected by the slower reduction in cross-entropy between state and corresponding action taken by the actor. Spatial visualization of the state/action space exploration through the temporal learning process is depicted in figure 3.6. One of the worker cohort experiments is chosen to portray this exploration. At an interval of 2000 episodes, a single worker is chosen, whose policy at that instance is saved. The servotracking of levels of the QT is shown alongside the corresponding manipulations in the pump flowrates in figure 3.7. It can be observed that the setpoints were reached with lesser effort as the training continued as indicated by the spatial visualization of the temporal learning process. With the progression in learning, it is observed that the actions turn near deterministic.

3.4.4 Servotracking of levels in QT

Using the optimal policy (with best accumulated rewards) from each experiment, the control quality is assessed. The metrics such as mean square of error (MSE), integral of absolute error (IAE), variance in control (VC) are provided in table 3.3 for tank 1 and table 3.4 for tank 2. where input-output pairing for VC consideration is tank 1 with pump 2, and tank 2 with pump 1.
It is observed that the variance in the metrics amongst varying cohort experiments is very minimal. Also, the visualization of servotracking provided in figure 3.7, indicates a near-identical control

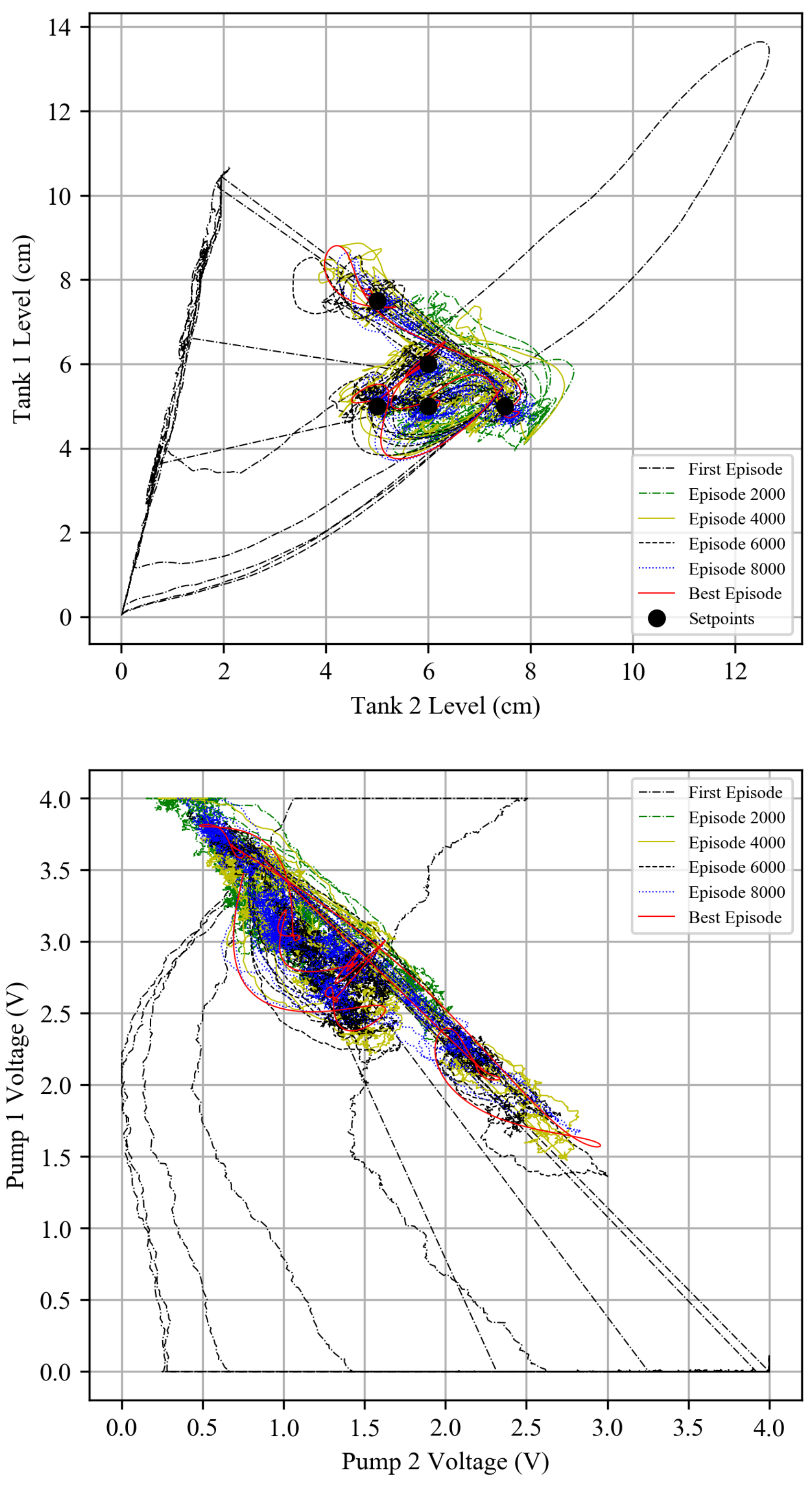
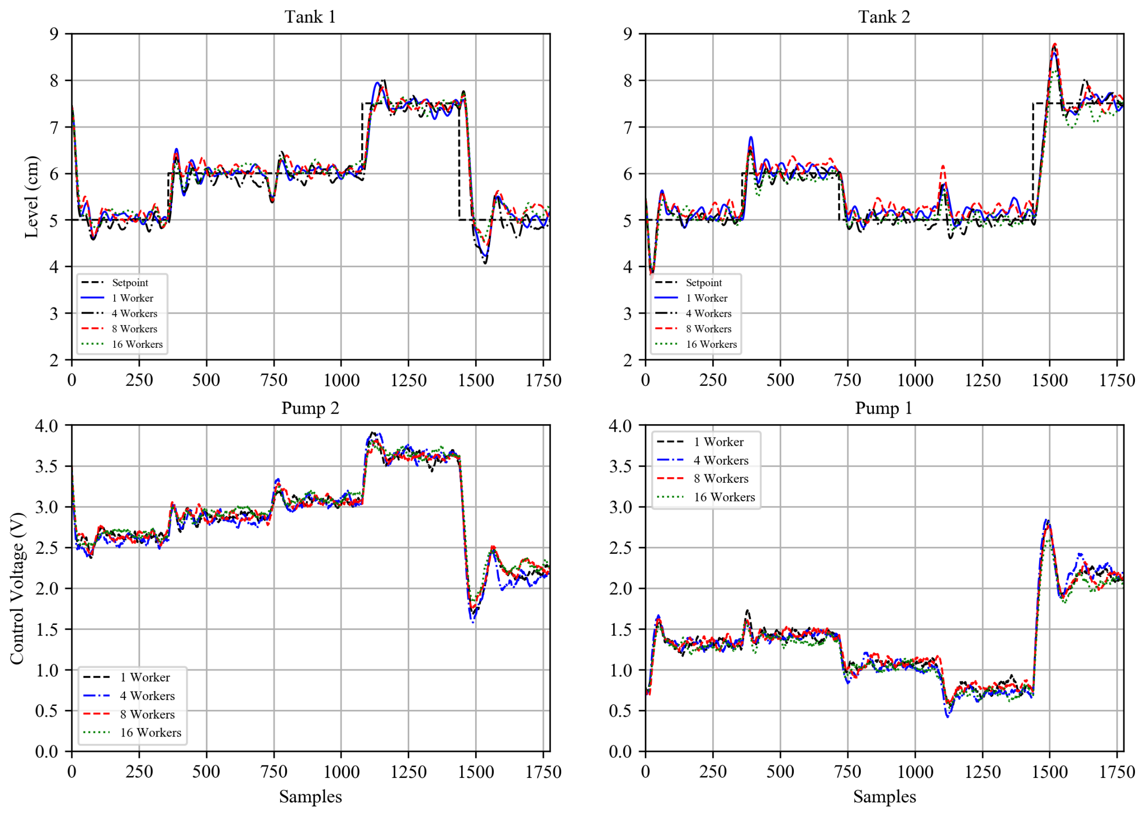


Figure 3.6: State/action exploration



[t]

Figure 3.7: Servotracking of levels through multivariable control

Table 3.3: Control performance on tank 1 level

Workers	**MSE**	**IAE**	**VC**
1	0.2133	326.4506	0.2413
4	0.2280	391.3606	0.2849
8	0.2091	389.3436	0.2207
16	0.2175	362.5388	0.2268

Table 3.4: Control performance on tank 2 level

Workers	**MSE**	**IAE**	**VC**
1	0.1485	388.8634	0.2343
4	0.1558	319.5568	0.2847
8	0.1792	504.7661	0.2209
16	0.1435	351.7604	0.2160

configuration, generalized by the actors. Such observations, ascertain that the learning process leads to a near-optimal policy, irrespective of the degree of parallel learning. If otherwise, the policies would have had varying steady-state combinations of control signals.

3.5 Conclusions

A reinforcement learning (RL) agent was implemented for servotracking of a multivariate nonlinear QT process using the A3C algorithm. A multitude of worker cohort combinations was utilized in the learning process to assess the impact of the degree of parallel learning both on control quality and convergence to an optimal policy. Action/state space exploration was increased by a higher degree of parallel learning, resulting in delayed convergence to the optimal policy. Amidst approaching optimal policy, a higher state-action cross-entropy was observed with a higher number of workers. Control related metrics (MSE, IAE, and VC) suggest that the policies obtained were near-identical in performance with minimal variations. With the higher degree of exploration, policies for nonlinear systems can be generalized more accurately. Therefore, A3C can address the continuous space convergence effectively, enabling its adaptation for continuous control in chemical processes.

Chapter 4

Hierarchical Constrained Reinforcement Learning for Optimization of Bitumen Recovery Rate in a Primary Separation Vessel

The contributions of this chapter have been submitted to:

Referred Journal Publication:

Hareem Shafi, Kirubakaran Velswamy, Biao Huang, *Hierarchical Constrained Reinforcement Learning for Optimization of Bitumen Recovery Rate in a Primary Separation Vessel*, IEEE Transactions on Control Systems Technology [submitted].

Higher-level optimization of the Primary Separation Vessel (PSV) is concerned with improving the bitumen recovery rate. It is achieved by a lower-level control of the froth-middlings interface level. This is necessitated by the nonlinear nature of the process and the unpredictability of the ore composition of the slurry fed into the PSV. It must occur concurrently with regulation of the tailings density. This work reports a two-level hierarchical control structure designed using the RL scheme. The lower level is concerned with servotracking and regulation of the interface level against variances in ore quality by manipulating middlings flow rate is designed. At the higher level, to optimize the bitumen recovery rate, a supervisory interface level setpoint control is implemented.

To prevent sanding, a tailings density regulation using tailings withdrawal rate is proposed. For each case, an asynchronous advantage actor-critic (A3C) based agent is chosen to interact with a high-fidelity PSV model to learn the near-optimal control strategy through episodic interactions. Function approximators are used to tackle the infinite continuous state-action space-oriented optimization of this policy. Each of the three control loops is sequentially learned. In the interface level control loop, behavioral cloning based two-phase learning scheme to promote stable state space exploration is proposed. Based on simulation results, the behavioral cloning scheme ensured improved convergence to the near-optimal policy. The proposed hierarchical structure successfully demonstrates improved bitumen recovery rate by manipulating the interface level while preventing sanding, demonstrating the feasibility of such approaches to chemical processes.

4.1 Introduction

One-fifth of the total oil sands production in Canada is based on open-pit ore extraction. It starts with the mining phase where oil sands ore is shoveled out of the ground. The mined ore is then crushed and transported for the extraction phase. For extraction, heat and chemicals are added to the crushed ore to form a slurry mixture. This mixture is then sent to a gravity separation vessel known as the Primary Separation Vessel (PSV). Once the slurry is fed into the PSV through a feed stream, it forms three distinct layers due to the difference in their densities. These layers are known as the froth layer, the middlings layer, and the tailings layer. The process described is illustrated by means of a block diagram in figure 4.1.

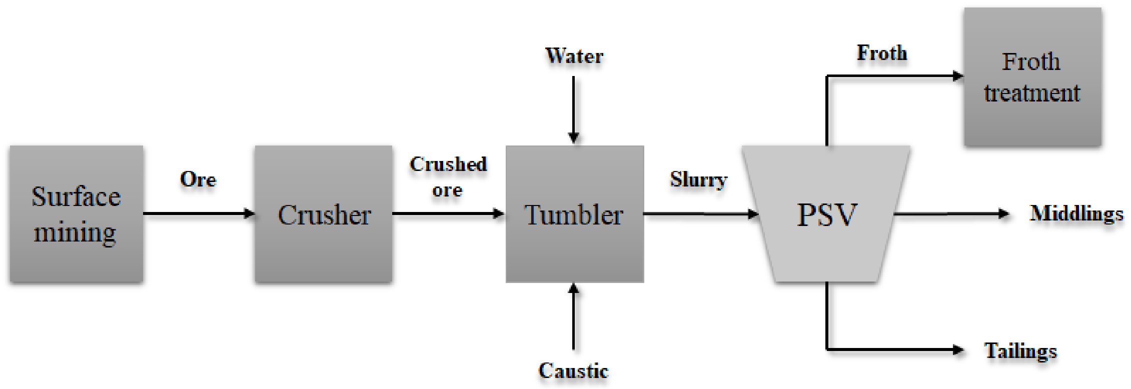


Figure 4.1: Block diagram of the ore handling process

The froth layer that contains mostly bitumen (around 60% bitumen, 10% solids, and 30% water), floats to form the top layer and overflows to upgrading for further treatment. The heaviest particles precipitate at the bottom forming the tailings layer which is withdrawn for further processing before being disposed into a tailings pond. The remaining composition contains mostly water (59% water, 24% bitumen, and 17% solids) and forms the middlings layer in between the froth and the tailings layer. A middlings side stream is pumped from the middle of the vessel to a secondary separation phase for further treatment to recover the leftover bitumen that does not float to the top froth layer.

A highly efficient PSV achieves maximum recovery of bitumen relative to water and solid particles. It reduces the additional processing load on the downstream separation processes. This is owing to the fact that high-quality froth obtained from primary separation requires less processing and energy to remove the remaining solids and water. Hence, the PSV plays a major role in the gravity-based separation of bitumen from oil sands. Optimal recovery of bitumen through the froth, and overall efficiency of the extraction process, plays a crucial role in the economic and environmental impact that the oil sands industry creates [37]. Hence, the optimal operation of PSV can help achieve environmental and financial targets.

Hence, with the objective of improving the bitumen recovery rate, the first process variable to be considered is the froth-middlings interface level which directly affects the bitumen recovery rate and has to be regulated within an operational range. Otherwise, it can result in either reducing the quality of the froth being recovered [38], or losing bitumen to the tailings layer causing further contamination of the tailings pond [39]. Besides improving the bitumen recovery, it is crucial to control the density of the tailings layer which is to be maintained below a certain operational value to avoid excess sand bed build-up in the vessel bottom [40]. This can lead to complete pipeline plugging, also referred to as the 'sanding' phenomenon, particularly if it is associated with a lower tailings withdrawal flowrate.

The theoretical and experimental work reported in [39] provides the foundation for modeling the PSV. This forms the basis for evaluation of the separation performance of PSV in general. More specifically, control-oriented as well as operating range oriented studies have also been reported. A typical control problem was developed in [40] with classical multi-loop Proportional Integral (PI) controllers for interface level and tailings density. However, improving the bitumen recovery was not considered in the control objective.

In [41], an improved economic model predictive controller (MPC) scheme was applied to a PSV model. The objective was to maximize the overall recovery rate of bitumen without, however, considering the sanding problem. In [42], the interface level was being controlled by a PI controller. Tests were run for different ore grades to evaluate its impact on the middlings density, froth quality, the bitumen flow, and the bitumen recovery rate. It has been suggested to use model-based predictive control strategy for all the controlled variables by simultaneously adjusting all manipulated variables. In [37], optimal input trajectories were calculated off-line for different known ore grade transitions. The actual implementation of these optimal trajectories requires the PSV operators to have prior knowledge of the ore quality which limits the applicability of such open-loop control only to known ore grades.

Based on the existing literature, it is concluded that factors such as ore grade, feed flow rate, assumed particle size distribution, and other uncertainties related to modeling assumptions are uncontrollable. They constitute sources of uncertainties and disturbances. Consequently, it impacts the density of the tailings and the middlings layer, resulting in reduced bitumen recovery, and affects the separation performance of the PSV in general. None of the existing works actually have taken into consideration the impact of the unpredicted nature of all different disturbances on the separation performance. Therefore, we cope, with such challenges by using a model-free approach like RL in order to provide a generalized solution to such a complex problem. Therefore, we formulate our control problem accordingly to maximize the bitumen recovery rate through control of the froth-middlings interface level while regulating the density of the tailings layer to prevent sanding in the tailings layer.

Reinforcement learning (RL) has gained popularity as a control scheme in recent times due to its ability to learn through trial and error. RL algorithms learn by interacting directly with the environment to sample the optimal actions in order to achieve a specified goal [13]. In the RL context, the action selection is carried out by the agent, and the process with which the agent interacts is the environment. The framework of their interaction is a Markov decision process (MDP). In an MDP, there are states that the environment can assume, actions that the agent can take, and the reward that is obtained by virtue of taking a particular action in the current observed states. The agent's state to action mapping vector is known as the policy while the cumulative long-term rewards are called the value/action-value function.

RL borrows its formal structure from optimal control where the objective is to design a controller to minimize an objective function of a dynamical system's behavior over time [14]. The approach towards solving this problem considers the state to generate actions and then the value function is used to improve the choice of actions for the dynamical system. This is considered to satisfy Bellman optimality. It is from here that the discrete stochastic version of the optimal control structure, MDP, hails from. Employing temporal-difference (TD) learning to find the optimal policy for a MDP in the 1980s resulted in the reinforcement learning structure that is now widely utilized [13].

Q-learning, which considers the action-value function (Q-function) in learning the policy was instrumental in the initial popularity of RL [18]. It was, however, limited by the curse of dimensionality. A solution to this problem was proposed in the form of neural network based function approximators to estimate the Q-function (DQN) for higher dimensional state spaces in RL problems [20]. This enabled control of continuous space environments. Continuous action space optimization was made possible with the introduction of the DDPG algorithm, which employed an actor-critic type model-free architecture for control of more than 20 simulated physics tasks [21].

Actor-critic algorithms constitute of an actor which represents the policy and a critic that represents the action-value function. They employ a Monte-Carlo kind of scheme where learning occurs over experience. Experience is gained from multiple repeated episodes to regress an approximation for returns in the form of the action-value function in actor-critic methods [19]. This allows the actor-critic algorithms to be model-free [13]. In such schemes, the local policy interacts with the environment, from which the rewards and consequently the returns are calculated. Based on the returns, the local policy pulls the global policy to optimize the returns. Every update to the actor is preceded by an update to the critic. The update to the critic is based on minimizing a mean square error (MSE) criterion in predicting the returns, meant to improve the estimate from the action-value function. Using this sequential approach to learning in actor-critic, convergence to a near-optimal policy can be guaranteed [13].

Continuous space control was demonstrated using DDPG on a variety of 3D tasks [43], a combination of DQN and DDPG for mobile robot control [44], stochastic value gradients (SVG) on several physics tasks [45], and an asynchronous variant of actor-critic on Atari domain [1]. Due to these successful RL implementations in various domains, it makes sense to extend it to process control applications. Drawing analogy between the two, the goal of the RL agent in the process control domain would be to keep a multivariable process within safe operational limits while maintaining it at the setpoint despite process disturbances and measurement noise [16].

The ability of RL algorithms to self-learn from direct interaction with the process data make them suitable for use with nonlinear processes where deriving the process model might not be possible or accurate [46]. Due to their self-learning nature, they also have the ability to adapt to process disturbances and shifts in operating conditions. Previously, successful control of thermostat scheduling for office space in a discrete action space setting has been reported [30]. Also, continuous space optimization using a policy gradient based approach has also been reported [29]. These schemes were based on on-policy proximal actor-critic setup.

In recent work by the authors, servotracking control of a benchmark quadruple tank process has been successfully demonstrated [47]. This chapter builds on that work to propose an asynchronous advantage actor-critic (A3C) based solution for the PSV as motivated previously.

The contributions of this chapter are listed as follows: A hierarchical structure is proposed for reinforcement learning with objective towards both economic optimization and optimal control. The proposed strategy is then applied for improving the bitumen recovery rate of PSV through froth-middlings interface level tracking while regulating the tailings density to prevent sanding. As a novel approach for interface tracking, a semi-supervised scheme based on behavioural cloning is proposed and employed during training for safe exploration of the action space. The multiloop un-coordinated lower level RL agents overlook froth-middlings interface level control and tailings density regulation, while the supervisory RL agent looks at bitumen recovery rate optimization. To the best of our knowledge, this is also the first study and application of reinforcement learning in the oil sands extraction process.

The rest of the chapter is arranged as follows: Section 4.2 details the high fidelity PSV model used, Section 4.3 discusses the multi-loop RL control architecture and the experimental setup, Section 4.4 shares the results and discussions, and finally Section 4.5 highlights the main conclusions and sets directions for future work.

4.2 PSV process model

4.2.1 Process description

Mass balance with gravity separation principles used in the PSV model in this work are all based on the work in [48] and its references. The gravity separation principles employed are taken from [37]. The following main assumptions are considered in the model development. The materials' species present in the PSV are bitumen, solids, and water (labelled by the subscript j that takes b, s, w respectively). They are considered to be present in three constant sizes: small, medium, and large. The bitumen and coarse solid particles are assumed to be spherical; whereas, the fine solids follow platelets' shape. The density of species (bitumen, water, and solids particles) are all assumed to be constant and the viscosity of the middlings layer is assumed to be that of water.

Each layer is assumed to be perfectly mixed, contains continuous medium, and modelled using mass balance principles with interactions between layers through froth-middlings and middlings-tailings interfaces. This interaction is characterized by considering particles' movement between layers to follow steady-state settling relationships (Stokes law). It will be briefly revisited in the forthcoming subsections in combination with hindered settling models for suspension of particles following the reference [49]. The froth-middlings interface is considered to be mobile and particles can move back and forth through it. While, the middlings-tailings interface is static and particles only move in a downward direction. The downwards direction is considered to be the positive direction of particles movement along with one-dimensional assumed flow.

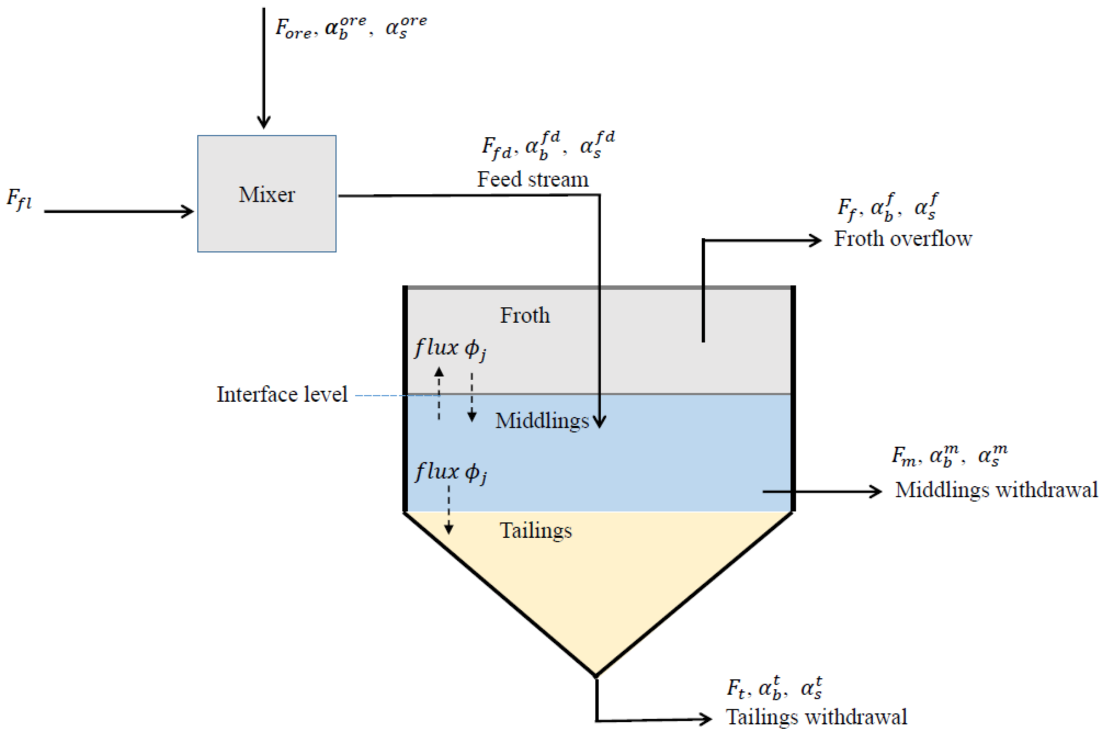

F_{ore}, α_b^{ore}, α_s^{ore}
F_{fl}
Mixer
F_{fd}, α_b^{fd}, α_s^{fd}
Feed stream
F_f, α_b^f, α_s^f
Froth overflow
Froth
$flux\ \phi_j$
Interface level
Middlings
$flux\ \phi_j$
F_m, α_b^m, α_s^m
Middlings withdrawal
Tailings
F_t, α_b^t, α_s^t
Tailings withdrawal

Figure 4.2: PSV schematic

In the mass balance equations presented hereafter, no material generation is assumed and can be expressed as expounded in the following subsections. The notation used is presented in table 4.1.

Table 4.1: Primary separation vessel notation

Parameter	**Description [unit]**
$i \in f, m, t$	froth, middlings, and tailings [no unit]
$j \in b, s, w$	bitumen, sand, and water [no unit]
$k \in 1, 2, 3$	small, medium, and large particle size [no unit]
I_{F-M}	froth-middlings interface level [m]
F_{fd}	feed flow rate [m^3s^{-1}]
F_{fl}	flood water flow rate [m^3s^{-1}]
F_f	froth overflow [m^3s^{-1}]
F_m	middlings withdrawal flowrate [m^3s^{-1}]
$\overline{F_m}$	nominal middlings withdrawal flowrate [m^3s^{-1}]
F_t	tailings withdrawal flowrate [m^3s^{-1}]
$\overline{F_t}$	nominal tailings withdrawal flowrate [m^3s^{-1}]
V_i	volume of i^{th} layer [m^3]
$V_{i_{SP}}$	volume setpoint of i^{th} layer [m^3]
α_j^i	volume fraction of species j in layer i [no unit]
ϕ_j	flux of species j [m^3s^{-1}]
A_{vessel}	vessel cross-sectional area [m^2]
v_I	froth-middlings interface velocity [ms^{-1}]
v_j^i	settling velocity of species j in layer i [ms^{-1}]
ρ_j	density of species j [kgm^{-3}]
ρ_i	density of layer i [kgm^{-3}]
d_j^k	particle diameter size k of species j [m]
g	gravitational constant [m^2s^{-1}]
e_t	error term [m]
Θ, ϵ	settling velocity correction terms [no unit]
η	dynamic viscosity [$kgms^{-1}$]

4.2.2 Froth layer

The volume of the froth layer V_f is assumed to be a function of the interface velocity v_I. This is because the top of the froth layer is assumed to be fixed and matches the top of the PSV. It is described by equation (4.1) where A_{vessel} represents the vessel cross-sectional area.

$$\frac{dV_f}{dt} = A_{vessel}v_I \tag{4.1}$$

As shown in figure 4.2, a species j's transport occurs as 1) a flux ϕ_j through the interface with the middlings (equation (4.3)) and 2) leaves the top of the PSV with a flow rate of F_f. Applying mass balance principles, the volumetric fraction of a species j in the froth layer (α_j^f) is described in equation (4.2).

$$\frac{d\alpha_j^f}{dt} = \frac{1}{V_f}(\phi_j - \alpha_j^f F_f - \alpha_j^f A_{vessel}v_I) \tag{4.2}$$

$$\phi_j = \begin{cases} \alpha_j^m A_{vessel}(v_I - v_j^m), & v_I > v_j^m \\ \alpha_j^f A_{vessel}(v_I - v_j^m), & v_I \leq v_j^m \end{cases} \tag{4.3}$$

where $j \in b, s$. α_j^m is the volumetric fraction of a species j in the middlings layer entering the froth layer. This occurs when the interface velocity v_I is greater than the settling velocity v_j^m of species j in the middlings layer.

As indicated by [50] and the reference within, the settling velocity v_j^m is calculated by equation (4.4). This equation corrects the free settling velocity v_j^{free} by Concha's correlation [39]. This correction is considered in order to account for the suspension resulting from the presence of other particles in a layer, so the settling of a particle is hindered as indicated in equation (4.4):

$$v_j^m = v_j^{free} \frac{(1 - 1.45 \sum \alpha^{particles})^{1.83}}{1 + 0.75^{\frac{1}{3}}} \tag{4.4}$$

The free settling velocity itself v_j^{free} was developed by Swanson [49] and is based on Stokes's equations for free-settling as shown in equation (4.5):

$$v_j^{free} = \frac{\frac{4}{3} g d^2 (\rho_j - \rho_i)}{\theta_j (2d^{\frac{3}{2}} (\frac{g \rho_j \rho_i}{3})^{\frac{1}{2}} + \sqrt{48} \epsilon_j \eta)} \tag{4.5}$$

where the shape factors of a species $j \in b, s$, is represented by the parameters θ_j and ϵ_j. g is the gravitational constant and η is the viscosity of water, and d refer to the particle diameter of a species. Three particles' sizes are considered for bitumen and three for sand particles as indicated previously. The shape factor is assumed to be spherical for bitumen and for coarse solid particles and assumed to be platelets for the fine particles (clays).

The suspension in a layer is assumed to be uniform and its density is calculated as the weighted summation of species' densities in it as represented in equation (4.6):

$$\rho_i = \rho_w \alpha_w^i + \rho_b \alpha_b^i + \rho_s \alpha_s^i \tag{4.6}$$

where $i \in f, m, t$ denotes froth, middlings, and tailings respectively. The subscripts w, b and s indicate the species, namely water, bitumen, and sands respectively. ρ_j indicates the density of a species j either bitumen or sand particle.

The interface velocity v_I is modelled as the Wallis shockwave equation (equation (4.7)) for a first order approximation as follows:

$$v_I = \frac{\sum_{k=1}^{3} \alpha_{bk}^m v_{bk}^m - \sum_{k=1}^{3} \alpha_{bk}^f v_{bk}^f}{\sum_{k=1}^{3} \alpha_{bk}^m - \sum_{k=1}^{3} \alpha_{bk}^f} \tag{4.7}$$

where k is the index of particle size (3 sizes were considered) and again α_j^i is the volume fraction of species j in layer i.

4.2.3 Middlings layer

Similar to the volume of the froth layer, the middlings layer volume V_m is assumed to be only a function of the interface velocity v_I as the middlings-tailings interface is stationary and only the froth-middlings interface is mobile. Thus, the middlings layer is represented as in equation (4.8).

$$\frac{dV_m}{dt} = A_{vessel} v_I \tag{4.8}$$

As shown in figure 4.2, a species j's transport in the middlings layer occurs as a 1) flux ϕ_j through the interface with both, the froth layer and the tailings layer as indicated in equation (4.10), 2) feed injected slurry with flowrate F_{fd}, and 3) as a withdrawal that leaves the middlings layer with the withdrawal flowrate F_m. Consequently, using mass balance principles, the volumetric fraction of a species $j \in b, s$ in the middlings layer (α_j^m) is described as given in equation (4.9).

$$\frac{d\alpha_j^m}{dt} = \frac{1}{V_m}(\alpha_j^{fd} F_{fd} - \alpha_j^m F_m - \alpha_j^m A_{vessel} v_j^t + \alpha_j^m A_{vessel} v_I + \phi_j) \tag{4.9}$$

$$\phi_j = \begin{cases} -\alpha_j^m A_{vessel}(v_I - v_j^m), & v_I > v_j^m \\ -\alpha_j^f A_{vessel}(v_I - v_j^m), & v_I \leq v_j^m \end{cases} \tag{4.10}$$

where α_j^{fd} is the volumetric fraction of species j in the feed stream, and v_j^t is the hindered settling velocity of a particle of species j in the tailings layer also calculated using equation (4.4).

4.2.4 Tailings layer

The volume of the tailings layer is constant as the middlings-tailings interface is considered stationary and this simplifies the model equations. As shown in figure 4.2, a species j transport occurs as a 1) flux ϕ_j through the middlings-tailings interface and 2) as a withdrawal that leaves with the withdrawal flowrate F_t from the bottom of the PSV. The volumetric fraction of species $j \in b, s$ in the tailings layer (α_j^t) is then described as given in equation (4.11).

$$\frac{d\alpha_j^t}{dt} = \frac{1}{V_t}(\alpha_j^m A_{vessel} v_j^t - F_t \alpha_j^t) \tag{4.11}$$

4.2.5 Feed equation

As shown in figure 4.2, with a flowrate of F_{ore}, ore is fed to a mixer of volume V_{mix} to be first mixed with flood water of flowrate F_{fl} before being fed into the PSV. Thus, the volumetric fraction of species j in the feed stream (α_j^{fd}) is described in equation (4.12).

$$\frac{d\alpha_j^{fd}}{dt} = \frac{1}{V_{mix}}(\alpha_j^{ore} F_{ore} - \alpha_j^{fd}(F_{ore} + F_{fl})) \tag{4.12}$$

The following flow rate balance is considered to calculate and constrain the overflow stream:

$$F_{fd} = F_{fl} + F_{ore}$$

$$F_f = F_{fd} - F_m - F_t$$

such that:

$$F_f \geq 0$$

4.2.6 Recovery rate

The efficacy of the PSV in extracting a bitumen rich froth directly affects the economic impact of the oil sands industry by determining the load on the downstream processes. This effectiveness is represented by the bitumen recovery rate RR. It depends on the bitumen content in the froth α_b^f and ore α_b^{fd} and the corresponding froth overflow rate F_f and ore flowrate F_{ore} as represented in equation (4.13).

$$RR = \frac{\sum \alpha_b^f F_f}{\sum \alpha_b^{ore} F_{ore}} \tag{4.13}$$

4.3 RL based control

4.3.1 Markov decision process

As previously motivated, the RL framework comprises of: a RL agent that is the learner in the process (analogous to the controller) and the environment (analogous to the plant including the rewarding mechanism). The agent interacts with the environment to optimize a certain facet of its behavior skewed by the designer's choice of reward. This framework is represented by a MDP that follows the Markov property [51]. It assumes that the present state of the environment is sufficient to make the optimal decision, i.e. it contains the relevant historical information. The MDP encapsulates the agent-environment interaction in discrete time steps within the finite time learning episode $t \in N$. The terminal state of an infinite horizon optimization RL setup is based on an episodic approach. In the following subsections, the projection of PSV's state space into the action space (middlings flow rate F_m, froth-middlings interface level setpoint $I_{F-M_{SP}}$, tailings flow rate F_t) based on the rewarding mechanism is described using this MDP structure.

The specific case of lower level froth-middlings interface level control is used as an example to understand MDP in this subsection. It is assumed that at each time instant t, there is a set of observable states $s_t \in S$ that the environment can assume, such as $s_t = [I_{F-M}, I_{F-M_{SP}}]$. There is also a set of actions $a_t \in A$ the agent can choose from, given the state observation s_t, to manipulate the F_m to track the interface level. This is done in accordance with its current policy $\pi(a_t \mid s_t)$. By virtue of the action a_t, the PSV transitions to a new state s_{t+1} and emits a scalar reward r_t associated with being in the new state and the action that had been taken, such as in equation 4.14). The rewards are accumulated over time by following the policy π. They are then corrected by a discount factor γ which helps to keep the returns bounded. This determines the relative importance of future rewards and is represented in the form of returns R_t as given in equation 4.15. Returns in this context are a direct feedback on the agent's performance at each state with reference to the terminal goal. They are calculated by considering the deviation from the setpoint over time. Their estimation, given only the states, is also known as the value function $V(s)$ as shown in equation 4.16. If the action taken (the flowrates chosen) is also considered in obtaining the expectation of returns, they constitute the action-value function $Q(s,a)$ given in equation 4.17. MDP setup for the lower loop corresponding to interface level control in the hierarchical architecture is illustrated in figure 4.3.

$$r_t = - \mid I_{F-M}(t) - I_{F-M_{SP}}(t) \mid \tag{4.14}$$

$$R_t = \sum_{k=0}^{n} \gamma^k (- \mid I_{F-M}(t+k) - I_{F-M_{SP}}(t+k) \mid) \tag{4.15}$$

$$V(s) = E_\pi(R_t \mid s) \tag{4.16}$$

$$Q(s,a) = E_\pi(R_t \mid s,a) \tag{4.17}$$

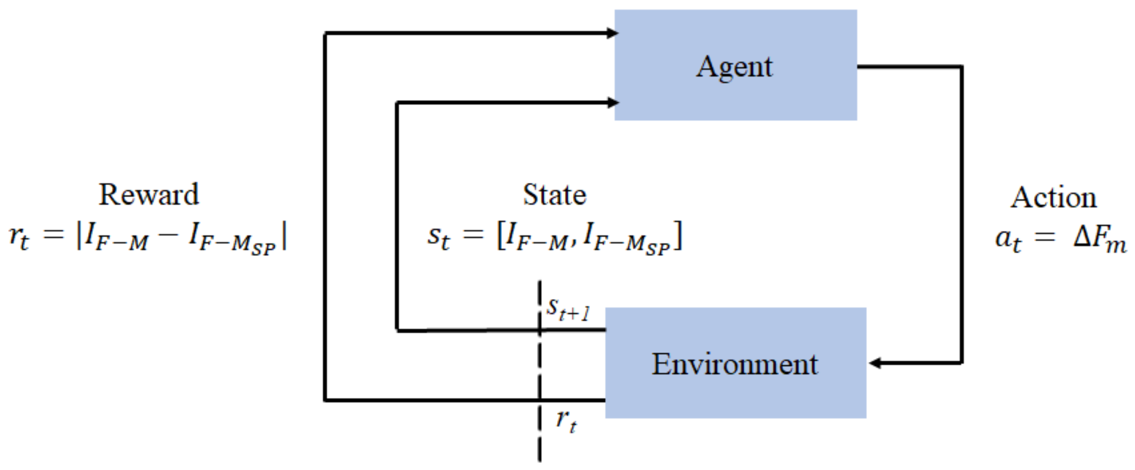


Figure 4.3: Markov decision process representation for lower level interface level tracking

4.3.2 Actor-critic

Actor-Critic combines the benefits of DQN and DDPG [16] to allow the state and action sets in the MDP context to transcend from discrete to continuous state/action space. Neural networks are employed as function approximators to implement the policy $\pi_{\theta_A}(a_t \mid s_t)$ represented by the actor, and the action-value function $Q_{\theta_C}(s_t, a_t)$ represented by the critic (which is a monotonic function), where θ_A, θ_C represent the neural network parameters for the actor and critic respectively. The objective of actor-critic is to improve the accuracy in estimating the returns using critic, followed by optimizing the estimated returns by updating the actor (equation 4.18). The learning gradient of the policy is considered an approximate solution to the Bellman optimality equation (equation 4.19). A baseline $x(s_t, a_t)$ term limits the variance in the gradients of the neural network approximators aiding in convergence (equation 4.20) [21].

$$max_{\theta_A} J(\theta_A) = E(R_t \mid \pi_{\theta_A}) \tag{4.18}$$

$$\nabla_{\theta_A} J(\theta_A) = E_\pi[\sum_{t=0}^{N} \nabla_{\theta_A} \log \pi_{\theta_A}(a_t \mid s_t)[R_t]] \tag{4.19}$$

$$A(s_t, a_t) = R_t - x(s_t, a_t) \tag{4.20}$$

λ samples of tuples containing the state, action, action-value and the reward are recorded in the experience replay buffer for each sample time t until the buffer is full. The experience replay buffer holds the information required to calculate the gradient from losses. Since the objective of the critic is accurate estimation of the returns, the critic parameters are updated by means of the critic loss function as shown in equation 4.21. The returns R_t are calculated from the rewards stored in the experience replay buffer, while the returns estimate (the action-value function) is obtained by passing the state/action information to the critic network.

$$\min_{\theta_C} J(\theta_C) = \sum_{t=0}^{N} || R_t - Q_{\theta_C}(s_t, a_t) ||^2 \tag{4.21}$$

After the network parameters' update in the critic network, the actor is updated by means of the actor network gradient derived from its loss. The actor loss is adjusted by the advantage function to reduce variance such as in Advantage Actor-Critic (A2C) where the critic action-value replaces $x(s_t, a_t)$ with $A(s_t, a_t)$ calculation as presented in equation 4.22.

$$\nabla_{\theta_A} J(\theta_A) = E_\pi[\sum_{t=0}^{N} \nabla_{\theta_A} \log \pi_{\theta_A}(a_t \mid s_t)[A_{\theta_C}(s_t, a_t)]] \tag{4.22}$$

4.3.3 Exploration

Policy π can either be deterministic (equation 4.23), that is, the policy maps the state observations s_t directly to the actions a_t, or stochastic (equation 4.24), where the policy samples a probability distribution described by μ_t, σ_t from which the action is sampled (equation 4.25). Stochastic policies inherently promote exploration making it suitable for improved convergence for continuous space, nonlinear chemical processes. Whereas, in the case of deterministic policies, exploration is encouraged by means of schemes such as ϵ-greedy or ϵ-soft.

$$a_t = \pi_{\theta_A}(s_t) \tag{4.23}$$

$$\mu_t, \sigma_t = \pi_{\theta_A}(s_t) \tag{4.24}$$

$$a_t \sim N(\mu_t, \sigma_t) \tag{4.25}$$

The theme of exploration and exploitation is central to reinforcement learning. Exploitation is when the agent chooses the action known to result in the highest returns, while exploration is the agent taking an equal probability action to explore the action space. Furthermore, Shannon's entropy $H(\pi)$ is introduced in the actor loss calculation to encourage exploration in the stochashtic format (equation 4.26). Higher entropy may result in delayed convergence while preventing convergence to a local optima. The actor loss is hence represented as given in equation 4.27 where β is a hyper-parameter representing the trade-off between optimizing the advantage function and exploration [52].

$$H(\pi) = -\sum_{t} P(a_t) \log P(a_t) \tag{4.26}$$

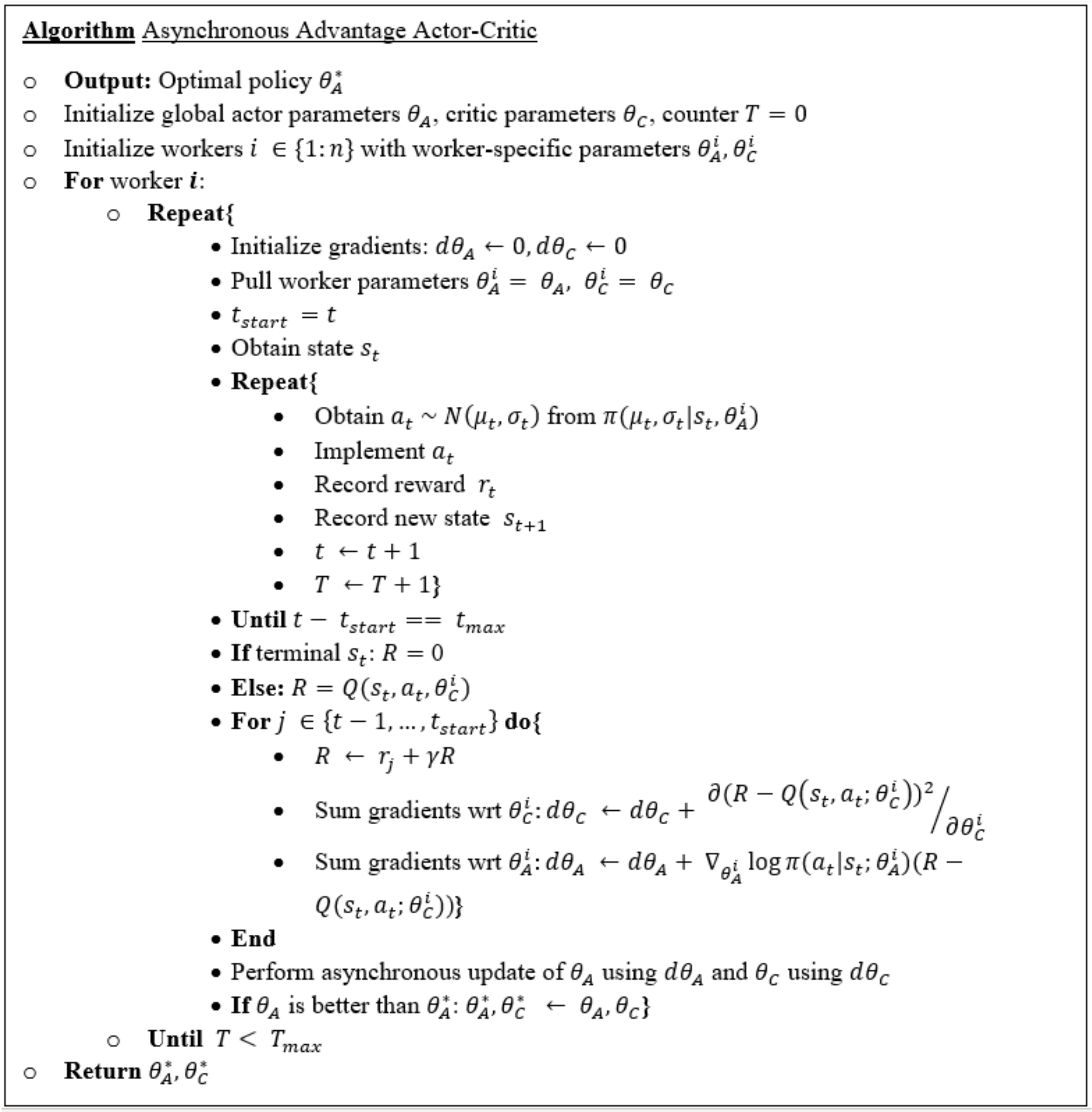
$$\nabla_{\theta_A} J(\theta_A) = E_\pi[\sum_{t=0}^{N} \nabla_{\theta_A}[\log \pi_{\theta_A}(a_t \mid s_t)[A_{\theta_C}(s_t, a_t)] - \beta * H(\pi)] \tag{4.27}$$

4.3.4 Asynchronous advantage actor-critic

The learning approach differs slightly between on-policy algorithms and off-policy algorithms. On policy algorithms interact with the environment using the same policy that they update to converge towards the optimal policy. Off-policy algorithms interact with the environment using a behavior policy, while a separate target policy is updated to find the optimal policy. Asynchronous advantage actor-critic (A3C) is an instance of such off-policy scheme where a global actor-critic network is updated using the experience gained through multiple local actor-critic behavior policies working asynchronously. Each local actor-critic interacts with its own local copy of the environment (in this case the PSV and the rewarding mechanism) to gain the experience. This aids exploration in the state/action space which is essential for development of a generalized solution for nonlinear process control applications. By employing multiple local copies of actor-critic and its corresponding environment, A3C redistributes the learning between multiple workers by making use of parallel computing. This also leads to improved and stable convergence [1]. The pseudocode of the A3C adapted from [1] for the PSV is given in figure 4.4. The sequence repeats itself for each worker for each learning episode except for the first time each worker interacts with the environment, where no updates are made to the worker networks.

4.3.5 Hierarchical multiloop control

The contribution of this chapter comes from the proposed hierarchical control scheme that tackles the multiple objectives of the problem statement presented by the PSV. The supervisory layer of the hierarchical agent overlooks optimization of the bitumen recovery rate. The bitumen recovery rate RR is controlled through changes in the froth-middlings interface level setpoint $I_{F-M_{SP}}$. A lower level RL agent manipulates the interface level I_{F-M} to track the setpoint changes. In addition, to ensure safe operation of the PSV, another non-coordinating lower level RL agent regulates the tailings density ρ_t below a set threshold to prevent sanding in the tailings. The control structure is illustrated in figure 4.5. The bitumen recovery rate optimization is carried out through the supervisory RL



```
Algorithm Asynchronous Advantage Actor-Critic
○ Output: Optimal policy $\theta_A^*$
○ Initialize global actor parameters $\theta_A$, critic parameters $\theta_C$, counter $T = 0$
○ Initialize workers $i \in \{1{:}n\}$ with worker-specific parameters $\theta_A^i, \theta_C^i$
○ For worker $i$:
    ○ Repeat{
        • Initialize gradients: $d\theta_A \leftarrow 0, d\theta_C \leftarrow 0$
        • Pull worker parameters $\theta_A^i = \theta_A$, $\theta_C^i = \theta_C$
        • $t_{start} = t$
        • Obtain state $s_t$
        • Repeat{
            • Obtain $a_t \sim N(\mu_t, \sigma_t)$ from $\pi(\mu_t, \sigma_t | s_t, \theta_A^i)$
            • Implement $a_t$
            • Record reward $r_t$
            • Record new state $s_{t+1}$
            • $t \leftarrow t + 1$
            • $T \leftarrow T + 1\}$
        • Until $t - t_{start} == t_{max}$
        • If terminal $s_t$: $R = 0$
        • Else: $R = Q(s_t, a_t, \theta_C^i)$
        • For $j \in \{t-1, \ldots, t_{start}\}$ do{
            • $R \leftarrow r_j + \gamma R$
            • Sum gradients wrt $\theta_C^i$: $d\theta_C \leftarrow d\theta_C + \partial(R - Q(s_t, a_t; \theta_C^i))^2 / \partial\theta_C^i$
            • Sum gradients wrt $\theta_A^i$: $d\theta_A \leftarrow d\theta_A + \nabla_{\theta_A^i} \log \pi(a_t | s_t; \theta_A^i)(R - Q(s_t, a_t; \theta_C^i))\}$
        • End
        • Perform asynchronous update of $\theta_A$ using $d\theta_A$ and $\theta_C$ using $d\theta_C$
        • If $\theta_A$ is better than $\theta_A^*$: $\theta_A^*, \theta_C^* \leftarrow \theta_A, \theta_C\}$
    ○ Until $T < T_{max}$
○ Return $\theta_A^*, \theta_C^*$
```

Figure 4.4: Pseudocode for A3C adapted from [1] for the PSV

agent that determines the setpoint for the froth-middlings interface level. The lower level RL agent tracks the setpoint changes to the interface level, and the sanding prevention RL agent regulates the tailing density as explained in the following subsections. The reward mechanism of each RL agent, the states to the actor and the critic, and the loss functions determine the learning of each agent and comprise the setup. The setup overlooking each objective is explained in subsections 3.5.1 to 3.5.3.

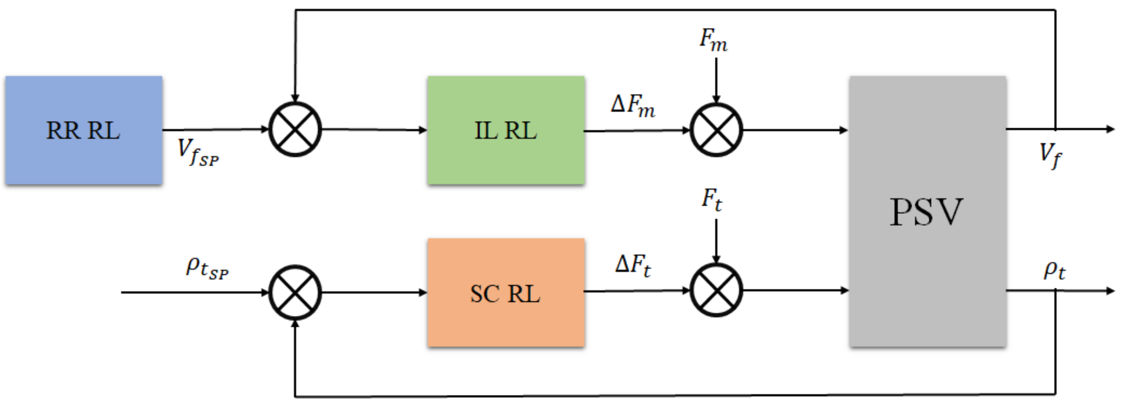


Figure 4.5: Multiloop control of PSV

Low-Level RL - Interface Level Control

Control of the froth-middlings interface level is achieved through manipulation of middlings flowrate F_m. The interface level I_{F-M} depends directly on the froth volume V_f as given in equation 4.28, where A_{vessel} is the area of the vessel. A finite-difference type simulation with a sample time of 1 hour (with one minute iterations at the lower loop) is executed. A new action ΔF_m is chosen by the actor based on the state observations $s_t^{IL_A}$ every 1 hour. The states observed by the actor, that is the input vector to the actor, is given in equation 4.29. Since a stochastic policy is followed, the output of the actor is an action probability distribution as shown in equation 4.30. The normalized action is sampled from the distribution and scaled to the PSV's practical operating range, shown in equation 4.31. The action then updates the middlings flowrate F_m as shown in equation 4.32. It is hard constrained to a ±20% bound of the nominal value of the middlings flowrate $\overline{F_m}$ (equation 4.33).

$$I_{F-M}(t) = \frac{V_f(t)}{A_{vessel}} \tag{4.28}$$

$$s_t^{IL_A} = [I_{F-M}(t), I_{F-M_{SP}}(t)] \tag{4.29}$$

$$\mu_t^{IL}, \sigma_t^{IL} = \pi_{\theta_{IL_A}}(s_t^{IL_A}) \tag{4.30}$$

$$\Delta F_m(t) \sim N(\mu_t^{IL}, \sigma_t^{IL}) \tag{4.31}$$

$$F_m(t) = F_m(t) + \Delta F_m(t) \tag{4.32}$$

$$0.8\overline{F_m} \leq F_m(t) \leq 1.2\overline{F_m} \tag{4.33}$$

The states observed by the critic $s_t^{IL_C}$ are given in equation 4.34 and further explained in equation 4.35. They are designed to provide the critic with the information required to calculate the returns using the value function. The output of the critic is hence estimates returns R_t^{IL} in the form of the value function $V_{\theta_C}(s_t)$ (equation 4.36).

$$s_t^{IL_C} = [\Delta I_{F-M}(t)] \tag{4.34}$$

$$\Delta I_{F-M}(t) = I_{F-M}(t) - I_{F-M_{SP}}(t) \tag{4.35}$$

$$\hat{R_t^{IL}} = V_{\theta_{IL_C}}(s_t^{IL_C}) \tag{4.36}$$

The reward function, given in equation 4.37, is shaped such as to minimize the deviation of the interface level from the setpoint, and is handled in terms of the froth-middlings interface level deviation given in equation 4.35.

$$r_t^{IL} = - \mid \Delta I_{F-M}(t) \mid \tag{4.37}$$

The critic loss, given in equation 4.21, employs the value function ($V_{\theta_{IL_C}}(s_t^{IL_C})$) and actual returns are calculated from equation 4.37. The actor loss is calculated with the advantage function values from the updated critic as shown in equation 4.22. The results obtained are shared in section 4.4.

Supervisory RL - Recovery Rate Optimization

The recovery rate (RR) depends on the bitumen content in the froth (α_b^f) and ore (α_b^{fd}), and the corresponding froth overflow rate (F_f) and ore flowrate (F_{ore}) as given in equation 4.13. Since the bitumen content in the ore and the ore flow rate are beyond control, the froth-middlings interface level (I_{F-M}) is considered to address the recovery rate.

The sampling time considered here is 2 hours since that is the frequency at which the ore quality measurements from the lab will be available (based on discussions with industrial partners). The input state vector observed by the actor is given in equation 4.38. These states are the recovery rate at the given time RR and the baseline recovery rate RR_{SP} taken from [41]. A stochastic policy is followed again, so the output of the actor is an action probability distribution, as shown in equation 4.39. The normalized control action, change in froth-middlings interface level setpoint $\Delta I_{F-M_{SP}}$, is sampled from the given distribution (equation4.40) and scaled to a practical operating range (equation 4.41), that is $\pm 1.2m$. The range for setpoints for the froth-middlings interface level is also set between the operating limits of $18.8m$ to $28.2m$.

$$s_t^{RR_A} = [RR, RR_{SP}] \tag{4.38}$$

$$\mu_t^{RR}, \sigma_t^{RR} = \pi_{\theta_{RR_A}}(s_t^{RR_A}) \tag{4.39}$$

$$\Delta I_{F-M_{SP}}(t) \sim N(\mu_t^{RR}, \sigma_t^{RR}) \tag{4.40}$$

$$-1.2 \leq \Delta I_{F-M_{SP}}(t) \leq 1.2 \tag{4.41}$$

For estimation of the returns, the states $s_t^{RR_C}$ are provided to the critic as shown in equation 4.42 and expounded in equation 4.43. The instantaneous rewards reflect the objective to maximize the recovery rate and maintain the system's stability by minimizing the magnitude of the action taken. The reward reflected is positive in the case when the recovery rate is above the nominal recovery rate RR_{SP} and negative otherwise, as given in equation 4.44. They are used in the actual returns R_t^{RR} calculation.

$$s_t^{RR_C} = [\Delta RR, -\Delta I_{F-M_{SP}}] \tag{4.42}$$

$$\Delta RR = RR - RR_{SP} \tag{4.43}$$

$$r_t^{RR} = \Delta RR - 0.5\Delta I_{F-M_{SP}} \tag{4.44}$$

The critic loss (equation 4.21) and actor loss (equation 4.22) extract information from the supervisory RL agent in a similar manner as the previous subsection and follow the same sequence of update. The results are shared in the next section.

Sanding Prevention

Accumulation of coarse solids in the tailings underflow adversely affects the pipe health and can choke the PSV. This phenomenon is known as sanding, and it occurs when the tailings density (ρ_t) increases beyond a certain threshold, causing solids to settle quicker than they can be removed. Through control of the tailings flowrate (F_t), the tailings density can be regulated below the sanding threshold, which is the third objective this work looks to optimize. The sanding threshold is given as $1650 kgm^{-3}$ in literature [37]. However, in the current study, a lower threshold of $1480 kgm^{-3}$ is chosen as a tighter constraint. With the same sampling time similar to the interface level tracking case, the states observed by the actor $s_t^{RR_A}$ are given in equation 4.45. The output of the actor is a probability distribution as shown in equation 4.46 from which the action ΔF_t is sampled every 1 hour (equation 4.47). The action updates the tailings withdrawal flowrate F_t as shown in equation 4.48. It is then hard-constrained within a feasible range of ±20% of its nominal value $\overline{F_t}$, represented in equation 4.49.

$$s_t^{SC_A} = [\rho_t, \rho_{t_{SP}}] \tag{4.45}$$

$$\mu_t^{SC}, \sigma_t^{SC} = \pi_{\theta_{SC_A}}(s_t^{SC_A}) \tag{4.46}$$

$$\Delta F_t(t) \sim N(\mu_t^{SC}, \sigma_t^{SC}) \tag{4.47}$$

$$F_t(t) = F_t(t) + \Delta F_t(t) \tag{4.48}$$

$$0.8\overline{F_t} \leq F_t(t) \leq 1.2\overline{F_t} \tag{4.49}$$

Similar to the previous two cases, the critic input states $s_t^{SC_C}$, given in equation 4.50, encapsulate the information required for an estimation of the returns. The actual instantaneous rewards are given in equation 4.51 and would be used to calculate the actual returns used in the critic loss function represented in equation 4.21, which will then be used to calculate the actor loss as shown in equation 4.22.

$$s_t^{SC_C} = [\Delta\rho_t, \Delta Q_t] \tag{4.50}$$

$$\Delta\rho_t = \rho_t - \rho_{t_{SP}}$$

$$r_t^{SC} = - \mid \Delta\rho_t \mid \tag{4.51}$$

Simulation details and results are provided in section 4.

Coerced Learning

Another novel contribution of this chapter is leveraging the existing control strategy to initially teach the RL agent to learn and explore in the stable operational region of the state/action space. This is especially useful when dealing with a nonlinear process such as the PSV. This is an adaptation

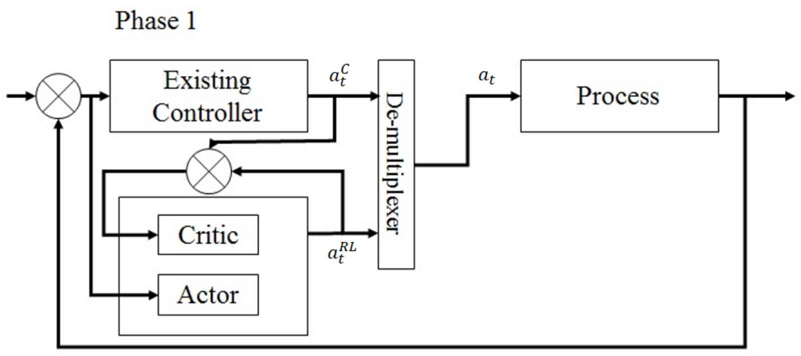


Figure 4.6: Phase 1 of coerced learning

of the imitation learning concept into this work. The strategy developed has been termed coerced learning and was implemented by learning from an interactive expert demonstrator namely learn from existing control strategy.

The training is carried out in 2 phases. In Phase 1, the action taken by the actor-critic is limited subject to a defined bound of the expert demonstrator's action. This is achieved by adding an additional factor in reward calculation in the first few episodes. In this phase, a demultiplexer chooses between the RL agent's action a_t^{RL} and the expert controller's action a_t^C as given in equation 4.52 and illustrated in figure 4.6. The RL agent's action is evaluated for the regular reward if it is within ±5% of the action that the expert demonstrator would choose for the given measurements. A penalizing mechanism considering the distance of the RL agent's action from the bounds is utilized otherwise. Beyond these bounds, the coerced learning factor *cc* factor penalizes the actions taken by the RL agent. This is represented in equation 4.53, where A represents the complete practical range of actions. The penalty is hence proportional to the deviation between the action taken by the RL agent and the expert demonstrator. If the RL agent's action lies within the acceptable range, the reward is proportional to the deviation from the setpoint in the case of setpoint tracking.

$$a_t = \begin{cases} a_t^C, & 0.95a_t^C < a_t^{RL} < 1.05a_t^C \\ a_t^{RL}, & otherwise \end{cases} \tag{4.52}$$

$$r_t = \begin{cases} - \mid I_{F-M_{SP}} - I_{F-M} \mid, & 0.95a_t^C < a_t^{RL} < 1.05a_t^C \\ \frac{cc}{A - |a_t^C - a_t^{RL}|}, & otherwise \end{cases} \tag{4.53}$$

After a specified number of episodes, the training switches to phase 2. In this phase, the RL agent's learning is independent of the expert demonstrator, as shown in figure 4.7. Compared to this, in behavior cloning, the learner assumes the demonstrator's policy to be optimal and aims to imitate it

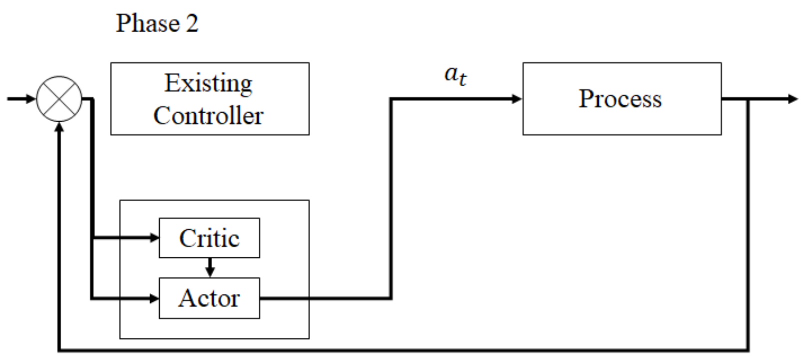


Figure 4.7: Phase 2 of coerced control training

by copying the actions it takes in given states. This is, thus, a semi-supervised learning scheme where the RL agent leverages experience from the expert demonstrator (like a conventional controller) to define the direction for exploration to ensure that the exploration happens within a stable region while control objectives are met. The impact of introducing this factor on the training and on-line execution along with its wider implications for control are discussed in section 4.

4.4 Results & discussion

4.4.1 Infrastructure

For this study, a high fidelity model of the PSV was considered. The PSV simulation as well as the RL code was implemented using Tensorflow v. 1.9.0 in Python 3.7.1. Windows 10 64-bit OS running on a Lambda computer with Intel i9-9820x processor with 20 threads was utilized for the A3C based learning.

Details of the input and output vectors to the actor as well as the critic have been elaborated in Section 3.5 for the hierarchical architecture based agents as well as sanding prevention scheme. Fully connected feedforward neural networks are used as function approximators for both the actor and the critic. There is 1 hidden layer for the actor, in all cases, which contains 200 nodes, and it is fully connected to the output layer, with a nonlinear activation function applied to its output. The output layer of the actor consists of a mean and standard deviation as shown in figure 4, from which the actions are sampled. The nodes corresponding to the mean (μ_t) have a *tanh* activation function applied in the output layer. The nodes corresponding to the standard deviation (σ_t) have a $softplus$ activation function applied in the output layer. Similarly the critic is structured with an input layer fully connected to 1 hidden layer with 100 nodes using a *tanh* activation function, which is in turn fully connected to the output layer. The sample time for each policy is mentioned in the corresponding sections.

4.4.2 Low-level RL - interface level control

Setup

To comprehensively illustrate the performance of the RL agent in tracking the froth-middlings interface level setpoint, it is compared to the conventional auxiliary controller enhanced from [41]. In a similar fashion to the RL agent, its control output takes into account the deviation of the interface level from its setpoint to determine the error term (e_t) as given in equation (4.54).

$$e_t = \Delta I_{F-M}(t) = I_{F-M}(t) - I_{F-M_{SP}}(t) \tag{4.54}$$

As mentioned in section 3.5.4, training occurred over two distinct phases, termed coerced learning. In phase 1 of coerced learning, the RL agent's action was limited to a defined bound from the expert demonstrator's (conventional controller) action. In phase 2, it was allowed to explore the action space freely. The sampling time for the control action taken by both the RL agent and the conventional controller is 1 hour. The RL agent is trained for a total of 20,000 episodes (constituting 4000 hours each), out of which, the first 1500 episodes are spent in phase 1 and the remaining are spent in phase 2.

Results

Servotracking based on versatile direction and magnitude changes to interface level setpoint $I_{F-M_{SP}}$ is carried out. The setpoint change is instigated once every 400 hours. All process and manipulated variables are recorded for quantitative assessment.

Without coerced learning, the actions taken by the RL agent in the initial episodes led the PSV to the unstable region from which it could not recover. This hindered learning and eventually the convergence to the optimal policy. Coerced learning enabled the RL agent to learn from the conventional controller to find a stable operating region, as shown in figure 4.8. As shown in subplots (a-c) of figure 4.8, the control performance improves as more training episodes elapse. This corresponds to the RL agent learning to take actions within the stable operating region, denoted by the upper and lower bounds on subplots (d-f) of figure 4.8. The adherence to this region improves from episode 500 to episode 1500 subplots (d & f) respectively in figure 4.8.

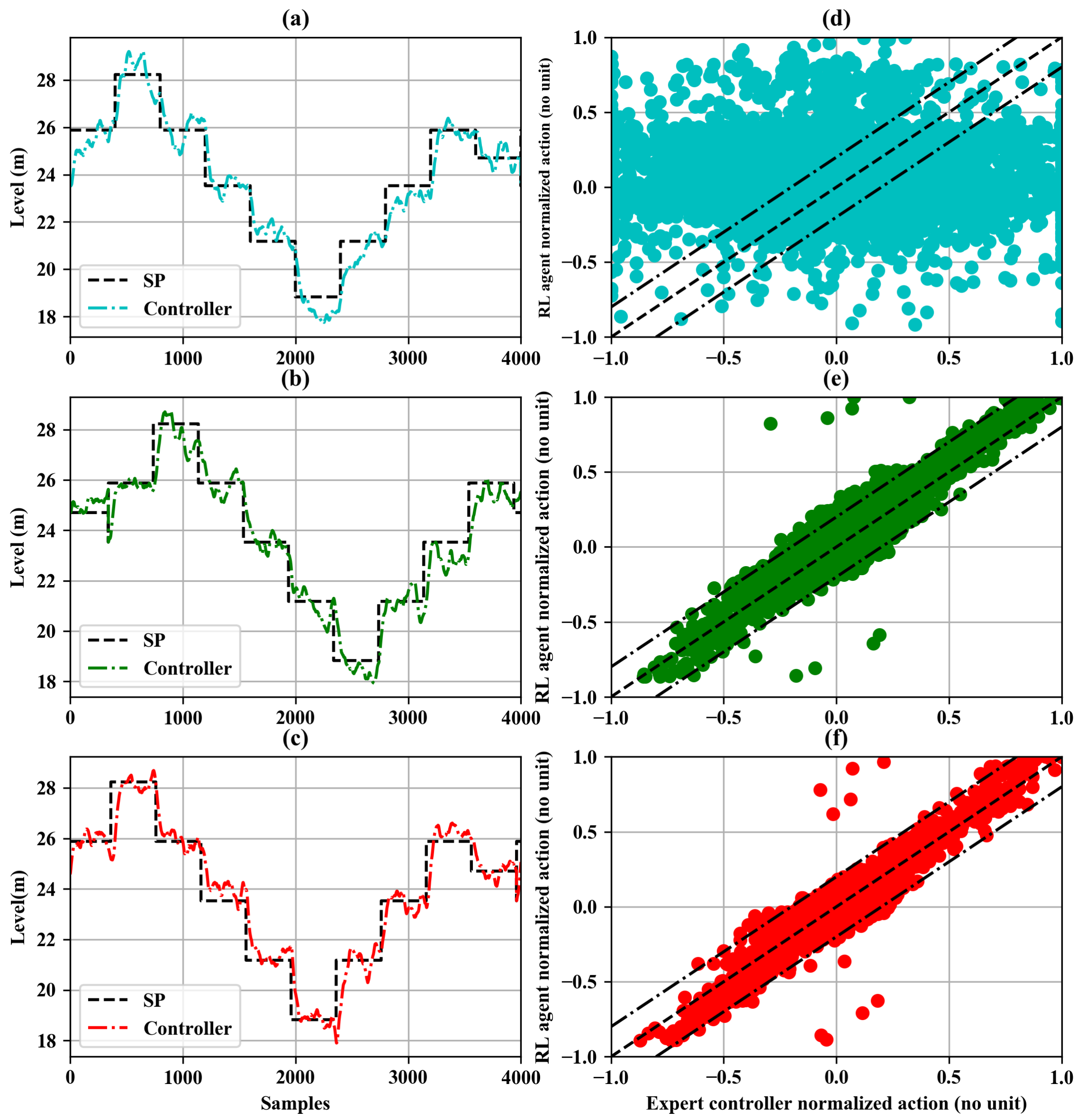


Figure 4.8: Phase 1 of coerced learning: setpoint tracking at (a) 500 episodes, (b) 1000 episodes, (c) 1500 episodes; control action taken by RL agent relative to an expert demonstrator at (d) 500 episodes, (e) 1000 episodes, and (f) 1500 episodes

In phase 2, the rewarding structure is based only on the control objective for the interface level. The best policy is based on the best cumulative rewards obtained in any episode in phase 2. As illustrated in figure 4.9, the RL agent has obtained the best projection of the state space into the action space. As evident from the figure and metrics such as mean squared error (MSE) and integral of absolute error (IAE), the RL agent tracks the setpoint better in comparison to the conventional controller. Furthermore, the ability of the RL agent to track the setpoint in the presence of disturbances relative to the conventional controller is tested. White noise of magnitude $\pm 30\%$ of the nominal bitumen content in the ore (α_b^{ore}) is included. The results obtained are displayed in figure 4.10. Hence, the RL agent displays successful servotracking abilities in the presence of varied bitumen content in the ore. The RL agent generalizes well over varying operating conditions, controlling the interface level to track the setpoint. The control performance is assessed by MSE, IAE, and variance of control (VC), which are provided in table 4.2.

Table 4.2: Froth-middlings interface setpoint tracking results

Control Scheme	**MSE**	**IAE**	**VC**
Constant ore quality			
RL Controller	12.61265	4955.31166	6.52318e-09
Conventional Controller	34.29068	12122.17030	8.79923e-10
Varying ore quality			
RL Controller	14.65082	8642.64619	8.34093e-09
Conventional Controller	41.80831	19805.94206	1.06232e-09

As is also visible in figure 4.9 and figure 4.10, the RL agent significantly outperforms the conventional controller in terms of MSE and IAE as presented in table 4.2. The lower MSE conveys the RL agent's ability to maintain lower variance of the interface level I_{F-M} from the setpoint $I_{F-M_{SP}}$ overall while the lower IAE shows that less error is accumulated over time. This is true for both the cases: with constant ore quality and with varying ore quality. This shows the effectiveness of coerced learning in leveraging imitation learning to learn from the conventional controller in phase 1 of training and eventually outperforming it without the need for model information. The RL agent, however, has a greater variance of control (VC) in both cases. Although the VC is within the acceptable range, this hints that the conventional controller is smoother. This is easily explained by the reward function shaping of the RL agent since it is the only feedback the controller received on its performance. A possible solution is to include a term penalizing the action taken by the RL agent, to ensure a smoother policy.

4.4.3 Supervisory RL - recovery rate optimization

Setup

The bitumen recovery rate is presented in section 2.6. The handle used to address the recovery rate RR was chosen to be the froth-middlings interface level $I_{F-M_{SP}}$. The sampling interval is 2

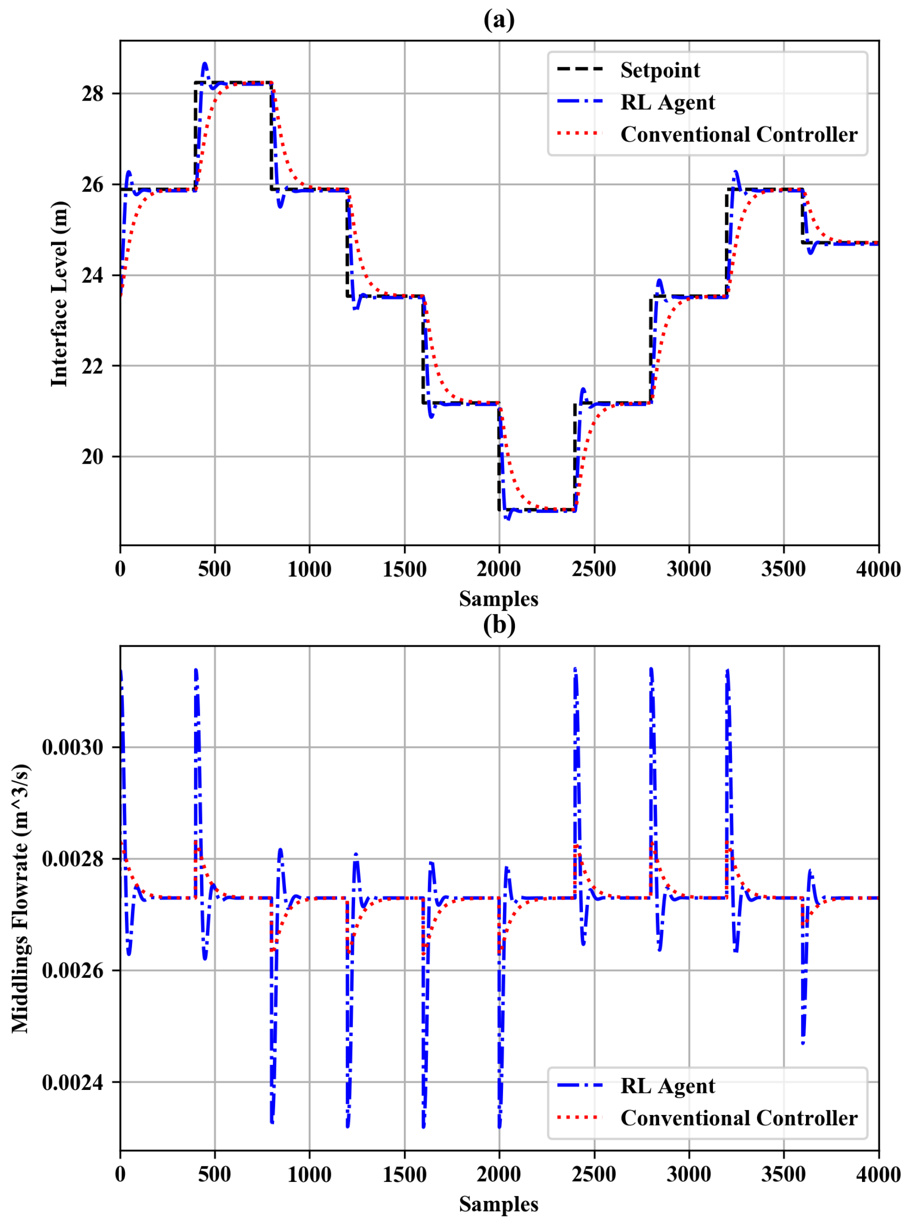


Figure 4.9: Froth-middlings interface setpoint tracking results

hours corresponding to the frequency at which lab samples are available (section 4.3.5). The action space of the supervisory RL agent is to vary the froth-middlings interface level setpoint $\Delta I_{F-M_{SP}}$ which then prompts the lower level RL agent to manipulate the middling flowrate ΔF_m to track the updated setpoint, completing the hierarchy.

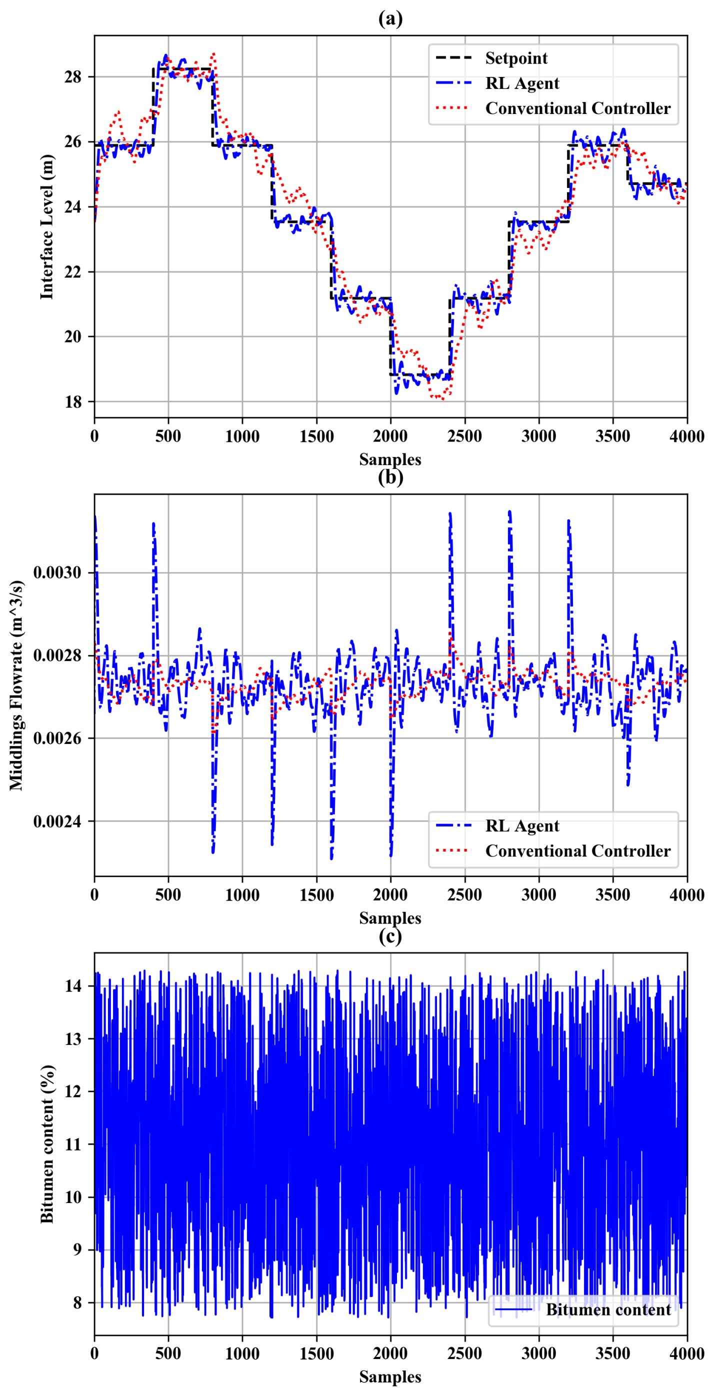


Figure 4.10: Froth-middlings interface setpoint tracking results with varying ore quality: (a) interface level, (b) middlings withdrawal flowrate, and (c) ore quality

Results

The RL agent was trained for 20,000 episodes where each episode ran for 100 hours, with a sampling interval of 1 hour. The results are displayed in figure 4.11 relative to the regulated interface level.

Subplot (a) of figure 4.11 displays the recovery rate RR while subplot (b) displays the control action $\Delta I_{F-M_{SP}}$ taken by the supervisory RL agent to maximize the recovery rate. Subplot (c) indicates the control action ΔF_m taken by the low-level RL agent to track the updated setpoint. The RR peaks above 1 and is explained through the observation that the froth volume (V_f) decreases in accordance with the froth volume setpoint changes directed by the supervisory RL agent in the hierarchical control scheme. It then finally settles to a value around the regulated interface level scheme value as the supervisory RL agent in the hierarchical control scheme ordains a final value for the froth-middlings interface setpoint. The overall RR relative to the regulated interface level is represented in table 4.3.

Table 4.3: Recovery rate optimization results

Control Scheme	**Average Recovery Rate**
RL Controller	0.85715
Open loop	0.76113

From figure 4.11 and table 4.3, it is clear that the RL based hierarchical control scheme is able to achieve a significantly higher average recovery rate RR as compared to the regulated interface level. It is able to do this while maintaining the change in setpoint ($\Delta I_{F-M_{SP}}$), the interface level (I_{F-M}), and the middlings flowrate (F_m) within operational limits.

4.4.4 Sanding prevention

Setup

Sanding prevention is implemented as a safety measure to ensure regular PSV function during interface level tracking. The tailings density (ρ_t) is regulated through the tailings flowrate (F_t). This control is activated when the tailings density exceeds a set sanding threshold. The sanding prevention RL agent then manipulates the tailings density by action (ΔF_t) to bring the tailings density below the set threshold. The threshold used in this experiment is set tighter than the industrial standard reported in literature $1680kgm^{-3}$ and is set at $1480kgm^{-3}$ to demonstrate the effectiveness [42]. This low-level sanding prevention RL agent is built to co-exist with the low-level interface level control RL agent reported in Section 4.2. A similar sample time is followed here. The two loops are sequentially executed during simulation.

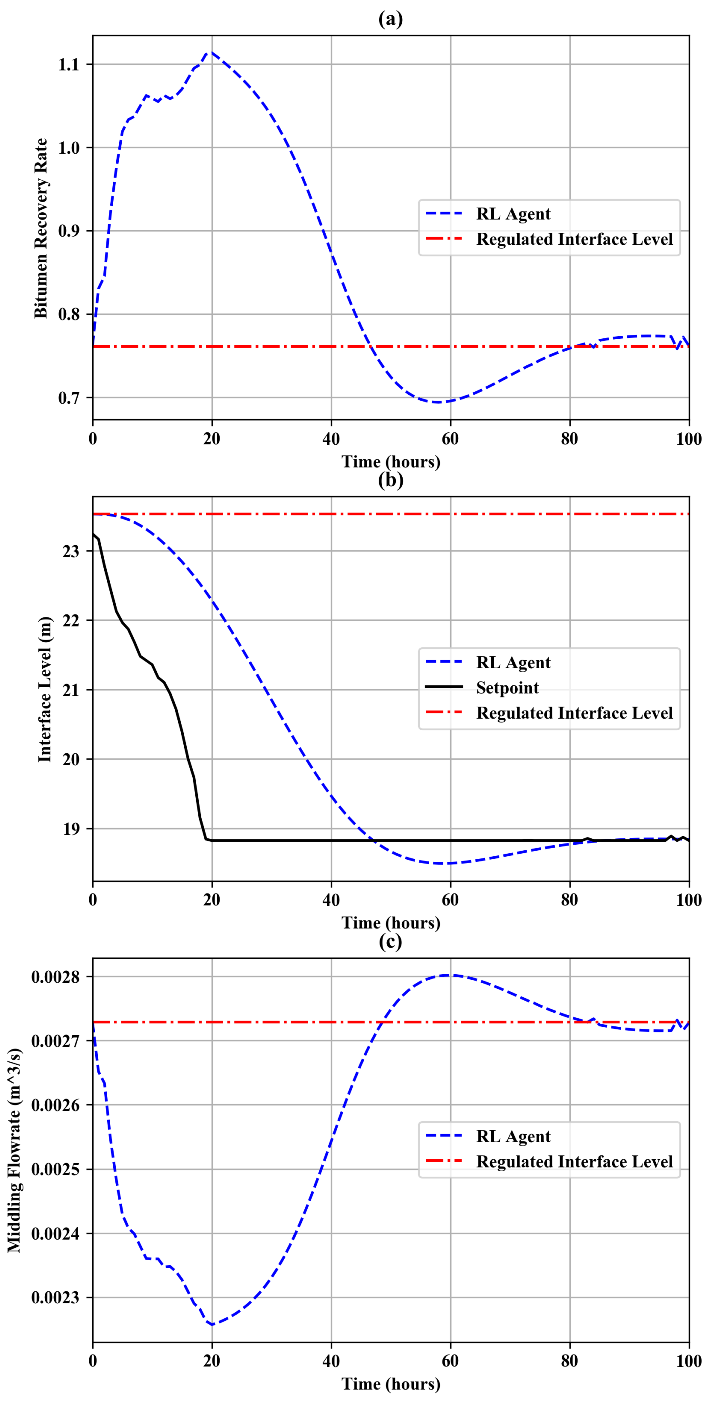


Figure 4.11: Recovery rate optimization results: (a) recovery rate, (b) interface level, and (c) middlings withdrawal flowrate

Results

The sanding prevention RL agent was trained for 20,000 episodes of 4000 hours each. A control action (ΔF_t) was chosen every 1 hour. While the setpoint was tracked by the low-level interface level control RL agent detailed in section 4.2 by determining a control action (ΔF_m), the tailing density is controlled through the non-interacting low-level sanding prevention RL agent determining a control action (ΔF_t) concurrently. The results obtained are shown in figure 4.12. As the subplot (a) of figure 4.12 shows, the low-level sanding prevention RL agent is able to successfully bring the tailings density (ρ_t) below the sanding threshold every time it goes beyond the threshold during interface level changes. These correspond to the changes in tailings flowrate (F_t) at the times when the control is activated as shown in subplot (b) of figure 4.12.

4.5 Conclusions

In this work, an RL based control strategy was developed to implement effective hierarchical control for PSV. The Supervisory A3C based RL agent manipulated the interface level set point to improve the bitumen recovery rate. The resulting lower level RL agent for servotracking and ore quality variance oriented regulation of interface level was implemented using an A3C based middlings flow rate manipulation. A sanding prevention scheme was also implemented using a separate A3C based RL agent. The A3C based global RL agents learn the optimal middlings and tailings flowrates to obtain each defined objective. The RL agents map the state space on to the action space through the experience gained by repeated interactions with a high fidelity model of the PSV. The existing conventional control strategy was leveraged using a variant of behaviour cloning, termed as coerced learning. This initially assisted the RL agent in discovering the stable operating region of the action space given the nonlinear nature of the gravity-based separation process. From there, the RL agent was able to independently learn the optimal actions to be taken in the range to achieve its goals. The lower level RL agent for interface level control demonstrated better performance in terms of IAE and MSE for stable and varying ore quality relative to the conventional controller. The supervisory RL agent also demonstrated a higher bitumen recovery rate than reported with conventional control in keeping with the economic optimization objectives. The low-level RL agent for sanding prevention was also able to maintain the tailings density below the set threshold to prevent sanding amidst tracking the setpoint changes in the interface level. Future work can focus on constrained RL design

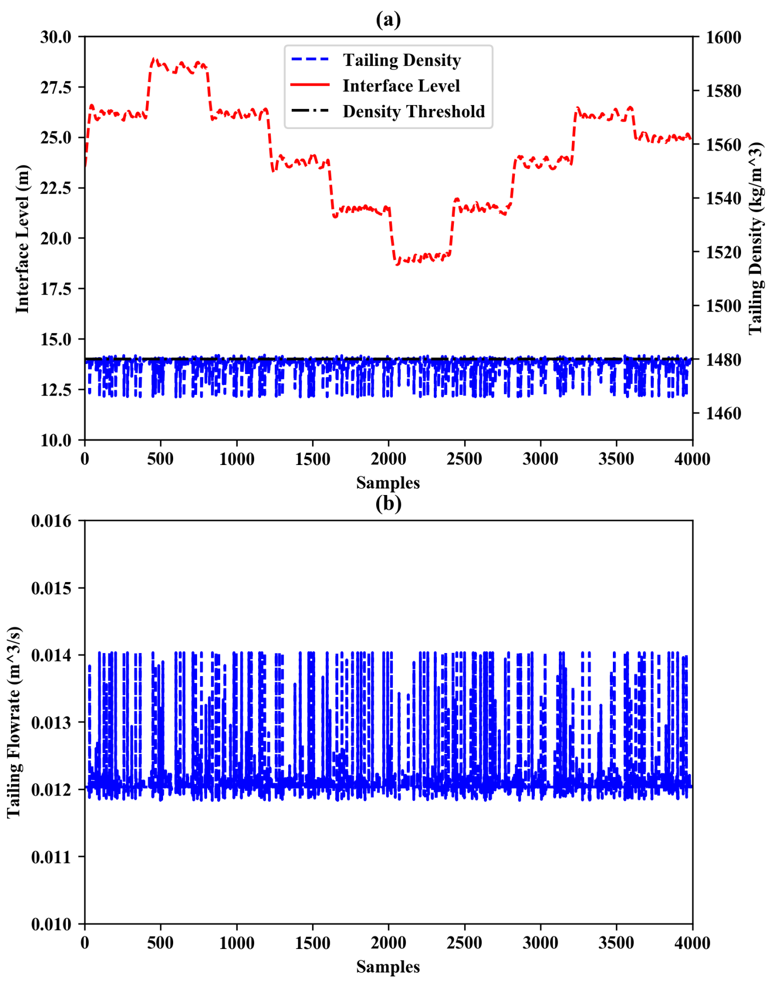


Figure 4.12: Sanding prevention results: (a) interface level and tailings density, and (b) tailings flowrate

for safe exploration of the action-state space during the training phase.

Chapter 5

Conclusions & Future Work

5.1 Summary of thesis

This thesis has focused on solving the multifaceted problem of leveraging reinforcement learning in process control applications. Reinforcement learning has been applied successfully to a variety of fields from robotics to medical diagnosis to natural language processing. Recent developments in the field including the use of neural networks as function approximators have allowed RL algorithms to be applied to continuous action space and continuous state space control problems. This has made process control a candidate for RL based solutions.

Model-free RL brings data-driven solutions that learn the control strategy directly from interaction with process data without the need for the process model. This is especially beneficial in the case of nonlinear processes where the process model might not be readily available or accurate. It circumvents the need for a model identification step. It is also able to re-train in the case of process shifts or process noise to improve performance. In contrast, traditional model-based control methods require an explicit process model, and the performance of parametric models deteriorates over time in case of process shifts or disturbances.

The first part of the thesis focused on developing an RL based control strategy for a benchmark MIMO nonlinear system: the quadruple tank system. It used A3C as the RL agent for servotracking of levels in the quadruple tank system. With appropriate feature selection, the RL agent improved its control performance over experience gained through multiple interactions with the process. A study on the effect of the number of parallel learners on the control quality and convergence onto an optimal policy was also carried out. It indicated identical control performance but suggested a higher degree of action and state space exploration with higher degree of parallel learning.

The second part of the thesis focused on employing RL's goal-based learning to develop a hierarchical control for bitumen recovery rate optimization of a PSV. Bitumen recovery rate optimization was carried out by a supervisory A3C based RL agent through manipulation of the froth-middlings interface level. A lower level RL implemented servotracking of the forth-middlings interface level in presence of ore quality variance through middlings flowrate manipulation. Sanding Prevention RL was also implemented as a safety feature for control of the tailings density below a sanding threshold through tailings withdrawal flowrate manipulation. This work highlighted the need to limit the RL exploration in nonlinear processes within a sustainable range to keep the process stable yet allowing RL to interact with the process to learn. For this purpose, a semi-supervised learning scheme was proposed where the RL agent learns in two phases, 1) under the supervision of an existing expert demonstrator, and 2) independently. This assisted the RL agent in discovering the stable operating region of the action space and from there on it was able to independently learn the optimal actions and eventually outperformed the expert demonstrator.

5.2 Future work

5.2.1 Constrained RL

Future work can focus on constrained RL design for safe exploration of the action-state space during the training phase. Building on the two-phase semi-supervised learning strategy proposed in chapter 4, the constrained RL can adopt a rewarding mechanism that trains the RL agent relative to the performance of the existing control strategy, thus also leveraging any available control strategy. Another possibility is to start with an initial policy regressed using the existing control strategy and learn from there.

5.2.2 Lab & pilot scale implementation

A planned next step is employing transfer learning to use the RL control scheme implemented on the simulations to control the lab-scale quadruple tank system and the lab-scale PSV system. This would demonstrate the effectiveness of the RL control strategy on physical process control systems. Since interaction with these systems occurs through Delta V, once control is established on the lab-scale models, the next steps can plan for the deployment of the RL strategy on a pilot PSV.

Bibliography

[1] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning,” feb 2016.

[2] C. J. Cleveland and C. Morris, *Handbook of Energy. Volume II_ Chronologies, Top Ten Lists, and Word Clouds*, vol. II. Elsevier, 2014.

[3] Government of Canada, “What are the oil sands? — Natural Resources Canada,” 2018.

[4] CAPP, “Statistical Handbook for Canada 's Upstream Petroleum Industry,” *2016-9999*, no. January, p. 233, 2016.

[5] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data.* AMLBook, 2012.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag, 2006.

[7] R. Kurzweil, *The Singularity is Near*, pp. 393–406. London: Palgrave Macmillan UK, 2014.

[8] K. Horecki and J. Mazurkiewicz, “Natural language processing methods used for automatic prediction mechanism of related phenomenon,” in *Artificial Intelligence and Soft Computing* (L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, eds.), (Cham), pp. 13–24, Springer International Publishing, 2015.

[9] M. Brady, “Artificial intelligence and robotics,” in *Robotics and Artificial Intelligence* (M. Brady, L. A. Gerhardt, and H. F. Davidson, eds.), (Berlin, Heidelberg), pp. 47–63, Springer Berlin Heidelberg, 1984.

[10] T. de Souza Alves, C. S. de Oliveira, C. Sanin, and E. Szczerbicki, “From knowledge based vision systems to cognitive vision systems: A review,” *Procedia Computer Science*, vol. 126,

pp. 1855 – 1864, 2018. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.

[11] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and R. A. Peters, "A review of deep learning with special emphasis on architectures, applications and recent trends," *CoRR*, vol. abs/1905.13294, 2019.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[13] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction 2018 complete draft," *UCL,Computer Science Department, Reinforcement Learning Lectures*, 2017.

[14] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," in *Proceedings of the American Control Conference*, vol. 3, pp. 2143–2146, apr 1991.

[15] A. H. Klopf, "BRAIN FUNCTION AND ADAPTIVE SYSTEMS - A HETEROSTATIC THEORY.," 1974.

[16] J. Shin, T. A. Badgwell, K. H. Liu, and J. H. Lee, *Reinforcement Learning – Overview of recent progress and implications for process control*, vol. 127. Elsevier Masson SAS, 2019.

[17] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018.

[18] C. Watkins and R. Holloway, "Technical Note : Q-Learning Technical Note," vol. 8, no. May, pp. 279–292, 2014.

[19] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications," 2018.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," sep 2015.

[22] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," jul 2015.

[23] M. L. Giger, "Machine Learning in Medical Imaging," *Journal of the American College of Radiology*, vol. 15, pp. 512–520, mar 2018.

[24] J. Luketina, N. Nardelli, G. Farquhar, J. N. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel, "A survey of reinforcement learning informed by natural language," *CoRR*, vol. abs/1906.03926, 2019.

[25] L. Adaptation and A. Optimization, *Reinforcement Learning State-of-the-Art*, vol. 12.

[26] K.-T. Song and W.-Y. Sun, "Robot control optimization using reinforcement learning," *Journal of Intelligent and Robotic Systems*, vol. 21, pp. 221–238, Mar 1998.

[27] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2015.

[28] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring Robot Skills via Reinforcement Learning," *IEEE Control Systems*, vol. 14, no. 1, pp. 13–24, 1994.

[29] Y. Wang, K. Velswamy, and B. Huang, "A Novel Approach to Feedback Control with Deep Reinforcement Learning," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 31–36, 2018.

[30] Y. Wang, K. Velswamy, and B. Huang, "A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems," *Processes*, vol. 5, no. 3, 2017.

[31] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Adv. Neural Inf. Process. Syst*, vol. 12, 02 2000.

[32] V. Kirubakaran, T. K. Radhakrishnan, and N. Sivakumaran, "Distributed multiparametric model predictive control design for a quadruple tank process," *Measurement: Journal of the International Measurement Confederation*, vol. 47, no. 1, pp. 841–854, 2014.

[33] J. Kubalík, E. Alibekov, and R. Babuška, "Optimal control via reinforcement learning with symbolic policy approximation," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4162 – 4167, 2017. 20th IFAC World Congress.

[34] C. Sammut and G. I. Webb, eds., *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010.

[35] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *CoRR*, vol. abs/1712.01275, 2017.

[36] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *CoRR*, vol. abs/1507.04296, 2015.

[37] W. A. Gilbert, *Dynamic Simulation and Optimal Trajectory Planning for an Oilsand Primary Separation Vessel*. Thesis, University of Alberta, 2004.

[38] B. Li, F. Xu, Z. Ren, and A. Espejo, "Extended abstract: Primary separation vessel interface control," in *2011 International Symposium on Advanced Control of Industrial Processes, ADCONIP 2011*, pp. 262–264, 2011.

[39] J. H. Masllyah, T. K. Kwong, and F. A. Seyer, "Theoretical and Experimental Studies of a Gravity Separation Vessel," *Industrial and Engineering Chemistry Process Design and Development*, vol. 20, pp. 154–160, jan 1981.

[40] R. Masliyah, J., Cluett, W., Oxenford, J., Tipman, "Dynamic Simulation o f a Gravity Separation Vessel," *Control*, no. Mineral/Metallurgical Processing, 1984.

[41] S. Liu, J. Zhang, and J. Liu, "Economic MPC with terminal cost and application to oilsand separation," in *IFAC-PapersOnLine*, vol. 28, pp. 20–25, jul 2015.

[42] M. Forbes, "A Simple Dynamic Model of the Syncrude PSV," tech. rep., Syncrude Research Department.

[43] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," jun 2015.