Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Eivind Sjøvold

Autonomous Drilling Using Reinforcement Learning

Master's thesis in Industrial Cybernetics
Supervisor: Ole Morten Aamo
Co-supervisor: John-Morten Godhavn
May 2021

NTNU
Kunnskap for en bedre verden

Eivind Sjøvold

Autonomous Drilling Using Reinforcement Learning

Master's thesis in Industrial Cybernetics
Supervisor: Ole Morten Aamo
Co-supervisor: John-Morten Godhavn
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU

Kunnskap for en bedre verden

Abstract

The process of drilling wells require substantial investment. Optimization of operational parameters to maximize rate of penetration(ROP) is therefore a natural topic of discussion. Optimization on a static ROP model in advance of operations has limited application, as modeling of drilling rate is challenging. The phenomena that affect it is not fully understood. The model parameters are location and condition specific, and historical data therefore has limited application. Real-time optimization of ROP is emerging as a feasible solution, with the development of measurement techniques and computational resources. There exists data-driven model-free approaches that optimize drilling rate in real-time. An example of this is minimization of the mechanical specific energy concept with the extremum seeking algorithm. This thesis investigates whether model-free deep reinforcement learning algorithms can act as real-time optimization algorithms for ROP.

The reinforcement learning framework is flexible, with a range of varying estimation techniques and solution algorithms. The A2C algorithm is a model-free, on-policy, deep reinforcement learning algorithm that utilize parallel instances to efficiently explore the state-space. It is an actor-critc method, which utilizes artificial neural networks to maintain estimates of both the parameterized policy, and value function. The algorithm uses the advantage function to evaluate the policy update. The design of the artificial neural networks is of importance for efficient learning.

In this project, four environment with increasing complexity is implemented to evaluate different aspects of applicability of reinforcement learning in real-time optimization. The reinforcement learning agent manipulates the input by choosing a direction to adjust it at each iteration to maximize output ROP. The reinforcement learning agents are trained on one configuration of the environments, and evaluated on unseen model configurations. The ROP model in environment 1 and 2 are simple parabolic functions, with one and three inputs respectively. The models are convex for all parameter configurations, and the reinforcement learning agents generalize well, so no real-time learning is required.

Environments 3 and 4 are based on Eckel's ROP model and Bourgoyne and Young's ROP model. The agents generalize poorly to unseen model configurations, but when real-time

learning is introduced, the agents maximizes ROP through manipulating the input. The agents handle sudden changes in model parameters. This mimics formation changes while drilling. The agents also handle parameters that vary with depth.

As the models are simplifications of realistic drilling systems, one cannot definitely conclude that reinforcement learning with real-time learning is an efficient solution to solve the drilling optimization problem. The agents did handle tests within the bounds of the models, and can be a promising method for drilling optimization. An interesting extension to this project would be to utilize realistic drilling simulators to generate ROP as a measurement, and have the RL algorithm minimize the mechanical specific energy concept. Another interesting extension would be to generate models from drilling data, and analyze performance on unseen data.

Sammendrag

Prosessen bak brønnboring krever store investeringer. Optimalisering av operasjonelle parametere med den henskikt å maksimere borerate(eng:rate of penetration (ROP)) er derfor et naturlig diskusjonstema. Optimalisering av en statisk modell i forkant av boreoperasjoner har begrenset bruksområde, da modellering av boreprossesen er utfordrende. Fenomenene som påvirker boreraten er ikke eksakt forstått. Modellparameterene er steds- og tilstandsspesifikke, og historisk boredata har derfor begrenset bruksområde. Sanntidsoptimalisering av ROP fremstår som en mulig løsning, da måleteknologi og beregningskapasitet utvikler seg. Det eksisterer datadrevne, modellfrie tilnærminger som optimaliserer ROP i sanntid, gjennom blant annet minimering av "mechanical specific energy(MSE)" med "extremum-seeking(ES)"-algoritmen. Dette prosjektet undersøker om modellfri, dyp forsterkende læring(eng:reinforcement learning(RL)) kan brukes til sanntidsoptimalisering av ROP.

RL-rammeverket er fleksibelt, med forskjellige estimeringsteknikker og løsningsalgoritmer. A2C-algoritmen bruker parallelle instanser for å utforske tilstandsrommet på en effektiv måte. Det er en "actor-critic" metode, som bruker dype nevrale nettverk(ANNs) til å estimere verdifunksjon og "policy". Algoritmen bruker "advantage function" for å evaluere policyoppdateringen. ANN-strukturen er viktig for effektiv læring.

I dette prosjektet er fire miljø med økende kompleksitet implementert for å evaluere forskjellige aspekter av RL i sanntidsoptimering. RL-agenten justerer pådrag ved å iterativt velge en retning å justere pådraget i for å maksimere ROP. RL-agentene er trent på en modellkonfigurasjon i miljøet, og evaluert på en annen konfigurasjon. ROP-modellene i miljø 1 og 2 er konvekse, paraboliske funksjoner. Her generaliserer agenten godt, og sanntidslæring er ikke nødvendig.

Miljø 3 og 4 er basert på Eckels ROP-modell og Bourgoyne og Youngs ROP-modell. Agentene generaliserer dårlig på usett data. Dette løses ved sanntidslæring. Da maksimerer agentene ROP. Agentene håndterer stegvise og gradvise parameterendringer, som er viktig i optimalisering av ROP.

Modellene er forenklinger av faktiske boresystemer. På grunn av dette kan man ikke definitivt konkludere med at RL med sanntidslæring er en effektiv løsning på problemet.

Agentene var dog robuste innenfor rammeverket av testing, og begrensningene i modellene, og kan være en lovende løsning på boreoptimalisering. En interessant videreføring av dette prosjektet er å teste en RL-algoritme med mer nøyaktige simuleringer eller faktisk boredata.

Preface

This thesis is the delivery for TTK4900 - Engineering Cybernetics. It is the final project of a two year master's programme in Industrial Cybernetics at NTNU, from August 2019 until May 2021. The project was conducted from January through May 2021. The programme itself has been challenging, with a steep learning curve. It has given me the opportunity to learn topics in an interesting field of science, with excellent facilities for learning along the way. I chose this project as it gave me an opportunity to investigate the exciting reinforcement learning paradigm, and apply the knowledge I acquired towards investigation of potential industrial applications.

The project has been challenging at times, as I had no code, data, models or previous work to base my project on. Producing models, simulations and code has been time consuming work. I would like to thank my two supervisors, Ole Morten Aamo(NTNU) and John-Morten Godhavn(Equinor). They have throughout this project provided me with the insight necessary to progress in the work, and encouraged me in slower periods. Additionally I would like to thank former employers who have given me unique opportunities to learn and develop as a person. A special thanks goes out to my colleague Ludvig G. Tronsaune.

Contents

Nomenclature

A2C Advantage Actor Critic

A3C Asynchronous Advantage Actor Critic

ANN Artificial Neural Network

BHA Bottom Hole Assembly

DDPG Deep Deterministic Policy Gradient

DOC Depth of Cut

DQN Deep Q-Networks

DSE Drilling Specific Energy

HFTO High-Frequency Torsional Oscillations

HMSE Hydromechanical Specific Energy

ML Machine Learning

MSE Mechanical Specific Energy

NN Neural Network

RL Reinforcement Learning

RMSprop Root Mean Square Prop

ROP Rate Of Penetration[ft/hr]

RPM Revolutions Per Minute[rev/min]

SGD Stochastic Gradient Descent

SL Supervised Learning

WOB Weight On Bit[klbf]

List of Tables

List of Figures

1 Introduction

Optimization of operational parameters to maximize rate of penetration during well drilling is a topic of interest as there are large economic costs tied to the operation. An autodriller is an algorithm that autonomously adjusts these operational parameters to maximize rate of penetration. This project investigates whether a reinforcement learning algorithm can act as an autodriller, and optimize rate of penetration.

1.1 Problem Description

The project is split into the following goals:

- Propose a simple simulation model for rate of penetration, where the autodriller can adjust the operational parameters weight on bit, flow, revolutions per minute.
  - Conduct a literature search on rate of penetration modeling and rate of penetration optimization, and propose a model.
- Make a simulation case with varying rock properties, and constraints in input and pressure.
  - Implement environments that make the basis of simulation and training of the reinforcement learning algorithm, and act as validation cases.
- Implement a reinforcement learning autodriller that adjusts the operational parameters to optimize ROP in the implemented simulation cases.
  - Conduct a literature search on reinforcement learning.
  - Identify a suitable algorithm.
  - Experiment with a reinforcement learning agent as an autodriller in validation cases.

1.2 Software

This section describes all software used in this thesis. All code is written in the Python programming language, and some third party libraries have been used for implementation.

1.2.1 Stable Baselines

Stable Baselines is a fork from OpenAI Baselines. OpenAI is a company that researches AI, and develops AI algorithms. They have published a reinforcement learning python library called OpenAI Baselines. Stable Baselines is a library based on further development of OpenAI Baselines. The Stable Baselines library have a selection of RL algorithms that are off the shelf applicable, and is well documented[1]. In this project, Stable Baselines 3 version 1.0 is used.

1.2.2 PyTorch

PyTorch is an open source machine learning framework, that features in the stable baselines implementation. In this project, the multiprocessing aspect of PyTorch is utilized, in addition to artificial neural network functionalities for building networks, selecting activation functions and hyper parameters. PyTorch version 1.7.1 is used.

1.2.3 Gym

Gym is a toolkit developed by OpenAI. It acts as a standardization of the environment structure in the reinforcement learning process. All environments implemented in this project follow the Gym interface. This is further described in section 4.1. Gym version 0.18.0 is used.

1.3 Limitations

The work done in this thesis is not based on any previous projects. The work conducted in this project was without access to any data-sets. In addition to this, no real-world drilling simulators were available. The ROP models applied in this thesis is found in literature, and has no connection to pressure. As a consequence of this, the pressure constraints has been neglected in this work.

1.4 Outline of Thesis

- chapter 2 introduces the rotary drilling process briefly. Typical phenomena that affects drilling rate, and rate of penetration models are outlined. In addition some relevant research is reviewed.
- chapter 3 outlines the reinforcement learning problem, important terminology, and estimation methods and solution algorithms to the reinforcement learning. Deep reinforcement learning is also introduced, along with the most prominent model-free deep reinforcement learning algorithms
- chapter 4 describes the implementation of four different environments with increasing complexity, and the deep reinforcement learning algorithm that are developed and implemented to act as autodrillers.
- chapter 5 outline the results based on the implementation described in chapter 4. The reinforcement learning algorithm is analyzed as a predictor and continual learner on different model configurations of the four environments. Simulations that mimic drilling through multiple rock formations and single formations are presented.
- In chapter 6, the solution method, simplifications and algorithm implementation is further discussed.

2 | The Rotary Drilling Process

This chapter presents theory related to the drilling process utilized in this thesis. The chapter is structured in the following way:

- chapter 2 presents the rotary drilling process briefly. Some of the phenomena and equipment appearing the process is presented.
- section 2.1 outlines drilling optimization, why it is a topic of interest and why it is a significant challenge to efficiently optimize the drilling process.
- section 2.2 presents some analytical models of rate of penetration with varying complexity and applicability. Section 2.2.1 describes some of the most important phenomena that affects the drilling rate when drilling in rock formations.
- section 2.3 presents important research that has been conducted on modelling and optimization of rate of penetration. The concept of specific energy is also presented in section 2.3.2.

The process of drilling for oil and gas require substantial investments. As a consequence of this, only large oil and gas companies have the financials to make the investment[2]. This section will briefly present the rotary drilling process, and some of the equipment involved. In section 2.2, a selection of analytical Rate of Penetration models will be presented, and lay the foundation for the mathematical models used in this thesis.

Rotary drilling rigs are used in most drilling performed. The processes are similar across different projects, and usually consist of a rotating bit at the end of a drillstring. A downward force is applied to the drilling bit from sections of pipe called drill collars. The drillstring is typically rotated from the surface, which is called topside. The cuttings generated from drilling in the rock formation needs to be removed. This is done through circulating a fluid, called the drilling fluid, down the drillstring. This lifts the cuttings to the surface. Here, the cuttings are separated from the drilling fluid, which is continually reused.

2.1 Drilling Optimization

Drilling is an expensive process. Drilling optimization is therefore a natural topic of discussion, as there are large economic costs to be reduced, which increases the overall profitability of the operation. There are two main aspects to drilling optimization. The first is designing and selecting drilling equipment for a given well structure[3]. The second is selecting operational parameters to increase the drilling rate itself. Over the last decades, researchers and engineers have made a substantial effort to optimize the drilling rate parameters[4], as the drilling rate is directly related to the time spent actually drilling, and therefore the overall cost.

One of the largest challenges in drilling rate, or rate of penetration(ROP), optimization is formulating a sufficient ROP model. Many papers have been published on this topic. The variables that affect ROP are not fully understood, and is therefore difficult to model[2]. As a consequence of this, no precise mathematical and dynamical model exists that is sufficient[5]. Several approaches has been tried to fill this gap. Analytical and semi-analytical models that combine some of the known phenomena that affects ROP has been combined with formation and drilling specific parameters. These typically have to be determined for specific formations and conditions. This can be done through historical data of drilling in similar formations, or through experimental drilling to collect the necessary data. In the former approach, sufficient model precision is rarely met, and in the latter approach, the cost often outweighs the benefit. Data driven models have also been tried to predict ROP. However, no model that is off-the-shelf applicable for ROP modeling is published yet.

Although many phenomena and parameters have an impact on the ROP, only a handful are controllable for the drilling engineer[5]. These are typically weight on bit(WOB), which is the force applied to the drilling bit, revolutions per minute(RPM), torque and hydraulics. Although bit type and circulation fluid choice also affects the ROP, these are not considered controllable parameters in real-time optimization of ROP[5]. Hardness of the formation, and other formation specific parameters are uncontrollable in the drilling process.

2.2 Rate of Penetration Models

There exists a large number of models of rate of penetration(ROP) that have been developed from the 1950's until present day. Typical models relate weight on bit(WOB), revolution per minute(RPM) and hydraulics to ROP. The models typically feature formation specific constants. Maurer[6] published a paper in 1962 where he collected previous empirical ROP models and argued that poor bottom hole cleaning was the reason for

the discrepancy in the results of the different models. He further argued that bottom hole cleaning is formation dependent, and cannot be represented by an exact, applicable model. Bingham[7] published a simple model in 1964 relating ROP to RPM, WOB and diameter of the bit(d_b).

$$ROP = a\left(\frac{WOB}{d_b}\right)^b RPM \tag{2.1}$$

Binghams ROP model(Equation 2.1) was the first relationship of ROP-RPM-WOB that featured a formation specific WOB exponent, making the model generally more applicable. Since this, many analytical ROP models have followed, and some will be presented in this chapter.

2.2.1 Drilling Rate Behaviour

The effect of WOB and RPM on ROP has been studied by several authors[2]. Typically, WOB has no effect on ROP until some threshold value[2]. ROP then increases with increasing WOB up to a certain value, after which the drilling rate either stalls or decreases. This threshold is commonly referred to as *founders point*[8]. The effect of RPM on ROP is typically linear for lower values of RPM, and the effect drops off at a certain value. Figure 2.1 and Figure 2.2 highlights this behaviour.

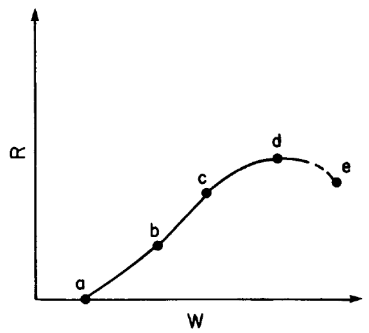


Figure 2.1: Bourgoyne and Young's illustration of the ROP-WOB relationship. Illustration taken from [2].

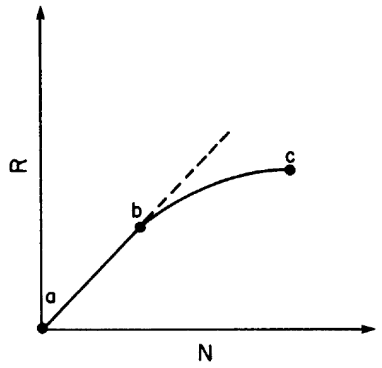


Figure 2.2: Bourgoyne and Young's illustration of the ROP-RPM relationship. Illustration taken from[2].

The reason for occurrence of founders point is the presence of a dysfunction in the drilling process[8]. Common drilling dysfunctions include vibrations, bit- or bottomhole balling and bit dulling. Bottomhole balling is the accumulation of rock cuttings at the bottom of the drilling hole, such that the bits interaction with the rock formation is interfered[8]. This can be caused by improper cleaning conditions. This typically occurs at hard formations. Bit balling is a cause of rock formations absorbing fluids and sticking to the

drilling bit[9]. This typically occurs on softer formations. Vibrations are common at all rock formations with non-optimal operational parameters. Dupriest et al.[8] presented a drilling curve similar to that of Bourgoyne and Young(Figure 2.1), with more detail. It is split into three regions, with different ROP-WOB relationships. Region I has a sub-optimal drilling efficiency because of the low depth of cut(DOC), meaning the bit is not cutting rock at its full capacity[8].Region II occurs when the WOB is sufficient to give an optimal DOC. The ROP-WOB relationship is close to linear. Dupriest argued that there is no environmental changes that can be made to improve ROP in this region, outside of changing the operational parameters. Region III starts at the occurrence of founders point, after which a dysfunction in the drilling process occurs. Dupriest's drilling curve is shown in Figure 2.3.

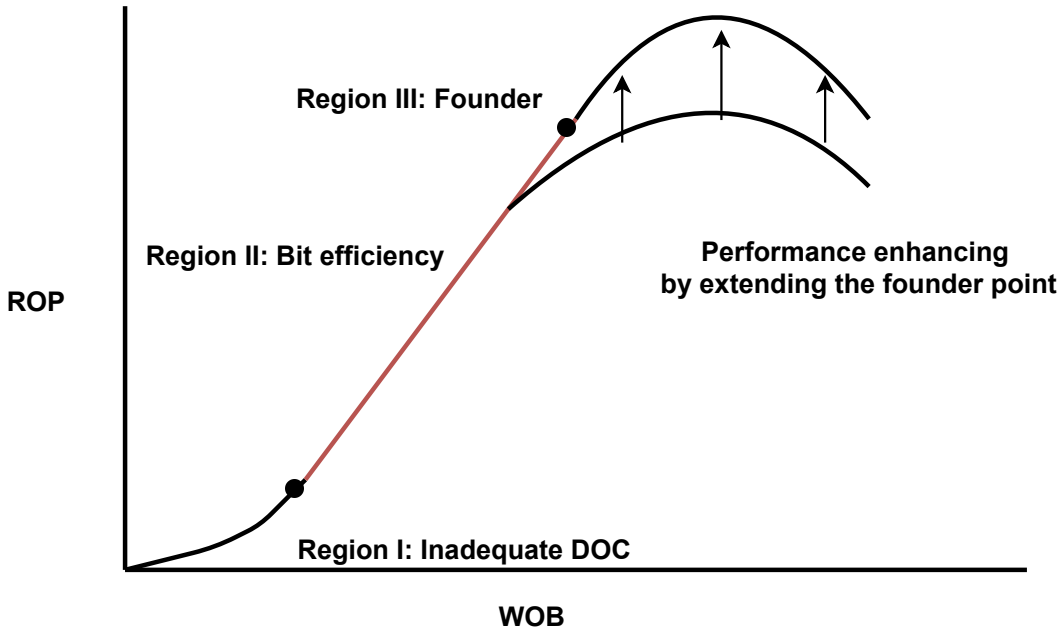


Figure 2.3: Illustration inspired by Dupriest's drilling curve, highlighting the three regions[8].

2.2.2 Eckel's model

Eckel published paper in 1967 where he investigated the effect of hydraulics, specifically fluid properties, on ROP[10]. Different drilling fluids were tested, and a relationship between ROP, WOB, RPM and hydraulics and fluid properties were developed.

$$ROP = KW^a N^b \left(\frac{kq\rho}{d\mu}\right)^c, 2 < \frac{kq\rho}{d\mu} < 100 \tag{2.2}$$

K, a, b are considered constant for a given formation, and c is constant within the given constraints. Eckel stated the model was developed for calculating ROP and field mud treating[10]. The usefulness of the model depends on the ability to determine K, k, a, b, c

from experimentation or experimental data. ρ is fluid density, and μ viscosity. In this model, W is $WOB[klbf]$, N is $RPM[rev/min]$, and q is flow[gal/min]. This gives output $ROP[ft/hr]$.

2.2.3 Bourguyne and Young's model

Bourgouyne and Young's model is a popular ROP model as it is one of the most comprehensive analytical models that have been developed[3]. It relates ROP to eight different phenomena.

$$ROP = f_1 \cdot f_2 \cdot f_3 \cdot f_4 \cdot f_5 \cdot f_6 \cdot f_7 \cdot f_8 \tag{2.3a}$$

$$f_1 = e^{2.303a_1} \tag{2.3b}$$

$$f_2 = e^{2.303a_1(1000-D)} \tag{2.3c}$$

$$f_3 = e^{2.303a_3 D^{0.69}(g_p-9)} \tag{2.3d}$$

$$f_4 = e^{2.303a_4 D(g_p-\rho_c)} \tag{2.3e}$$

$$f_5 = \left(\frac{\frac{W}{d_b} - \frac{W_t}{d_b}}{4 - \frac{w_t}{d_b}} \right)^{a_5} \tag{2.3f}$$

$$f_6 = \left(\frac{N}{60} \right)^{a_6} \tag{2.3g}$$

$$f_7 = e^{-a7h} \tag{2.3h}$$

$$f_8 = \left(\frac{F_j}{1000} \right)^{a_8} \tag{2.3i}$$

$$F_j = K \cdot \rho \cdot q \cdot v \tag{2.3j}$$

The different equations describe the effect of phenomena encountered while drilling. Table 2.1 describe what the different equations model[2]. ROP has units ft/hr. W is $WOB[klbf]$, N is $RPM[rev/min]$, and $q[gal/min]$.

Function	Models
f_1	Effect of formation strength and bit selection
f_2	Formation strength increase through normal compaction with increasing depth
f_3	Effect of abnormal pressure(undercompaction)
f_4	Effect of overbalance
f_5	Effect of WOB on ROP
f_6	Effect of RPM on ROP
f_7	Effect of bit wear
f_8	Effect of bit hydraulics

Table 2.1: Description of phenomena of the BY model.

The constants a_{1-8} are formation and condition specific, and has to be uniquely defined for a given formation. The constants typically lie within a range[3], given by Table 2.2. The precision of the model is decided by the potential of defining the constants.

Constant	Lower Bound	Upper Bound
a_1	0.5	1.9
a_2	0.000001	0.0005
a_3	0.000001	0.0009
a_4	0.000001	0.0001
a_5	0.5	2
a_6	0.4	1
a_7	0.3	1.5
a_8	0.3	0.6

Table 2.2: Typical parameter range of BY model coefficients.

2.3 Existing Research

Precise ROP prediction has been topic for several research papers. Some rely on modeling phenomena that has an impact on ROP, and combining these with formation specific constants. Other approaches are data-driven, and utilize machine learning techniques to forecast ROP, based on the data from previously drilled wells. This section will present some attempts at modelling ROP, and some of the research in ROP optimization.

2.3.1 Rate of Penetration Modeling Attempts

As mentioned in section 2.2, Bingham published one of the first models that combined laboratory experiments with models of the parameters that affect ROP. After this, many similar models have been derived, some adding more phenomena to the ROP model. Hareland and Rampersad published a model in 1994 that focused on rock interaction, lithology coefficients and bit wear[11]. The model, given by Equation 2.4, applies conversion of mass to describe the penetration of individual cutters of the bit[3]. The factor

A_v describes the compressed area of rock in front of the individual cutters, and can be modified to model any rotating bit utilizing circular motions[11].

$$ROP = \frac{a}{\left(RPM^b WOB^c\right)} \frac{14.14 N_c RPM}{db} \cos\alpha \sin\theta A_v \tag{2.4a}$$

$$A_v = \left[\left(\frac{d_c}{2}\right)^2 \cos^{-1}\left(1 - \frac{4W_{mech}}{\pi \cos\theta d_c^2 \sigma_c}\right) - \left(\frac{2W_{\mathrm{mec}\,h}}{\pi \cos\theta \sigma_{\mathrm{c}}} - \frac{4W_{\mathrm{mech}}^2}{\left(\pi \cos\theta d_c \sigma_{\mathrm{c}}\right)^2}\right)^{0.5} \left(\frac{d_{\mathrm{C}}}{2} - \frac{W_{\mathrm{mec}}}{\pi \cos\theta d_{\mathrm{C}} \sigma_{\mathrm{C}}}\right) \right] \tag{2.4b}$$

Hareland et al. suggested that bit wear reduced the contact area A_v as the mechanical weight(W_{mech}), which is WOB, increased. The model uses the unaxial compressive strength of the formation, σ_c to model the rock hardness. This is a constant that is known if the rock formations are known[11]. Because of this, Hareland et al. concluded that the model could be used in advance to optimize the drilling bit utilized in the drilling process. In addition, the model could be used to optimize the operational parameters to reduce drilling time[11].

In 2010, Motahhari et al. published a paper that looked into a new drilling optimization procedure, where the drilling procedure includes positive displacement motors(PDMs) and poly-crystalline diamond compact(PDC) bits. In the paper, a new ROP forecasting model was developed. The model was based on Harelands approach[12], described above. The model assumes perfect bit cleaning conditions[12], and does not explicitly feature any hydraulics.

$$ROP = W_f \left(\frac{G \cdot \mathrm{RPM_t}^{\gamma} \mathrm{WOB}^{\alpha}}{d_b \cdot S}\right) \tag{2.5}$$

The model, as seen in Equation 2.5, has a term that models bit wear, W_f. G represents a bit geometry coefficient. Motahhari et al. used two examples to verify the applications of their model. One example was selecting the optimal PDM out of a selection of three motors. The other example was optimization of operational parameters in advance of drilling. The results were verified with comparison of data from a well drilled in Alberta, US[12].

The previously mentioned models are analytical or semi-analytical. Another approach for modeling ROP is utilizing data-driven models, based on linear regression or machine learning algorithms. When data and real-time measurements became more readily available, the research in these types of modeling attempts became more common. Bourgoyne and Young, the formulators behind the model described in section 2.2.3 published a paper in 1974 that aimed to determine the constants a_1 to a_8 with multiple regression analysis[13]. The data used was based on previously drilled wells and drill-off tests. In the later years, use of more complex machine learning techniques have been widely applied

in an attempt to accuratly model ROP. The most common has been applications of neural networks(NNs), trained on data from previously drilled wells, used to predict drilling rates from a known or semi-known formation. Bilgesu et al.[14] published one of the first papers where ROP prediction through NNs were applied. They trained neural networks with different compositions, and different input parameters. The results were compared to drilling data from previously drilled wells, and trained on data from a drilling simulator[14]. The predictions of ROP were comparable to the drilling data, but it was concluded that the results were valid withing the bounds of data used in development of the networks[14]. For data outside the bounds, new networks had to be developed.

In the 2010s and early 2020s, multiple papers investigating artificial neural networks(ANNs, section 3.8.1) to predict ROP has been published. Moran et al. used ANNs to predict ROP based on existing well data. They concluded that ANNs can be programmed to extrapolate data between wells, and estimate ROP[15]. Several authors have investigated similar problems[16, 17, 18]. Batanee et al. used ANNs to correctly relate decreasing ROP to increasing depth, and lower drilling fluid density to increasing ROP[19]. Esmaeili et al. used ANNs and the data from a mini-scale laboratory drilling rig to predict ROP. The drilling rig also included vibration sensors, and the data was used for training. They compared ANN models using data both with and without vibrations. They concluded that introducing vibrations clearly increased the precision of the model[20]. Shi et al. published a paper that focused on efficient real-time prediction of ROP in offshore drilling. They concluded that the input parameter selection for the ANNs were non-trivial, and that efficient parameter selection might lead to a wider application of the models[21].

Other machine learning algorithms have also been modified in an attempt to model ROP. Mantha and Simon attempted to model ROP using the Random Forests(RF)[22] algorithm. They also analysed the performance of k-nearest neighbour(KNN) and support vector regression(SVR). They concluded that RF yielded the lowest error of the algorithms, but that the others could be used if there were constraints in the parameter selection[23].

There have also been published comparisons between analytical models and data-driven models. Soares et al. published a comprehensive comparison between several analytical models to data-driven models based on machine learning algorithms used in previous ROP prediction papers. The analytical models analyzed were the models of Bingham(section 2.2), a simplified Bourgoyne and Young model(section 2.2.3), Hareland and Rampersad, and Motahhari(section 2.3.1). The data-driven models were random forests, support vector machines and artificial neural networks. The data-driven machine learning models were trained on data from parameters that feature in the analytical models. Soares et al. found that the machine learning models on average had an error that was 20% lower than the analytical models[3]. The Bourgoyne and Young model performed

best of the analytical models, and the RF algorithm had the lowest overall error[3].

2.3.2 Specific Energy

Mechanical specific energy(MSE) was a concept introduced by Teale in 1965. He specified that drilling in rock formations was the breakage of fragments out of a face of a solid wall of rock[24]. As the drilling was a case of breaking, instead of cutting small fragments, he argued that the energy/volume relationship was of importance. He defined specific energy as the energy required to excavate unit volume of rock[24]. He divided the work exerted on the rock formation into two parts, thrusting and rotary, as seen in Equation 2.6b. The thrusting makes an indentation of the bit into the rock formation, and the rotary work breaks the rock fragments from the formation[24].

$$MSE = \frac{Input\ Energy}{Output\ ROP} \tag{2.6a}$$

$$MSE = \left(\frac{g \cdot WOB}{A}\right) + \left(\frac{2 \cdot \pi}{A}\right)\left(\frac{RPM \cdot T}{ROP}\right) \tag{2.6b}$$

MSE can be viewed as an efficiency measurement of "work in" versus "volume(of rock) out"[25]. Dupriest viewed it as a quantification of the relationship between input energy and ROP[8]. It is argued that the minimum MSE required to drill in rock formations at atmospheric pressure is numerically close to the unaxial compressive strength(UCS) of the rock[26]. This value should be constant for a given rock formation, meaning a given amount of energy is required to break a specific rock. This leads to a close correlation between drilling efficiency and MSE, and hence ROP. The drilling curve presented by Dupriest described in subsection 2.2.1 visualizes this relationship. In region II, drilling is optimal, meaning the energy put into the system is utilized in the rock drilling. However, when the point of founder occurs, a drilling dysfunctions enter the system. This means a portion of the input energy is lost to other phenomena, and MSE increases as ROP decreases[8]. This logic is the base of several ROP optimization studies which revolve around minimizing MSE. Some of these studies will be mentioned in section 2.3.3

Dupriest et al. proposed a system for real-time surveillance of drilling efficiency based on the MSE concept in 2005. The proposed system accurately detected drilling dysfunctions[25]. It could also identify and correct some issues, mainly bit balling[5]. The system was developed for Exxon, and was implemented on most of the company's drilling rigs within a year. The system contained an adjusted MSE model, given by Equation 2.7.

$$MSE_{adj} = MSE \cdot EFF_m \tag{2.7}$$

The mechanical efficiency factor EFF_m is introduced to increase the applicability of the

model[8]. This is done as the theoretical relationship of minimum MSE equalling the UCS of the rock is impossible in reality[8]. This is because an energy loss due to friction of the drillstring is present in all operations[8]. EFF_m adjusts for this. This can however induce some confusion when comparing unadjusted MSE values to adjusted ones.

Teale's MSE concept has later been modified to include the hydraulic component of the drilling process[27]. Drilling specific energy(DSE) describes the amount of energy to excavate rock, and remove it from underneath the bit[28]. DSE, as given by Equation 2.8, contains an extra term compared to MSE, that models the hydraulic energy exerted at the bit.

$$DSE = MSE - \frac{1980000 \cdot \lambda \cdot HP_b}{A \cdot ROP} \tag{2.8}$$

1980000 is a conversion factor and λ is a dimensionless, bit-specific constant, and HP_b is the hydralic horsepower.

Kshitij et al. introduced a specific energy model in 2015 that also included an hydraulic component[29]. They argued that MSE, which was designed for rock mining, was insufficient for modeling the energy relationships in complex wells[29]. The model, as seen in Equation 2.9, called hydromechanical specific energy(HMSE) includes the energy exerted from the jet force onto the rock formation.

$$\mathrm{HMSE} = \frac{\mathrm{WOB_e}}{A_\mathrm{b}} + \frac{120\pi \mathrm{NT} + 1154\eta \Delta P_\mathrm{b} Q}{A_\mathrm{b} \mathrm{ROP}} \tag{2.9}$$

In this model, WOB_e is the exerted WOB on the rock formation, as described in Equation 2.10, where F_j is the jet force from the nozzle and η is a factor for energy reduction[29]

$$\mathrm{WOB_e} = \mathrm{WOB} - \eta F_j \tag{2.10}$$

2.3.3 Rate of Penetration Optimization

Selection of operational parameters for the drilling process, with goal of increasing ROP has been a topic in several research papers. There have been two main pathways. One is based on pre-computations on a static ROP model, and optimizing operational parameters in advance. Another pathway is based on real-time optimization of operational parameters based on measurements available while drilling.

One of the most important, early attempts at ROP optimization on a static models were conducted by Bourgoyne and Young[5]. The study, mentioned in section 2.3.1, aimed to optimize operational parameters through multiple regression analysis, from data collected from a minimum of 25 wells. They proposed a linear model, and concluded that drilling rate could be improved as much as 10% through a relatively simple optimization objective function[13].

Galle and Woods proposed a procedure for selecting the best constant WOB and RPM based in relations to drilling cost[30]. The proposed procedure consisted of graphs used by drilling engineers to select constant values based on drill-off tests. The solution took bit-dullness and bit-hours into account, as one of the first optimization studies. Bit-dullness and bit-hours is often called bit wear in later studies.

In 1969, Young developed an on-site drilling control computer system[31]. The system was supposed to control WOB and RPM to reach minimal cost drilling. The model utilized in the calculations was a very simplified ROP model, and models for bit wear dependant on RPM. Lastly a model of drilling cost given by the above mentioned effects was formulated.

Several authors have analyzed static ROP models and the results vary, and applicability is constrained to the accuracy of the models. Eren analyzed several similar attempts in his PhD dissertation in 2010[5]. Eren also proposed a method for optimizing operational parameters based on the Bourgoyne and Young ROP model(section 2.2.3). A multiple regression technique was used to optimize the parameters with respect to minimum drilling cost based on data-sets from wells drilled in Mediterranean Offshore[5]. It was concluded that the coefficients used in the model, based on drilling form different locations, were specific to those formations, and new coefficient needed to be found at new locations, similar to the conclusion of Bourgoyne and Young[2, 5]. It was assumed that the drilling cost in the specific drilling case Eren analyzed would have been reduced by 22%.

The specific energy concept described in section 2.3.2 is frequently used in more recent optimization studies based on analysis of real-time data. The advantages of specific energy is that it relies on measurements that are available for the driller, WOB, torque, RPM, ROP, hydraulics and bit size. In the previously mentioned study by Dupriest[8], a drill efficiency surveillance system was proposed and later implemented at Exxon's drilling rigs. The real time system detected drilling dysfunctions based on the energy model with high accuracy. Dupriest also discussed redesigning the constraints that defined founders point to extend and increase the second region of the drilling curve in Figure 2.3[8].

Hamrick used and modified the MSE concept to optimize operational parameters in the drilling process in his PhD dissertation in 2011[25]. The proposed optimization method found the optimal operational parameters in test datasets, and data from a laboratory test drilling rig[25]. The calculations were only valid for the datasets used.

In 2019, Abughaban et al. presented an intelligent drilling advisory system(IDAS), that evaluated DSE, DOC and and torsional vibrations[32]. The DSE model was presented as in section 2.3.2 with other conversion factors to utilize SI-units, and the torsional vibrations were modeled with a dynamic state-space model. DOC was modelled in mm/revolutions, and is given by Equation 2.12. The objective function for the optimization is

given by Equation 2.11.

$$\text{Obj}_{(ROP,DOC,DSE,SS)} = \frac{1 + \Delta ROP/ROP_{i-1} + \Delta DOC/DOC_{i-1}}{1 + \Delta DSE/DSE_{i-1} + \Delta SS/SS_{i-1}} \tag{2.11}$$

$$DOC = 16.66 \cdot \frac{ROP}{RPM} \tag{2.12}$$

The solution was calculated through a multiple regression analysis technique called optimum parameter global retrieval[32]. The direction of the optimal operational parameters were found through gradient search. The system monitors the specific energy in relationship to ROP, and detects drilling dysfunctions through this. It also detects and adjusts for formation change[32], and mitigates damaging vibrations. The system is being tested in a field pilot feedback[32].

Aarsnes et al. demonstrated the feasability of controling the hook load(the sum of all downward force) to optimize ROP in real-time drilling[33]. They utilized the extremum seeking(ES) algorithm, a model-free gradient ascent algorithm from adaptive control on a dynamical ROP model. The founders point that sometimes induces a reduction in ROP after an increase in WOB makes the curve convex around the founders point, making the method feasible[33].

In 2021 Nystad et al. published a paper that also investigated the use of the ES algorithm to optimize drilling efficiency[34]. The ES algorithm was utilized to minimize the MSE in real-time. The presented algorithm was data-driven and did not require any model. It found the operational parameters that gave minimum MSE in real-time while keeping within the given constraints[34]. The algorithms performance was analyzed through simulations. The ES method also tracked changes in optimum WOB and RPM[34]. In the simulated results, ROP improvements from 20-170% was found through this method[34].

3 | Reinforcement Learning

This chapter presents reinforcement learning(RL), algorithms and theory applied in this thesis. This section briefly explains the concept of reinforcement learning, as well as some important terminology and distinctions in reinforcement learning. The chapter is outlined in the following way:

- section 3.1 describes the reinforcement learning problem
- section 3.2 presents the concept return, which all RL algorithms uses to assess their performance
- section 3.3 presents the term policy in RL
- section 3.4 describes value functions used in RL
- section 3.5 presents the exploration vs. exploitation dilemma which is a fundamental principle in RL
- section 3.6 briefly outlines what optimality is in RL
- section 3.7 present different solution methods in RL. Different estimation techniques and algorithm types are outlined.
- section 3.8 presents deep reinforcement learning, how it differs from the overall RL concept, and algorithms that fall under the deep reinforcement learning paradigm.

Reinforcement learning is an area of machine learning. More specifically, it is the training of machine learning algorithms to make a sequence of decisions. The decisions are made by considering the reward, a numerical signal. The decision maker is called the reinforcement learning agent, or simply just agent.

The term reinforcement learning, as with the rest of machine learning, contains both a problem, a class of solutions that solves said problem, and the study of the solution solving the problem[35]. Reinforcement learning differs from other machine learning paradigms as it learns from its own actions. Terms like "trial and error" and "exploration" are important in reinforcement learning. Two of the most explored areas of machine learning are supervised- and unsupervised learning. Reinforcement learning differs these paradigms

in a fundamental way[36].

- Supervised learning is the task of learning a representation from labelled data.
- Unsupervised learning is the task of drawing inferences from a data-set without any labels.
- Reinforcement learning is the task of learning to take a sequence of decisions to maximize a cumulative reward. This learning process includes a trade off between exploring expected sub-optimal decisions to discover better decisions.

It is important to understand some of the terminologies that defines reinforcement learning[35]. The concept of a state is central in RL. A **state**(S_t) describes the current situation the agent finds itself in. The agent must be able to take **actions** that affects the state. Based on the **reward** the agent evaluates if the action was a good or bad move. The reward is a numerical signal that reflects how good the action was in regards to a predefined goal. The **environment** describes everything that is outside the agent. The agent interacts with the environment by choosing actions, given the states presented by the environment. The environment also decides the reward signal. Figure 3.1 visualizes the agent-environment interaction.

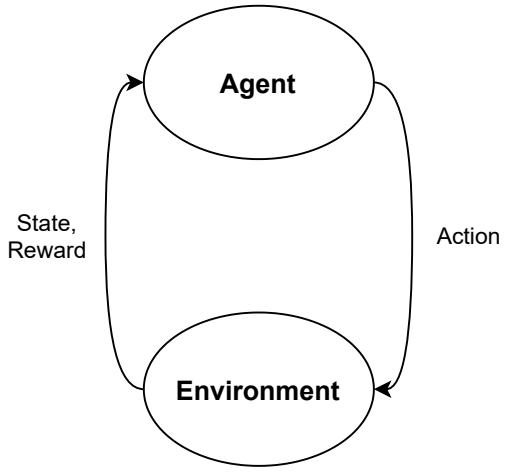


Figure 3.1: The agent-environment interaction cycle.

The **policy** of an agent is a description of its behaviour given the states it finds itself in. Simplified, the policy is the mapping of states to actions. Less formally, it can be viewed as the agents associations. A **value function** specifies the long-term desirability of states. The value function is essential in RL, as it allows for future planning[35]. States that are low-yielding with respect to reward in the short term can still be desirable, as they might enable a larger cumulative long-term reward. The value is what the agent bases the decision upon.

3.1 The Reinforcement Learning Problem

The reinforcement learning problem is the optimal control of incompletely known Markov decision processes(MDPs)[35]. MDPs describe sequential decision making, and almost all RL problems can be formalized as an MDP[35]. In an MDP, the actions taken in a given state effects not only the immediate reward of the process. It also changes the trajectory of the process through changing the subsequent states, and thereby the long term reward[35]. An MDP is defined by the five-tuple $(\mathcal{S}, \mathcal{A}, p, R, \gamma)$[36, 37].

- $\mathcal{S}$ is the space that contains all valid states s_t of the process. It is often called observation space in RL[37].
- $\mathcal{A}$ is the space that contains all valid actions that can be taken. a(s) describes all the actions that can be taken by the decision maker in state s.
- p is the transition function. It models a probability function over all states and actions[35]. It defines the dynamics of the MDP, and gives the probability of transitioning from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ when action $a \in \mathcal{A}$ is taken in s[37]. $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \Rightarrow [0, 1]$
- R defines the reward function of the process. The reward is calculated based on the desirability of ending up in state s' from s given a. It is an element in the reward space $\mathcal{R}$. That is it is a single value on the continuous space [R_{min},R_{max}][37] $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \Rightarrow \mathcal{R}$.
- γ is a discount factor that quantifies the desirability of immediate reward of states vs. long term reward of states. $\gamma \in [0, 1]$

A property of a Markov decision process is that the state s_t is immediately dependant on the previous state s_{t-1}, and not on the states prior to this. That is, all information that impacts future states must be represented in the current state[35]. This is called the Markov property. Put in simpler words, a process has the Markov property(is Markovian) if the future states only depend on the current observation[36].

Most RL problems can be designed to fit the MDP framework as it is flexible. The actions taken by the decision maker can be low-level actuator control, or high-level choices in a large system of processes[35]. The process in the RL problem does not necessarily need to be Markovian, as Markovian states can be constructed from non-Markovian states[35].

3.2 Return

The aim of the RL agent is to maximize reward over the course of the decision process[35]. This is formalized through the return, G_t. In most RL algorithms, the agent seeks to maximize the expected return over the run of the episode. The episode is a finite period of time, in which a goal, or part of a goal is achieved. The simplest possible return is defined as in Equation 3.1, where R_t is the reward in time t.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_{t_{final}} \tag{3.1}$$

Equation 3.1 poses a problem when $R_{t_{final}}$ is undefined or $t_{final} \Rightarrow \infty$. It works well for what is called episodic tasks.[35]. An episodic task has a finite ending time. A solution to infinite ending time is the discounted reward as seen in Equation 3.2.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

γ is the discount factor as described in section 3.1. This solves the issue of infinite time steps, as the series will converge for increasing k. The discount factor gives direct value to future reward, in the present time[35]. The return of successive time steps are connected through this relationship, and can be rewritten as in Equation 3.3. This relationship is called the consistency condition, and is met by returns of both finite(Equation 3.1) and infinite(Equation 3.2) time-horizons. This property forms the basis of many RL algorithms[35].

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{3.3}$$

3.3 Policy

The learning in RL is in many cases done through updating the policy, denoted π. The policy gives a probability distribution for taking action a in state s. This probability is denoted $\pi(s|a)$. The policy can be viewed as the mapping from states to actions[35]. Updating the policy is done through assessment of what states are desirable, and what actions take the process to these states. Different RL algorithms define how the policy is updated, and is one of the fundamental elements that separate the different algorithms[35].

3.4 Value Functions

Most RL algorithms involves estimation of a value function. Value functions quantify the desirability for the agent of being in given states.[35]. There are two main types of value

functions that feature in RL. One is state-value functions, and the other is action-value functions.

The state-value function, denoted $v(s)$, is defined by the expected return from state s. For this calculation to be viable, the state-value function assumes the agent follows policy π. A state value function following policy π is denoted $v_\pi(s)$, and is defined by Equation 3.4[35].

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{3.4}$$

The action value function, defined by Equation 3.5[35], gives the expected return after taking action a in state s, and thereafter following policy π.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.5}$$

The value functions also fulfill the recursive nature of the consistency condition[35]. Equation 3.6 is the bellman equation, and is one of the fundamental properties of value functions[35]. The Bellman equation describes the relationship between the current state s and all its possible successor states. It implicitly defines all values of all possible successor states from s, and gives an expected value through calculating the probability of those states occurring. The Bellman equation also describes the recursive relationship of the action-value function, as seen in Equation 3.7[35]. Here, p is the transition function, and r is the reward in time t.

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{3.6a}$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \tag{3.6b}$$

$$= \sum_a \pi(a \mid s) \sum_{\text{e}} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \tag{3.6c}$$

$$= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S} \tag{3.6d}$$

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right] \tag{3.7a}$$

$$= \Sigma_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \Sigma_{a'} \pi\left(s', a'\right) q_\pi\left(s', a'\right) \right. \tag{3.7b}$$

Another value function that features in some RL algorithms is the advantage function. The advantage function is defined as the action-value function subtracted the state-value function. This value represents how good an action is compared to all other actions

available in that current state.

$$A_\pi(s_t, a_t) = q_\pi(s_t, a_t) - v_\pi(s_t) \tag{3.8}$$

3.5 Exploration vs. Exploitation

One of the challenges in reinforcement learning is the balance between exploration and exploitation. Exploitation in RL refers to the agent taking the assumed best action in a given state to maximize the expected return. Exploration is when the agent takes expected sub-optimal actions to get better estimates of the state-space, and possibly discover better actions. An agent can not exclusively explore or exploit, and some balance needs to be found. There are different ways an RL algorithm can introduce exploration. Some algorithms adds a random bias to the agent actions, and other algorithms draws actions from a uniform probability distribution. These are independent factors that vary from algorithm to algorithm, but are present in all algorithms.

One of the more intuitive examples of exploration in RL is the ϵ-greedy policy. The policy is greedy with a rate of $1-\epsilon$. This means that the agent takes expected best actions with a rate of $1-\epsilon$, and a sub-optimal action with a rate of $\epsilon \in [0, 1]$.

3.6 Optimality

If the agent can learn a policy that takes the best actions possible in a given state trajectory, the RL problem is solved. This is an optimal policy. For finite MDPs, optimal policies can be precisely defined[35]. As the number of states and actions are finite, there exists a policy that yields a higher or equal return compared to all other policies. This is the optimal policy. The optimal policy is denoted π_*. More formally, a policy π is optimal if and only if $v_\pi \geq v_{\pi'}$ for all π'. The optimal state-value function follows the optimal policy, and gives the highest expected return, as seen in Equation 3.9[35].

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{3.9}$$

Similarly, the optimal action-value function gives the expected of taking action a in state s, and thereafter following an optimal policy(Equation 3.10).

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \tag{3.10}$$

A special case of the Bellman equations(Equation 3.6) formulated for optimal value functions is called the Bellman optimality equations. The Bellman optimality equation is

based on that the optimal choice is taking the action giving the highest expected return in state s[35].

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \tag{3.11a}$$

$$v_*(s) = \max_a \mathbb{E}_{\pi_*} \left[G_t \mid S_t = s, A_t = a \right] \tag{3.11b}$$

$$v_*(s) = \max_a \mathbb{E} \left[R_{t+1} + \gamma v_* \left(S_{t+1} \right) \mid S_t = s, A_t = a \right] \tag{3.11c}$$

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_* \left(S_{t+1}, a' \right) \mid S_t = s, A_t = a \right] \tag{3.11d}$$

In a finite MDP, the Bellman optimality equation has one unique solution[35]. If the probability dynamics of the system is known, and the value function is know, the MDP can then be solved by solving the Bellman Equations at each step, and simply taking the action with the highest return. For infinite MDPs however, the Bellman equation is a set of n equations and n unknowns, where n is the size of the state-space. In addition, the dynamics of the system is rarely precisely known[35]. The computational resources required to compute the value of all state-action pairs is also a limiting factor. As a consequence of this, feasible RL implementations are based on estimating the value functions, as opposed to model or calculate them. This implies that one typically has to settle for an approximate solution, not an optimal one[35]. The RL framework contains some efficient learning methods for estimation.

3.7 Solution Methods

This section presents solution methods and estimation techniques that feature in RL theory. Some of the sections are included for completeness, and some sections forms basis for decisions in algorithm implementation.

3.7.1 Estimation Methods

As mentioned in section 3.6, typically, an approximate solution involving estimation of value functions is the only feasible solution to an RL problem. This can be because the dynamics of the system(environment) is either unknown or only partially known. It can also be because of constraints in computational resources. There are two main methods for estimating functions in RL, being Monte Carlo methods and temporal difference methods.

3.7.1.1 Monte Carlo Methods

Monte Carlo(MC) methods do not require any model of the environment, only data from sample sequences of states, actions and corresponding reward[35]. A collective term for this data is experience. This experience can be real data, or simulated data. The term Monte Carlo methods can address any method that observe some element of randomness[38]. In RL, MC can be used to estimate optimal policies. The underlying idea behind MC methods is to update an average value associated with some state each time that state occurs. As the number of occurrences, or visits, to that state increases, the average value will converge towards the actual value for that state[35]. This value is typically the expected return(section 3.2), the value of which the RL agent assesses how good a state is.

There are more than one way to handle the averaging of returns. Examples of these are every-visit Monte Carlo estimation, and first-visit Monte Carlo estimation. First-visit MC estimation only averages the first encounter of the state in the episode, while every-visit MC averages all visits, also multiple visits in a single episode[35]. An example of the update step in state-value estimation by MC is given in Equation 3.12a[35].

$$V\left(S_t\right) \leftarrow \text{ average } \left(\text{ Returns } \left(S_t\right)\right) \tag{3.12a}$$

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha[G_t - V\left(S_t\right)] \tag{3.12b}$$

In this example, α is the step size, G_t is the experienced return and V is the value function the MC algorithm is estimating.

The fact that MC methods average returns means that the value estimate is not updated until the end of the episode. That is, the RL algorithm does a full interaction with the environment until termination, and the estimates are thereafter updated. This gives MC methods a clear disadvantage where the terminal state is not guaranteed to occur. States visited in this episode will not be used to update the value function estimate, even though an optimal state might have occurred in the episode.

3.7.1.2 Temporal Difference Learning

Temporal difference(TD) learning is a combination of MC methods and dynamic programming(An optimization technique that relies on recurring problem structures)[35]. TD can learn directly from experience like MC methods. Unlike MC methods, TD methods update the estimates without knowing the actual return value. TD methods update the estimated value functions using the estimates from successor states[35]. This is called bootstrapping. The fact that TD methods bootstrap gives a more frequent update rate.

TD methods can update the estimates at each iteration or time-step, or at another set frequency. TD methods where the estimates are updated every step are called one-step TD methods. If the estimates are updated every n steps, it is called n-step TD methods, or n-step bootstrapping for a wider term. A simple example of the update step of an estimate in one-step TD learning is given in Equation 3.13[35].

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{3.13}$$

The one-step TD algorithm makes the update on $V(S_t)$ immediately after transitioning to S_{t+1} and receiving the reward R_{t+1}. If Equation 3.12a and Equation 3.13 is compared, the fundamental difference between MC methods and TD learning becomes apparent. Equation 3.12a updates the the value function based on the entire observed return G_t, while Equation 3.13 updated the value function on a part estimate of the total return.

The part of Equation 3.13 in brackets models the error in the estimate. In RL, it is called the TD error δ. The TD error is a model of the error in the estimate at that particular time[35]. The TD-error appears in several TD learning methods[35], and is given by Equation 3.14

$$\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{3.14}$$

3.7.2 Types of Reinforcement Learning Algorithms

There are two main classes of algorithms to solve the RL problem[39]. These are model-based and model-free algorithms. These main classes have several sub-classes of algorithms. This thesis will focus on model-free algorithms.

Model-free algorithms update the value function estimate or policy representation directly from experience in the environment[39]. This is what is commonly denoted learning in RL. Model-based algorithms utilize a model of the environment to predict the environments response to the agents actions[35]. A model of the environment is anything an agent can use for this purpose, and can vary in complexity. Models can also be used to simulate experience. In extreme cases, the model could be used to simulate all possible outcomes before the agent picks an action[35]. A model based RL agent can learn a model based on experience from the environment, and use this to update value functions[39].

Similarities for model-free and model-based methods is that both rely on computation or estimation of value functions, and all methods do some sort of forecasting of a desired value(return/reward) to update an experience based estimate[35].Figure 3.2 shows the basis of model-based RL, and Figure 3.3 model-free RL.

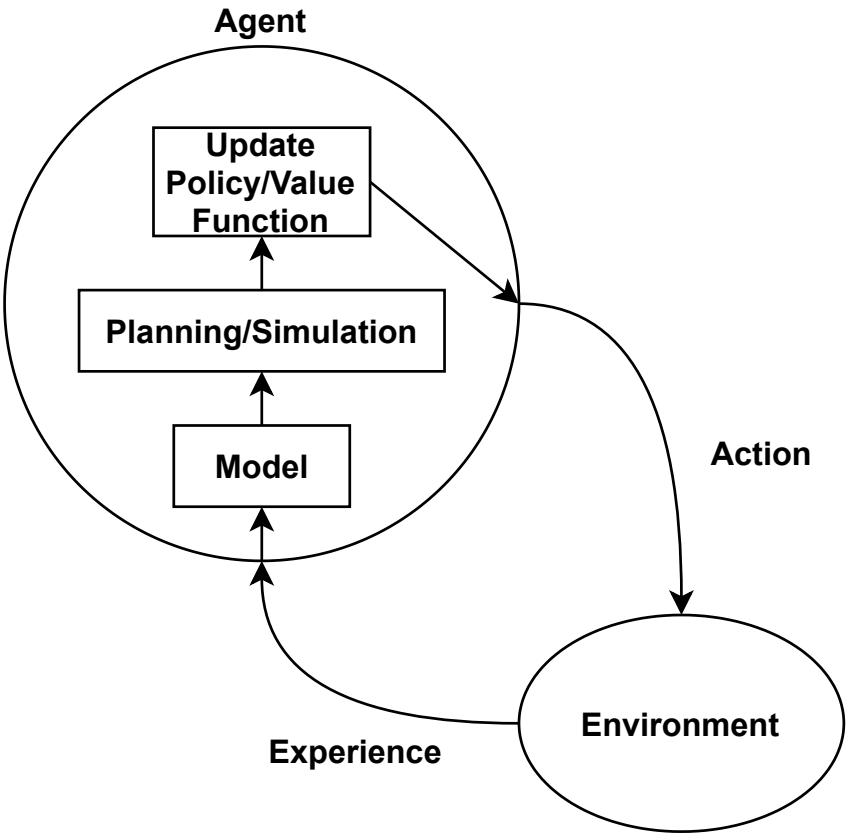


Figure 3.2: Model-based reinforcement learning.

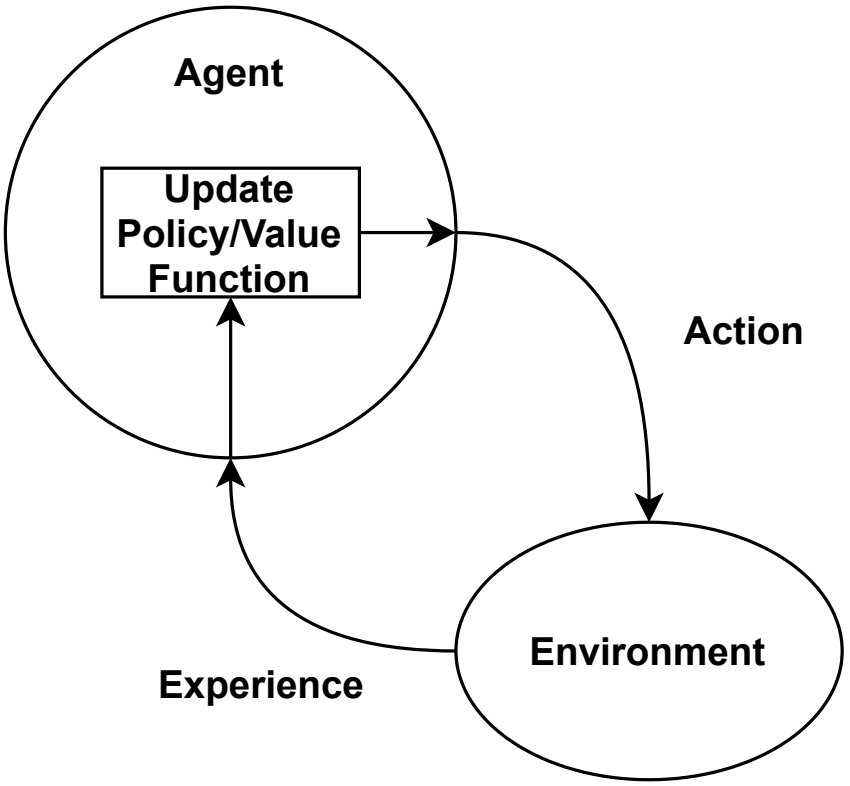


Figure 3.3: Model-free reinforcement learning.

Model-free methods can be split into value-based methods, policy search methods and actor-critic methods[40, 41]. In general, value-based methods estimate value functions, policy search methods searches the policy space to improve the policy, and actor-critic methods combine the two.

Reinforcement learning algorithms can also be split into on-policy and off-policy algorithms.

- On-policy algorithms improve the policy that is used to make the sequence of decisions that generate the experience.
- Off-policy algorithms follow another policy when it is learning. The experience from following this policy is used to update a separate policy that approaches the optimal policy.

This allows off-policy algorithms to maintain a level of exploration, while the on-policy algorithms tend to explore less as the policy approaches the optimal policy.

3.7.2.1 Value Based Methods

Value based methods learns a value function(section 3.4) that estimates the expected return. This is used to construct a policy, by choosing the actions that maximizes the value function[41]. In other words, the action value function given by Equation 3.5 forms the basis for constructing the optimal policy. To yield the highest expected return, or state value, the action corresponding to the highest action value function can be greedily chosen[40], giving the relationship in Equation 3.15. Greedy in this setting refers to the concept of simply choosing the action that seems best in the current time step.

$$v_\pi(s) = \max_a q_\pi(s, a) \tag{3.15}$$

Because of the recursive nature of the Bellman equation(Equation 3.7,Equation 3.6), the action-value function can be updated through bootstrapping.[40]. This forms the foundation for one of the more well-known reinforcement learning algorithms, single agent Q-learning[42].Equation 3.15 gives the update step of the Q-learning algorithm.

$$q_\pi\left(\mathbf{s}_t, \mathbf{a}_t\right) \leftarrow q_\pi\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha\delta \tag{3.16}$$

δ is the TD-error described in section 3.7.1.2, and α is the step size. The main trait of all value-based methods is that they maintain estimates of a value functions, similar to Equation 3.16.

3.7.2.2 Policy Search Methods

Instead of maintaining value function estimates, policy search methods directly searches for an optimal policy. Typically, this involves learning a parameterized policy, π_θ [40]. The parameter vector, θ, is updated to maximize the expected return, $\mathbb{E}[G|\boldsymbol{\theta}]$, to solve the RL problem. Policy search methods are typically more suited for high-dimensional action- and state space than value based methods[43]. The optimization of parameters is done with either gradient-free or gradient-based methods[40, 43]

Gradient-free methods improve the policy by heuristic search methods in a space of predefined policies, and has successful implementations for low-dimensional problems[40]. The biggest advantage of gradient-free methods is that they can optimize non-differentiable problems[40]

Gradient-based methods use gradients to improve a parameterized policy[40]. They update the parameter vector by using gradient ascent[35], given in general form by Equation 3.17. $J_{\boldsymbol{\theta}}$ is a scalar performance measure, and its gradient is used to update the parameter vector. In the case of policy search gradient-methods, this performance measure is based on evaluation of how good the policy is.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J\left(\boldsymbol{\theta}_t\right) \tag{3.17}$$

The policy is evaluated as in the other methods by the expected return. The expected return is still sampled from an average of possible trajectories of states and/or actions. In the model-based RL methods, these possible trajectories, or state-transition dynamics, are available, and can be accurately calculated. In the model-free RL methods, these state-transition values have to be estimated by averaging returns, like in section 3.7.1.1, or section 3.7.1.2. This leads to a challenge as gradients cannot pass through samples of stochastic functions[40]. This leads to many gradient-based policy search methods estimating these gradients.

The REINFORCE update rule is commonly used for this purpose[40]. The REINFORCE update rule is based on the policy-gradient theorem. The policy-gradient theorem, given by Equation 3.18, gives an analytic expression of the performance measure gradient that does not involve the unknown state-transition dynamics[35]. $\mu(s)$ is an on-policy distribution the agent follows[35].

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta}) \tag{3.18}$$

The policy gradient theorem can be rewritten to describe state s_t instead of the summation over all states in the state-space. The resulting equation is given by Equation 3.19[35].

The first term of the equation models the value of taking a sequence of actions following policy π, with the probability of taking said actions in state s_t. This summation gives the return from state s_t, giving the rewrite in Equation 3.19b.[1]

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a \pi \left(a \mid S_t, \boldsymbol{\theta} \right) q_{\boldsymbol{\pi}} \left(S_t, a \right) \frac{\nabla \pi \left(a \mid S_t, \boldsymbol{\theta} \right)}{\pi \left(a \mid S_t, \boldsymbol{\theta} \right)} \right] \tag{3.19a}$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi \left(A_t \mid S_t, \theta \right)}{\pi \left(A_t \mid S_t, \theta \right)} \right] \tag{3.19b}$$

This gives the REINFORCE update rule as given by Equation 3.20[35]. The policy gradient methods only maintain an estimate of the policy. This means that the policy search algorithms uses the sampled return G_t in the update step. If this return is replaced with a value function estimate, the method would be an actor-critic method.

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \nabla \log \pi \left(A_t \mid S_t, \theta \right) \tag{3.20}$$

The largest disadvantages of policy gradient methods are[40]:

- High variance, which makes convergence slower.
- Low sample utilization, leading to low data efficiency.

3.7.2.3 Actor-Critic Methods

Actor-critic methods involve both value function estimation and policy gradient methods[35, 36, 40]. The name "actor-critic" comes from the way the actor-critic algorithms behave. The algorithms have an actor, that chooses actions, and a critic that evaluates how good the action was. In this analogy, the actor is the policy, and the critic is the value function[40]. Actor-critc methods can learn from full returns or bootstrap[40].

As an example, one-step actor-critic methods replace the experienced return in the REINFORCE update rule(Equation 3.20) with a one-step temporal difference estimate, like described in section 3.7.1.2. More specifically, it uses the TD error, the error in the current value function estimate, to improve the policy parameter vector. In addition, the value function estimate is updated with one-step TD learning(Equation 3.13). The policy parameter update step is shown in Equation 3.21[35].

$$\theta_{t+1} = \theta_t + \alpha \left(R_{t+1} + \gamma v \left(s_{t+1} \right) - v \left(s_t \right) \right) \nabla \log \pi \left(A_t \mid S_t, \boldsymbol{\theta} \right) \tag{3.21a}$$

$$\theta_{t+1} = \theta_t + \alpha \delta_t \nabla \log \pi \left(A_t \mid S_t, \boldsymbol{\theta} \right) \tag{3.21b}$$

[1]Note that the rewrite $\frac{\nabla \pi \left(A_t \mid S_t, \boldsymbol{\theta} \right)}{\pi \left(A_t \mid S_t, \theta \right)} = \nabla \log \pi \left(A_t \mid S_t, \boldsymbol{\theta} \right)$ is commonly used

3.8 Deep Reinforcement Learning

Deep reinforcement learning(DRL) is a class of reinforcement learning with function approximations done by deep learning algorithms, mainly deep artificial neural networks[35]. The theory and methods that are derived previously in this section applies equally to RL and DRL, with DRL having a the additional introduction of deep learning algorithms.

Deep learning is a class of machine learning that focuses on function approximation through learning hierarchical representations of data through experience[44]. The method allows a program to build complex representations from a hierarchy of simpler representations. This gives a graph structure with several layers. This is why it is labelled deep learning[44]. All deep learning models are based on some sort of artificial neural network structure with several layers. Deep learning popularity has excelled in recent years. The emergence of large data-sets and computational power has been a contributing factor for this.

Deep learning is effective in RL as it can approximate complex function from simple parameter representations. This means that deep learning can be used to approximate value functions and policy representations with a parameter space that is significantly smaller that the number of states and actions.

3.8.1 Artificial Neural Networks

Artificial neural networks maps input from a data set, $\boldsymbol{x} \in X$, to the target $\boldsymbol{y} = f^*(\boldsymbol{x})$. The goal of an artificial neural network(ANN) is to approximate the function, $f^*(\boldsymbol{x})$[44]. This achieved by learning $\boldsymbol{\theta}$ that gives the best parameterized function $f(\boldsymbol{x}; \boldsymbol{\theta}) \approx f^*$, with parameters in $\boldsymbol{\theta}$.

ANNs consist of collections of artificial neurons, which are functions that perform specific operations[44]. These neurons apply an activation function to a linear combination of operations. In this case, $\boldsymbol{\theta}$ is made up of a set of weights, $\boldsymbol{w}$ and biases, $\boldsymbol{b}$. The output of a neuron, denoted h, is given by Equation 3.22.

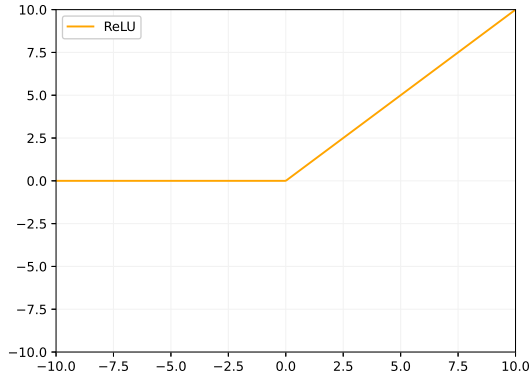
$$h = \phi(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{3.22}$$

ϕ is the activation function, which introduces the non-linearity to the function approximation[44]. Typical activation functions are rectified linear unit(ReLU)(Equation 3.23a), hyperbolic tangent(Equation 3.23b), or the sigmoid function(Equation 3.23c)[44]. The activation functions are shown in Figure 3.4

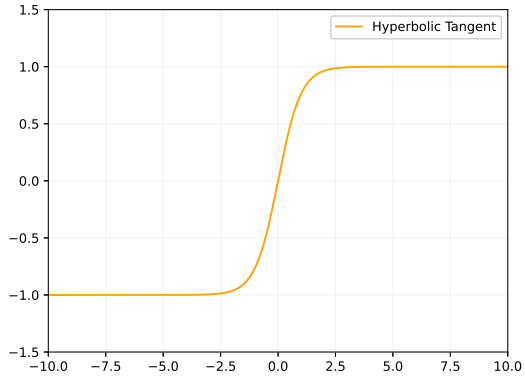
$$\phi(z) = \max(0, z) \tag{3.23a}$$

$$\phi(z) = \tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.23b}$$

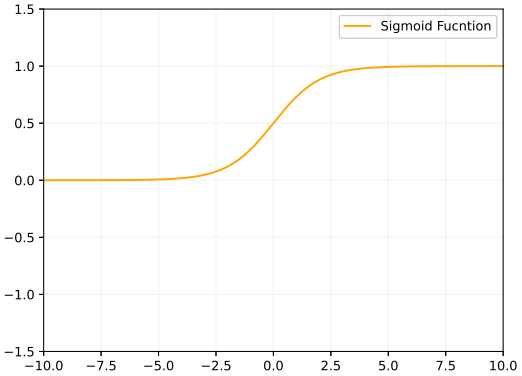
$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{3.23c}$$



(a) ReLU.



(b) Hyperbolic tangent.



(c) Sigmoid function.

Figure 3.4: Activation functions.

3.8.1.1 Feed-forward Neural Networks

Feed-forward neural networks, also called multi layered perceptors(MLPs) are a class of ANNs. The neurons are chained, and these chains form layers. The signals flows only forward through the chain, giving origin to the name[44]. The number of artificial neurons in a chain form the depth of the network. Neurons that are in the same order in the chain form a layer. The first layer is called the input layer, and contains input units which get signals from the environment. The last layer is the output layer, with output units, and gives the final output. All layers between input and output layers are called hidden layers(with hidden neurons). The number of neurons in a hidden layer is the width of the

network. The networks can be fully connected, where all neurons in a layer receive the output from all neurons in the previous layer, or not. Figure 3.5 shows a fully connected feed-forward neural network with one hidden layer. This network is three layers deep, and three neurons wide, with one hidden layer.

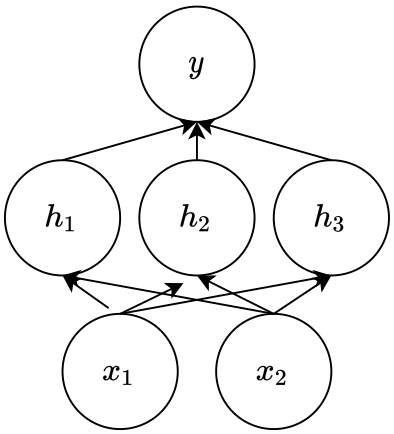


Figure 3.5: Representation of the structure of a fully connected feed-forward neural network.

3.8.1.2 Learning in Artificial Neural Networks

The learning in ANNs, also called training, can be stated as a sum of finite optimizations[44].

$$\begin{aligned} &\min_{\theta} J(\theta) \\ J(\theta) &= \frac{1}{n}\sum_{i=1}^{n} J_i(\theta) \end{aligned} \tag{3.24}$$

Learning is the process of adjusting the weights to improve the overall performance. The performance is measured by a loss function, $J(\boldsymbol{\theta})$. The loss function models the error between the output from the network, and the target value from the training data-set[44]. An intuitive loss function that has limited application in ANNs is the mean square error over the data set.

Almost all neural nets are trained with some form of gradient descent method[44]. Gradient descent methods rely on the loss function gradient to update the parameters, as in Equation 3.25. The idea behind the method is to update the parameters iteratively in the negative direction of the gradient to gradually approach the parameter set that gives the lowest overall loss function[45].α is the step size, and scales the magnitude of the update step.

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha \left.\frac{\partial J}{\partial \theta}\right|_{\boldsymbol{\theta}_k} \tag{3.25}$$

Gradient descent methods takes a step in the direction that reduces the loss function the most, improving the overall error in the ANN. The algorithm can compute all the gradients

over the entire data-set, or it can compute some gradient(s) at random. The former option is called batch gradient descent(BGD) or deterministic gradient descent, and the latter stochastic gradient descent(SGD)[44]. BDP require $O(n \cdot p)$ operations, where n is the number of data points in the data-set, and p is the number of parameters in the network. When either of these are two large, BGD is infeasible[44]. This is why SGD methods are the most popular. SGD methods make the approximation given by Equation 3.26. They make the assumption that the average over gradients can be approximated as one of the gradients chosen at random.

$$\frac{1}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\theta}}J_i(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}}J_j(\boldsymbol{\theta}) \tag{3.26}$$

Where $j \in 1, 2, ..., n$. This reduces the number of operations to $O(p)$. If the index i is chosen at random, this produces an unbiased estimate of the gradient of $J(\boldsymbol{\theta})$[44]. Algorithm 1 shows the pseudocode for SGD[44].

Algorithm 1: SGD

Require:$\theta_0, \alpha > 0, K > 0$;
$k \leftarrow 0$;
while $k < K$ **do**
  $j \backsim \mathcal{U}(1, n)$;
  $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha\nabla_{\boldsymbol{\theta}}J_j\left(\boldsymbol{\theta}_k\right)$;
  $k \leftarrow k + 1$;
end

The SGD algorithm is generally well-working before the optimal region is reached, but it struggles close to the optimum due to the simplification in Equation 3.26. As the gradient is sampled at random, it has a high variance. A compromise between SGD and BGD is mini-batch gradient descent(MBGD). MBGD samples b, independent samples from the dataset, and calculates the gradient from this. This reduces the variance of the gradient, and improves performance overall.

One of the state-of-the-art ANN optimization algorithm is Root-Mean-Square prop(RMSprop). It is an unpublished optimization algorithm for neural networks first proposed by Geoffrey Hinton. It is a mini-batch gradient descent algorithm that improves the standard gradient-step iteration in the following way:

- Adaptive learning rates that scale with an exponentially decaying moving average of past gradients.

This reduces oscillation that can occur in standard SGD algorithms[44] due to high variance, and gives possibility for large initial step-sizes that decreases when optimum is approached[45]. V_k in Equation 3.27 is the moving average of past gradients, and is

referred to as the second-order momentum.

$$V_k = \beta_1 V_{k-1} + (1 - \beta_1)(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k))^2 \tag{3.27}$$

β_1 is the decay rate, and is typically in the order of 10^{-3}. This leads to the momentum being most reliant on the current gradient. The parameter update-step in RMSprop is given by Equation 3.28.

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \frac{\alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)}{\sqrt{V_k + \epsilon}} \tag{3.28}$$

α is the step size, and it is affected by the square root of the momentum. This leads to an exponentially decaying factor that will decrease the step size as the gradients gets smaller near the optimum. The inclusion of average previous gradients does however ensure that the step-size never diminishes, although the gradients near optimum is near zero. Note that the squaring-operation and summation is element-wise, as the parameter-set is multidimensional. Algorithm 2 shows the pseudocode for RMSprop[46].

Algorithm 2: RMSprop

Require: Training Set $\mathcal{T}$, Step size α, Decay rate β_1, ϵ, Mini-batch size b;
Initialize: $\boldsymbol{\theta}$ from some predefined distribution;
$k \leftarrow 0$;
while *Not convergence* **do**
  Draw mini-batch $\mathcal{B}$ with size b from $\mathcal{T}$;
  Compute gradient on mini-batch $g_k = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{B})$;
  Update momentum $V_k = \beta_1 V_{k-1} + (1 - \beta_1)(g_k)^2$;
  Update variable $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \frac{\alpha g_k}{\sqrt{V_k + \epsilon}}$;
  $k \leftarrow k + 1$;
end

Learning in ANNs typically happen in as batch training. Here, all the data is available instantaneously, and the learning consists of looping through this data and optimizing the network. In a DRL setting, the learning happens as iterative training. The optimization algorithm is then called as a part of another algorithm(DRL algorithm), and the networks parameters are updated at each iteration with a small amount of data(experience). This data is typically gradients calculated with respect to the loss function of the DRL algorithm.

3.8.2 Value Based Methods

Value based methods in DRL are based on the same fundamental theory as described in section 3.7.2.1. The main goal is to build a value function, which then can be used to define a policy[36]. In DRL, a deep learning algorithm, often an ANN features in this

process. One of the first DRL algorithms was introduced by Mnih et al. in 2015, when they combined ideas from Q-learning and deep learning to form the Deep Q-Networks(DQN)-algorithm[36, 47].

3.8.2.1 Deep Q-Networks

Using non-linear function approximatiors such as ANNs to approximate value functions had a history of making RL algorithms unstable[47]. This was due to the correaltion between observations, and the fact that small changes in the action-value function could considerably change the policy[47]. Minh et al. adressed this in their paper, which proposed a value-based DRL algorithm called Deep Q-Network(DQN). They proposed a solution that utilized a concept called experience replay. This replayed random data from the environment-agent interaction, reducing the number of interactions and removing the correlation between the observations[36].

In their solution, they used a deep convolutional neural network to estimate a parameterized action-value function $q(s, a; \boldsymbol{\theta})$, with parameters in $\boldsymbol{\theta}$. The loss function at any iteration i used to evaluate the network is given by Equation 3.29. All experience from the agent-environment interaction is stored in a dataset D. During learning, the action-value function is updated by drawing data uniformly from this dataset, and applying the Q-learning step(Equation 3.16). This is experience replay. γ is the discount rate, θ_i are the Q-network parameters at iteration i and θ_i^- are the network parameters used to compute the target at iteration i[47].

$$L_i\left(\theta_i\right) = \mathbb{E}_{(s,a,r,s)\sim U(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right] \tag{3.29}$$

The DQN algorithm outperformed every other RL algorithm at playing the games of Atari 2600, in 43/49 possible games that are difficult for humans[47]. This breaktrough made DRL surge in popularity. The major drawback of the DQN algorithm is that it does not support continuous state- and action spaces.

3.8.3 Policy Gradient Methods

Policy gradient methods, as described in 3.7.2.2 uses stochastic gradient ascent methods with respect to the policy parameters to optimize the expected return. ANNs can be used to estimate the policy, giving algorithms that fall into the sub-class of DRL. Modern DRL algorithms typically use value functions as well as policy optimization to improve the overall learning in the algorithm.

3.8.4 Actor-Critic Methods

Actor-critic methods are increasing in popularity as they combine policy search methods and learned value functions[40]. These algorithms can learn from either full returns or TD-methods. Actor-critic methods offer a variety of implementations of deep learning. Artificial neural networks can be used in policy approximation, value function approximation, and typically in both.

3.8.4.1 Deep Deterministic Policy Gradient

Deep deterministic policy gradient(DDPG) is an actor-critic DRL algorithm that estimates both a action-value function and a policy. It was introduced by Lillicrap et al. in 2016[48], and was based on the deterministic policy gradient algorithm, which supports continuous action- and state spaces. The DDPG algorithm utilizes the same network architecture, hyperparameters and learning algorithm as the DQN[48]. However, as DQN only handle discrete and low-dimensional spaces, it has limited application in control problems. The DDPG was developed to fill this gap.

3.8.4.2 Advantage Actor Critic

There exists several advantage actor critic algorithms. The term advantage refers to that the algorithm uses the advantage function to evaluate the policy. The advantage function, as defined in Equation 3.8 quantifies the value of taking an action, compared to other available options.

One advantage actor critic algorithm was implemented by Google DeepMind scientists Mnih et al in 2016[49]. It is called **Asynchronous Advantage Actor Critic(A3C)**[49]. The A3C algorithm is one of the newer state-of-the-art on-policy DRL algorithms[40]. In the original implementation, the A3C algorithm maintains a value function estimate, and a policy estimate. These are implemented as separate ANNs that update asynchronously. The algorithm maintains several parallel instances of the environment with independent agents interacting with the respective environments. The parallel environments introduce two main factors to the algorithm:

- Parallel environments stabilize the parameter update process, as larger samples are visited in each iteration of the algorithm[40]. It also decorrelates data, replacing the replay buffer of DQN.
- Multiple environment instances explicitly increases exploration. Larger portions of the state-space are explored in each iteration, leading to theoretically faster learning. In the original implementation, the training time reduction is approximately linear with the number of parallel workers[49].

In addition to this, the parallel workers allow of for more efficient use of hardware. The algorithm can run effectively on consumer hardware that has a CPU with multiple cores. Individual agent-environment instances are assigned to individual threads. Although this has no effect on the algorithm learning, it is a significant factor in this thesis as all experiments are run on consumer hardware.

The advantage function in A3C is defined by Equation 3.30a. $V(s;\theta_v)$ is the state-value function estimate from an ANN, with parameters in θ_v. The summation of discounted rewards is equal to the return, and can be rewritten as in Equation 3.30b.

$$A\left(s_t, a_t; \theta\right) = \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V\left(s_{t+k}; \theta\right) - V\left(s_t; \theta_v\right) \tag{3.30a}$$

$$A\left(s_t, a_t; \theta\right) = G_t - V\left(s_t; \theta_v\right) \tag{3.30b}$$

The A3C algorithm bootstraps, meaning it does not wait for a full episode to finish before it updates the value function estimate. The individual agents does k time steps before the parameters are updated, where k can vary. This is the n-step TD learning update described in section 3.7.1.2. The algorithm minimizes the loss function given by Equation 3.31[50].

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} A_{\theta,\theta_v}\left(s_t, a_t\right) \log \pi_\theta\left(a_t \mid s_t\right) + \beta H_\theta\left(\pi\left(s_t\right)\right) \right] \tag{3.31}$$

The update steps in the A3C algorithm separate two sets of parameters:

- θ and θ_v which are global parameters.
- θ' and θ'_v which are parameters for the individual agents.

Each agents starts with a copy of the global network parameters, θ and θ_v, and does k interaction with the environment. When k interactions are reached, the individual agent calculates the loss function. The loss function is used to calculate the gradients of the global network with respect to the parameters from the individual agent. This update step is shown in Equation 3.32. Note that the policy parameters are updated with the policy gradient theorem update, scaled with the advantage function instead of the full return.

$$d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi\left(a_t \mid s_t; \theta'\right)\left(G_t - V\left(s_t; \theta'_v\right)\right) \tag{3.32a}$$

$$d\theta_v \leftarrow d\theta_v + \nabla_{\theta_v}\left(G_t - V\left(s_t; \theta'_v\right)\right)^2 \tag{3.32b}$$

The update from the individual agents does not happen simultaneously, hence the name asynchronous. When an agent updates the parameters, it is updated to the global parameters. As a consequence of this, the parallel agents may work with different policy parameters, increasing exploration further.

One successor of the A3C algorithm is advantage actor critc(A2C). It is a synchronous version of the A3C. It builds on the same principles, and minimizes the same loss function given by Equation 3.31. Instead of updating the global networks with the individual networks separately, it waits for all the agents to finish their n-step TD segment, and updates the global parameters with all the computed gradients. When the A3C algorithm was proposed, the effect of the asynchronous updates were pulled into questioning, and the A2C implementation was a result of this[51].The A2C algorithm performed better than the A3C algorithm overall[51].

As all the parallel agents has to wait for the others agents to finish their segment, the algorithm A2C is marginally slower than the A3C. The synchronous update step contributes to more stable updates than the asynchronous, as a larger sample of the state-transitions are taken into consideration. The update step in the A2C algorithm can be seen in Equation 3.33, where i denotes the individual agents[50].

$$d\theta \leftarrow d\theta + \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta \left(a_{t,i} \mid s_{t,i}\right) \left(G_{t,i} - V_{\theta_v}\left(s_{t,i}\right)\right) \tag{3.33a}$$

$$d\theta_v \leftarrow d\theta_v + \sum_{i=1}^{n} \nabla_{\theta_v} \left(G_{t,i} - V_{\theta_v}\left(s_{t,i}\right)\right)^2 \tag{3.33b}$$

The original implementations of A3C and A2C use an entropy regularization term in the loss function for improved exploration. As both algorithms are on-policy algorithms, they can have a tendency to converge to sub-optimal deterministic policies, as both the actor and the critic updates the network based on the same data. This makes it easier for the actor and the critic network to optimize only a small subset of the state-space. The entropy-term, denoted $H_\theta(\pi(s_t))$ is given by Equation 3.34.

$$H\left(\pi\left(s_t\right)\right) = -\sum_{a \in A} \pi\left(a \mid s_t\right) \log \pi\left(a \mid s_t\right) \tag{3.34}$$

The entropy term is low when the certainty in an action is high, and vice versa. The entropy term is added to the loss function of the A2C algorithm. As a consequence of this, actions with high uncertainty will reduce the loss function with a larger amount than an action with high certainty. This is what explicitly encourages exploration through entropy. Alogrithm 3 shows the A2C pseudocode. The implemented solution utilizes a normalized advantage function, $A_t^{GAE(\gamma,\lambda)}$, where γ is the discount factor for rewards as

in Equation 3.30a and λ is a normalization constant.

Algorithm 3: A2C

Require: Initial network weights θ, θ_v, number of environments n;

Initialize: Global step counter $T = 0$, environment step counter $t = 0$;

while $T < T_{max}$ **do**

  $t_{start} = t$;

  Get state s_t;

  while *($t - t_{start} < t_{max}$)* ***or*** *(terminal observation = False)* **do**

    Sample $a_t \in \pi_\theta(a_t|s_t)$;

    for $i \in 1...n$ **do**

      Take action $a_{t,i}$ in environment i;

      Receive reward $r_{t,i}$ and state $s_{t+1,i}$;

      $t \leftarrow t + 1$;

      $T \leftarrow T + 1$;

    end

  end

  if *terminal observation* **then**

    $G_t = 0$;

  end

  else

    $G_t = V_{\theta_v}(s_t)$;

  end

  for $t \in t - 1...t_{start}$ **do**

    Calculate TD error: $\delta_t^V = r_t + \gamma V_{\theta_v}(s_{t+1}) - V_{\theta_v}(s_t)$;

    Calculate normalized advantage: $A_t^{GAE(\gamma,\lambda)} = \delta_t^V + \gamma\lambda A_{t+1}^{GAE(\gamma,\lambda)}$;

    $G_t = A_t^{GAE(\gamma,\lambda)} + V_{\theta_v}(s_t)$;

    $d\theta \leftarrow d\theta + \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta(a_{t,i} \mid s_{t,i})(G_{t,i} - V_{\theta_e}(s_{t,i}))$;

    $d\theta_v \leftarrow d\theta_v + \sum_{i=1}^{n} \nabla_{\theta_v}(G_{t,i} - V_{\theta_v}(s_{t,i}))^2$

  end

  Update θ, θ_v with the gradients $d\theta, d\theta_v$ respectively, using the step of a network optimizer algorithm.

end

4 | Implemetation

This chapter describes the implementation of the environments that simulate the drilling process and RL agents that are trained to act as autodrillers. The implementation consists of training and testing the model-free deep RL algorithm A2C(section 3.8.4.2) on environments with increasing complexity. The two most complex environments are based on Eckel's ROP model and Bourgoyne and Young's ROP model. Points to note:

- The models does not utilize SI-units, and follow the units found in literature. This is done for simplicity, and comparability, as the point of this thesis is not to produce a new, precise ROP model, but to implement an RL agent to optimize a drilling model.
- The term observation in this section refers to the term state in RL terminology. The term system state refers to states of the drilling model.

The chapter is outlined in the following way:

- section 4.1 describes the interface all environments follow.
- section 4.2 describe the common structure for all environments.
- section 4.3, section 4.4, section 4.5 and section 4.6 describes the implementation of the different environments, and the parameters and constraints implemented in the training process.
- section 4.7 describes the algorithm which is implemented through Stable Baselines 3.
- section 4.8 clarifies how the agents are evaluated and what parameters are configured in the respective environments during validation.

4.1 Interface

The environments implemented in this thesis follow the OpenAI gym interface. The gym interface generates the environment part of the RL problem, and assumes interaction with an agent that takes in an observation from the gym environment[52]. It then assumes

the agent outputs an action that the environment uses to generate a new observation. In addition to this, the environment produces a reward signal at each interaction.

The gym interface is implemented as a class in Python called Env. To customize environments, a class that inherits Env has to be implemented. To meet the interface standard of the gym Env class, it needs the following functions:

init(): Initializes all the constant values and/or random variables in the environment class. This includes defining the dimensions of the action and state space.

step(action): Is the main function of the Env class. It takes in a valid action from the action space, and has to produce the following four variables:

- observation: An observation that is changed by the input action. This is what is called state in RL theory.
- reward: A numerical signal that quantifies the desirability of the current state. The rewards can be produced at each iteration or sparsely.
- done: A Boolean that is 1 if the environment has reached its terminal condition, and 0 otherwise.
- info: Additional information about the agent-environment interaction that can be empty. Typically this variable holds information of normalization statistics, etc.

reset(): The function resets the observation, and other variables that should be reset or changed when the environment has reached terminal observation. This function can be used to change parameters in the environment at specific conditions.

render(): A function that is used for visualization of the agent-environment interaction, and can be as complex as graphic visualization, but can also be an empty function. The function is typically used in evaluation of agents, and not in training.

4.2 Environment Structure

The different environments implemented in this project follow the same general structure. The RL agent sends an action vector, a_t, to the environment. The action vector has the same size as the number of input variables in the ROP model. E.g, a model that has WOB,RPM and q as input would receive an action of size three, one for each variable. The individual actions can either be -1, 0 or 1. This represents the direction the respective input variables will be changed. The action vector is then scaled with a scalar λ. This decides how much the input values are adjusted at each iteration. The system input in time t is given by Equation 4.1.

$$u_t = u_{t-1} + \lambda a_t \tag{4.1}$$

The ROP is then calculated with the ROP model implemented in the respective environments. The system states in time t, x_t, holds ROP in time t, ROP_t, and the input values in time t, u_t, as shown in Equation 4.2. The size of ROP_t is scalar, and the size of u_t is model specific.

$$x_t = [ROP_t, u_t] \tag{4.2}$$

The environment updates a memory vector with past system states. The number of past system states the memory vector holds is environment specific.

$$m_t = [x_{t-1}, ..., x_{t-n}] \tag{4.3}$$

The environment observation vector holds the current system states and the memory vector. This is the output from the environment, which the agent receives at each interaction.

$$s_t = [x_t, m_t] \tag{4.4}$$

The memory in the observation vector ensures that the agent receives the necessary information for evaluating ROP without knowing the model, and that the problem has the Markov property(section 3.1). All information the agent needs to take an action is present in one observation. Figure 4.1 shows the general structure of the environments implemented in this project. Note that the observation vector s_t is what is referred to as a state in RL theory. The system state vector, x_t holds the ROP model variables.

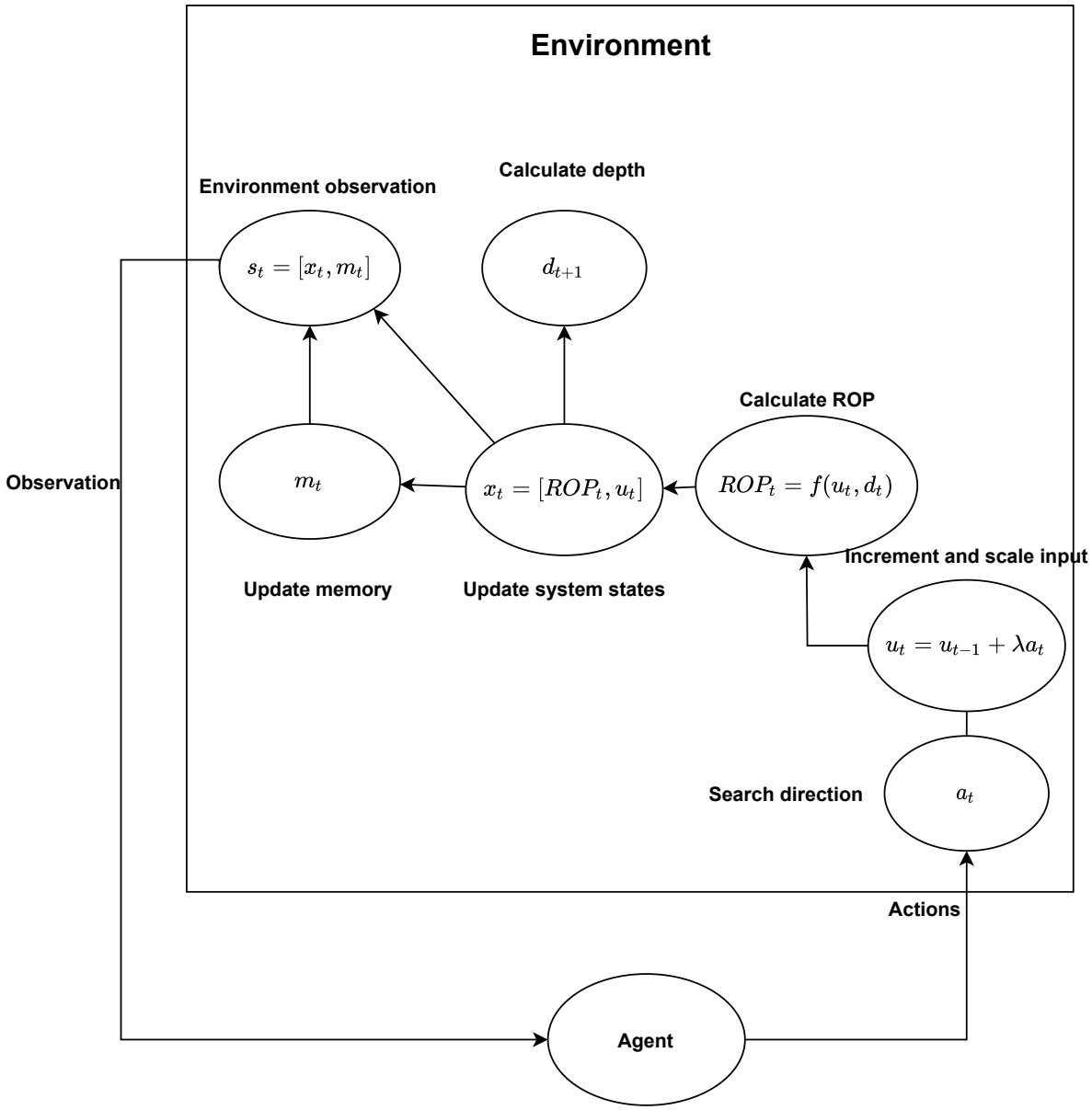


Figure 4.1: General environment structure.

4.3 Environment 1: Single Input

Environment 1 uses the ROP model given by Equation 4.5, where WOB_* is the optimal WOB that gives the maximum ROP.

$$ROP = \frac{1}{WOB_*}(WOB_*^2 - (WOB - WOB_*)^2) \tag{4.5}$$

The agent generates an action at each time step, and the input WOB is calculated through the input equation 4.1. As the model takes one input, WOB, the size of the action vector a_t is one(scalar). The environment generates the system states x_t through calculating ROP and input. The memory vector holds one previous system state, giving $m_t = [x_{t-1}]$.

This gives a total observation vector of $s_t = [x_t, x_{t-1}]$. The total size of the observation vector is $s_t \in \mathbb{R}^4$.

The objective of the agent is to converge to the optimal value $WOB = WOB_*$ This is reflected through the reward function r_t. The reward is generated at each iteration, and is given by Equation 4.6.

$$r_t = ROP_t - ROP_{t-1} \tag{4.6}$$

The reward function gives a positive scalar when ROP increases, and a negative scalar if it decreases. The agent receives reward frequently, leading to frequent policy updates. The maximum obtainable reward for an episode occurs when the agent converges to $WOB = WOB_*$.

The training process is episodic. This means that the agent interacts with the environment until some terminal condition is met. This terminal condition is a set depth. After this, the environment calls the reset() function, and a new episode begins. The terminal state of the system is implemented as a set depth of 200 ft. As there is no guarantee that the agent reaches the terminal state, the episode is ended if the agent does 10000 iterations without reaching the end goal. Neither WOB nor ROP is constrained by any threshold in this environment. Table 4.1 shows training specific parameters for environment 1.

Parameter	Value	Description
T	$1e5$	Finite time-step for training process
WOB_*	100	Optimal input value during the training process
λ	1	Step length on input from action
d_{final}	200	Terminal condition

Table 4.1: Training specific parameter for environment 1.

4.4 Enviroment 2: Multiple Input

Environment 2 is implemented as a multiple input version of environment 1. The ROP model of environment 2 is given by Equation 4.7, where $\boldsymbol{u} = [WOB, RPM, q]$

$$ROP = \sum_{i=1}^{3} \frac{1}{u_{i,*}} (u_{i,*}^2 - (u_i - u_{i,*})^2) \tag{4.7}$$

As the model takes three input variables, the size of the action vector a_t is three. The input vector is updated through Equation 4.1, and ROP is calculated through Equation 4.7. The system state vector $x_t = [ROP_t, u_t]$ is of size $x_t \in \mathbb{R}^4$. The memory vector m_t holds one previous system state, $m_t = [x_{t-1}]$. This gives the environment observation vector $s_t = [x_t, m_t] \in \mathbb{R}^8$.

The goal of this environment is to approach $\mathbf{u} = \mathbf{u}_*$. The reward function of environment 2 is identical of environment 1, and is given by Equation 4.6. Table 4.2 shows the training specific values for the agent training process in environment 2.

Parameter	Value	Description
T	$1e5$	Finite time-step for training process
WOB_*	100	Optimal input value during the training process
RPM_*	150	Optimal input value during the training process
q_*	200	Optimal input value during the training process
WOB_{max}	200	Minimum penalty value
RPM_{max}	300	Minimum penalty value
q_{max}	400	Minimum penalty value
λ	1	Step length on input from action

Table 4.2: Training specific parameter for environment 2.

4.5 Environment 3: Eckel's Model

Environment 3 is based on Eckel's model for ROP, described in section 2.2.2. The model, given by equation 2.2, will increase indefinitely for increasing input parameters, as Eckel's model assume constant parameter values for a specific rock formation. The model has therefore been modified to construct individually defined optimal values for the input parameters. The modified Eckel model can be seen in Equation 4.8. The model takes three input variables, $\boldsymbol{u} = [WOB, RPM, q]$

$$\begin{aligned} ROP =& K(WOB^a - \frac{a_{11}}{K+100}WOB^3) \\ & (RPM^b - \frac{a_{22}}{K+500}RPM^2) \\ & (\left(\frac{kq\rho}{d\mu}\right)^c - \frac{a_{33}}{K+1000}\left(\frac{kq\rho}{d\mu}\right)^2) \end{aligned} \tag{4.8}$$

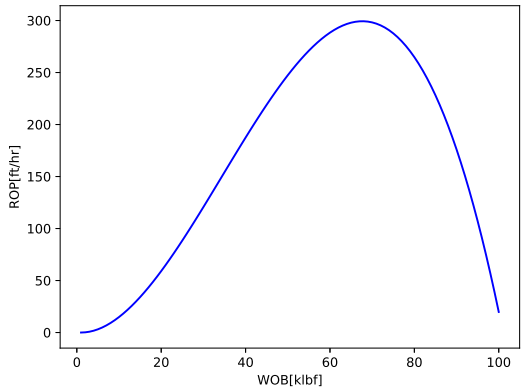
The model modification has been made to resemble the founder's point described in section 2.2.1. The chosen values for a, b, c are $a = 2$, $b = 1$ and $c = 1$. These were chosen to resemble the curve presented by Dupriest, and are held constant during experimentation. Although Dupriest describes the WOB-ROP relationship as linear, the exponent $a = 2$ is chosen to make WOB the dominant input. The K value is the drillability constant. This models the hardness of the rock formation. It is included in the negative terms in Equation 4.8 such that the optimal input values depends on the given rock hardness. K is the only parameter that is adjusted during experimentation. The rest of the constants are chosen experimentally to get the desired ROP behaviour according to Dupriest's curve. This desired behaviour is to have ROP increase for increasing input until some threshold value, after which it decreases. Table 4.3 shows the constant model parameter values that

are not adjusted during agent evaluation.

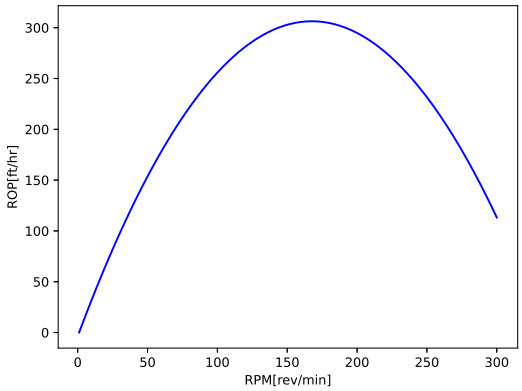
Parameter	Value
k	0.1
a_{11}	0.005
a_{22}	0.005
a_{22}	0.005
a	2
b	1
c	1
ρ	1000
μ	0.4
d	0.5

Table 4.3: Constant parameter Values for Eckel's modified model.

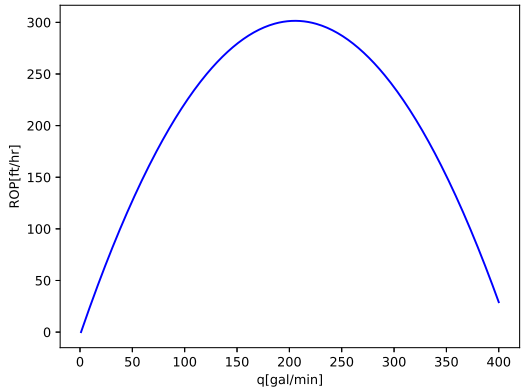
Figure 4.2 shows the resulting input-output curves with the model given by Equation 4.8 and Table 4.3, with a drillabillity constant $K = 0.5$.



(a) ROP-WOB curve in the modified Eckel model.



(b) ROP-RPM curve in the modified Eckel model.



(c) ROP-q curve in the modified Eckel model.

Figure 4.2: Input-output relationships of the modified Eckel model.

The action vector is identical to that of environment 2, as the ROP model takes three input variables. The system state vector, x_t, the memory vector, m_t and the environment observation vector s_t is also identical as environment 2. This is because they share the same number of input variables in the ROP model, and the memory vector only holds one previous system state.

The goal of the RL agent is to adjust input to the separate input values that maximizes ROP. The input values have a soft constraint that penalizes values above some set threshold. Equation 4.9 gives the reward function implemented in environment 3.

$$r_{1,t} = ROP_t - ROP_{t-1} \tag{4.9a}$$

$$r_{2,t} = \begin{cases} -10, & \text{if } u_i > u_{i,max} \\ 0, & \text{otherwise} \end{cases} \tag{4.9b}$$

$$r_t = r_{1,t} + r_{2,t} \tag{4.9c}$$

The RL algorithm is trained on episodic instances similar to environment 1 and 2, on one set model configuration. Table 4.4 shows the model configuration for the training process in environment 3.

Parameter	Value	Description
T	$2.5e6$	Finite time-step for training process
K	0.5	Drillability constant
WOB_{max}	200	Minimum penalty value
RPM_{max}	300	Minimum penalty value
q_{max}	400	Minimum penalty value
d_{final}	200	Terminal condition on episode
λ	1	Step length on input from action

Table 4.4: Training specific parameter for environment 3.

4.6 Environment 4: Bourgoyne and Young's Model

Environment 4 uses the ROP model described in section 2.2.3 to simulate the drilling process. Bourgoyne and Young's(BY) model is one of the more comprehensive analytical models for ROP, and is therefore one of the most popular. The model has eight different variables that effect the ROP model in some way, represented in eight individual functions. To simplify the model, some of the functions are removed. The remaining functions

represent the core functions for real-time drilling optimization, except the effect of depth on ROP. This is similar to what Soares et al. did in their ROP modeling study(section 2.3.1). The simplification removes the effect of overbalance, compaction and bit-wear. In the model given by Equation 4.10, W is WOB[Klbf], N is RPM[rev/min]. The output ROP is given in ft/hr. q features in the jet force F_j and has units gal/min. Although f_8 models the jet force, in this implementation, it represents all hydraulics in the model, cleaning included.

$$ROP = f1 \cdot f5 \cdot f6 \cdot f8 \tag{4.10a}$$

$$f1 = e^{2.303 * a_1} \tag{4.10b}$$

$$f_5 = \left(\frac{\frac{W}{d_b} - \frac{W_t}{d_b}}{4 - \frac{w_t}{d_b}} \right)^{a_5} \tag{4.10c}$$

$$f_6 = \left(\frac{N}{60} \right)^{a_6} \tag{4.10d}$$

$$f_8 = \left(\frac{F_j}{1000} \right)^{a_8} \tag{4.10e}$$

$$F_j = K \cdot \rho \cdot q \cdot v \tag{4.10f}$$

The implemented environment assumes constant parameter values for a given rock formation. a_1 represents the drillability parameter for the specific rock formation, and a_5, a_6 and a_8 model the effect input WOB, RPM and q has on output respectively. To get an input-output curve similar to that of Dupriest, the equations are modified to have individual optimal values.

$$f_{mod,5} = \left(\frac{\frac{W}{d_b} - \frac{W_t}{d_b}}{4 - \frac{w_t}{d_b}} \right)^{a_5} - a_{11} \left(\frac{\frac{W}{d_b} - \frac{W_t}{d_b}}{4 - \cdot \frac{w_t}{d_b}} \right)^{a5 + \frac{4}{3}} \tag{4.11a}$$

$$f_{mod,6} = \left(\frac{N}{60} \right)^{a_6} - a_{22} \left(\frac{N}{60} \right)^{a_6 + 8} \tag{4.11b}$$

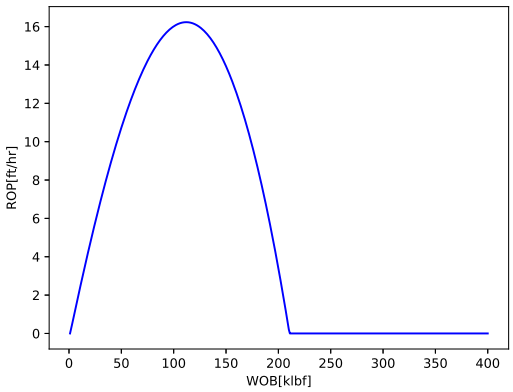
$$f_{mod,8} = \left(\frac{F_j}{1000} \right)^{a_8} - a_{33} \left(\frac{F_j}{1000} \right)^{a_8 + \frac{4}{3}} \tag{4.11c}$$

To further align with the theory of Dupriest's drilling curve(Figure 2.3), only non-negative values for ROP should be viable. This results in the final ROP model implemented in environment 4 as given by Equation 4.12

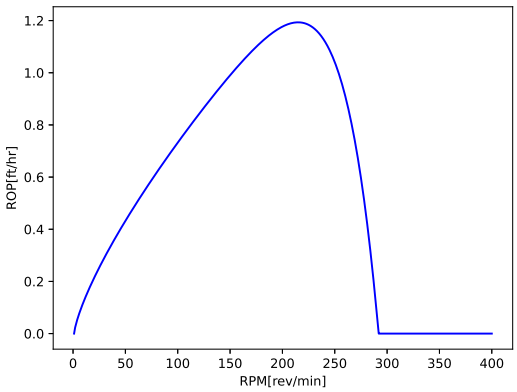
$$ROP = f_1 \cdot max(0, f_{mod,5}) \cdot max(0, f_{mod,6}) \cdot max(0, f_{mod,8}) \tag{4.12}$$

To get the desired input-output curves, according to drilling in Region II of Dupriest's

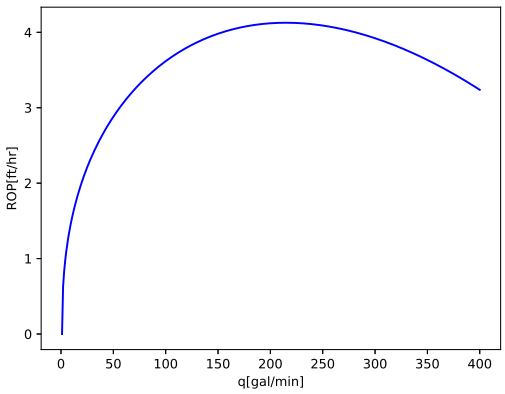
drilling curve, $a_5 = 1$ and $a_6 = 0.75$ is chosen for the training process. $a_8 = 0.4$ is chosen to represent the effect of hydraulics on output. It is chosen such that hydraulics have a large effect up until some value. After this, cleaning is sufficient and the effect of hydraulics diminish. The formation hardness coefficient $a_1 = 1$ is arbitrarily chosen within the bounds of Table 2.2. This leads to the input-output curves given by Figure 4.3. The values chosen for the training process is less important in this model, as the formation specific constants are modified during experimentation and evaluation.



(a) ROP-WOB curve in the modified BY model.



(b) ROP-RPM curve in the modified BY model.



(c) ROP-qcurve in the modified BY model.

Figure 4.3: Input-output relationships of the modified BY model.

Table 4.5 shows the constant parameters in the BY model. The constants are chosen to get the desired behaviour in Figure 4.3.

Parameter	Value	Description
W_t	1	Threshold WOB value to get sufficient DOC
ρ	22	Fluid density[lbs/gal]
v	10	Nozzle velocity[ft/min]
K	0.001	Constant in F_j
d_b	1	Diameter of bit[in]

Table 4.5: Constant parameter values for BY's modified model.

The action vector is identical to that of environments 2 and 3. It holds three values that represents the direction to adjust input. u_t holds $u_t = [WOB, RPM, q]$. It is updated through Equation 4.1. The system state vector $x_t = [ROP_t, u_t] \in \mathbb{R}^4$ is defined as in environment 2 and 3. The memory vector, m_t holds the three past system state vectors, $m_t = [x_{t-1}, x_{t-2}, x_{t-3}] \in \mathbb{R}^{12}$. This integration of more memory is unique to environment 4, and gives the observation vector $s_t = [x_t, m_t] \in \mathbb{R}^{16}$.

The reward function for environment 4 is identical to that of environment 3, and is given by Equation 4.9. The agent is rewarded for converging towards the optimal ROP, and penalized for exceeding the constraints on input.

Parameter	Value	Description
T	$5e6$	Finite time-step for training process
a_1	1	Drillability constant
a_5	1	WOB interaction coefficient
a_6	0.75	RPM interaction coefficient
a_8	0.4	Hydraulics interaction coefficient
WOB_{max}	200	Minimum penalty value
RPM_{max}	300	Minimum penalty value
q_{max}	400	Minimum penalty value
d_{final}	200	Terminal condition on episode
λ	1	Step length on input from action

Table 4.6: Training specific parameter for environment 4.

4.7 Algorithm

The algorithm implemented in this project is the advantage actor critic(A2C) with parallel agents. It is a model-free, on-policy deep reinforcement learning algorithm. It has no knowledge of the environment, and learns only based on observations from the environment. Discussion of algorithm choice will be presented in section 6.3. The stable baselines A2C implementation is used, with some modifications to be more in line with the original implementation described in section 3.8.4.2, and to stabilize learning. The parallel agents are initialized through the PyTorch multiprocessing library, which assigns instances of the algorithm and environment to different threads on the CPU. This is functionality present

in the stable baselines code. The stable baselines library was chosen as it has good documentation, which allows for easier customization, and an implementation of the desired A2C algorithm.

The optimizer algorithm used to update the parameters in the artificial neural networks is RMSprop(algorithm 2). The stable baselines implementation uses a different optimizer algorithm called Adam. This made training unstable, and RMSprop was chosen. The original implementation used RMSprop.

The artificial neural network parameters are saved using stable baselines functionality. This allows for easy saving and loading of trained agents. All input to the ANNs, which is the observation vector s_t and the reward r_t, are normalized so that they are in the range $[-1, 1]$. The normalization statistics for each trained agent are saved together with the network parameters for the trained agents using stable baselines functionality.

The algorithm is entropy-regularized as with the standard implementation. This functionality also exists in the stable baselines implementation, and can be modified through the hyper-parameter β_1.

The network structure of the algorithm is implemented as two separate feed-forward neural networks, with parameters in θ and θ_v. The original implementation used convolutional neural networks, which is more specialized towards pixel input. Table 4.7 shows the stable baselines recommended network architecture, which is used for environments 1,2 and 3. This is modified for environment 4, and is presented together with the results. Further discussion of algorithm implementation will be presented in section 6.3.

Network	**Hidden Layers**	**Input nodes**	**Output nodes**
Policy network π	64-64	Size of observation vector	Size of action vector
Value network V	64-64	Size of observation vector	Size of observation vector

Table 4.7: Network specifications.

Hyperparameter	Value	Description
γ	0.99	Return discount rate in A2C
α	10^{-2}	Step parameter in RMSprop
ϵ	10^{-8}	Numerical stability term in RMSprop
β_1	0.01	Entropy regularization coefficient in A2C
β_2	10^{-3}	Weight decay in RMSprop

Table 4.8: Hyper-parameters.

4.8 Evaluation of Agents

One agent is trained for each environment. In the training process, the agents utilize multiprocessing according to the A2C pseudo-code(algorithm 3). The training process

follows the environment configuration described in the respective environment sections. To evaluate the different agents, simulations are generated from the environments. In these simulation instances, some of the model parameters are tweaked to see how well the different agents generalize and optimize ROP. In all simulation cases, the multiprocessing aspect of the algorithm is disabled, and there is only one agent-environment instance running. The multiprocessing is disabled to mimic the agent being deployed in a drilling system.

The continuous run of the simulations for the different environments follow the same structure. The environment produces an observation vector. The agent outputs an action vector based on the observation. The environment then produces a new observation, as shown in Figure 4.1. This loop runs until the terminal depth is reached. The agents can be evaluated on a single formation(parameter configuration), or the simulation can be looped to simulate drilling a case with several formation types.

The environments have different parameters that can be adjusted to evaluate the agent. Table 4.9 gives an overview of which parameters are adjusted in the respective environments to assess the RL agents performance.

Parameter	**Environment**	**Description**
WOB_*	1&2	Directly places the optimum of the input
RPM_*	2	Directly places the optimum of the input
q_*	2	Directly places the optimum of the input
K	3	Drillability constant. Affects the optimum
a_1	4	Drillabillity constant. Scales ROP
a_5	4	Adjusts WOB-ROP relationship
a_6	4	Adjusts RPM-ROP relationship
a_8	4	Adjusts q-ROP relationship
d_{final}	1,2,3&5	Terminal condition

Table 4.9: Adjustable parameters in agent evaluation.

The RL agents can be evaluated as deterministic predictors, where the network parameters are not further updated after the initial training process. They can also be evaluated as continuous learners where the parameters are updated during evaluation. The agents trained on environment 1 and 2 are evaluated as predictors, and agents trained on environment 3 and 4 are evaluated as both predictors and continual learners. This will be specified with the results.

5 | Results and Discussion

This chapter presents the results from training and experimenting with four different RL agents on the four environments described in chapter 4. The algorithm the agents follow is the model-free, deep RL algorithm A2C. Some points to note are:

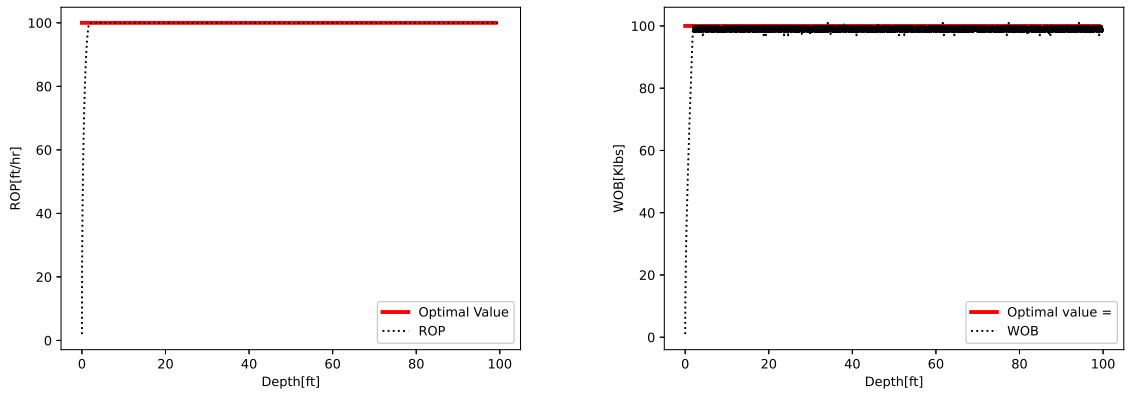
- The output ROP is plotted as the top left plot, with the remaining three plots being input WOB,RPM and q. In environment 1, which is single input, ROP is plotted to the left and WOB to the right.
- One agent is trained per environment.

The chapter is outlined in the following way:

- Section 5.1 presents the results from experimentation with the RL agent trained on environment 1. The agent acts as a predictor and does not learn.
- Section 5.2 presents the results from experimentation with the agent trained on environment 2. The agent acts as a predictor
- Section 5.3 presents the results from experimentation with the agent trained on environment 3. The agent is evaluated both as a predictor and a continual learner.
- Section 5.4 presents the results from experimentation with the agent trained on environment 4. The agent is evaluated as both a predictor and continual learner. As environment 4 contains the most comprehensive ROP model, this section contains the most experimentation.

5.1 Environment 1: Single Input

Figure 5.1 shows the agent trained on environment 1. The model in the simulation has identical parameters to the training process($WOB_* = 100$). The agent does not learn in this simulation. This means that the agent acts as a predictor, that does not consider the reward it receives from the environment, only the observation. The closed-loop interaction between agent and environment results in WOB converging to the optimal input value $WOB = WOB_*$. This simulation replicates an instance where the model of the system can be accurately described in the environment. The RL agents trains a number of simulation on the known formation, and is deployed in the real system when performance in training is satisfactory. It does however provide little knowledge of how robust the policy of the agent is, and how well it generalizes to slight modeling errors. To assess this, a simulation where the model parameters are different to the training process needs to be evaluated.

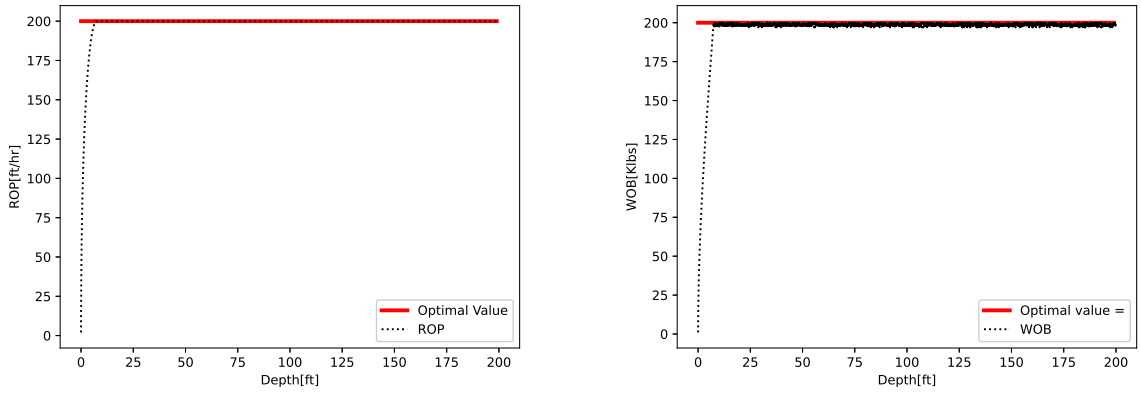


(a) ROP (b) WOB

Figure 5.1: Test of stationary model identical to training process.

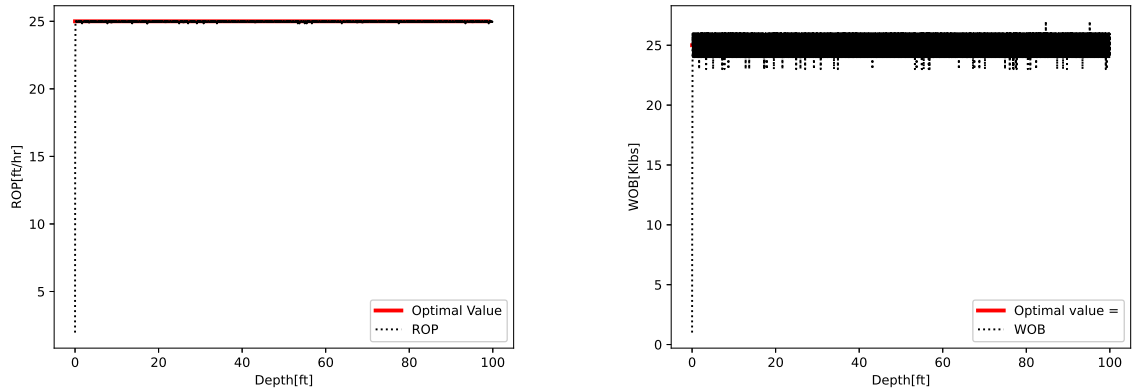
5.1.1 Validation

Figure 5.2 and Figure 5.3 shows the agent evaluated in section 5.1 without any further training. The optimal input value WOB_* of Equation 4.5 has been modified. The agent does not learn during the simulation. It can be seen that the agent manipulates the input WOB to the optimal values $WOB_* = 200$ and $WOB_* = 25$.



(a) ROP (b) WOB

Figure 5.2: Adjusted optimum to $WOB_* = 200$.



(a) ROP (b) WOB

Figure 5.3: Adjusted optimum to $WOB_* = 25$.

5.1.1.1 Discussion

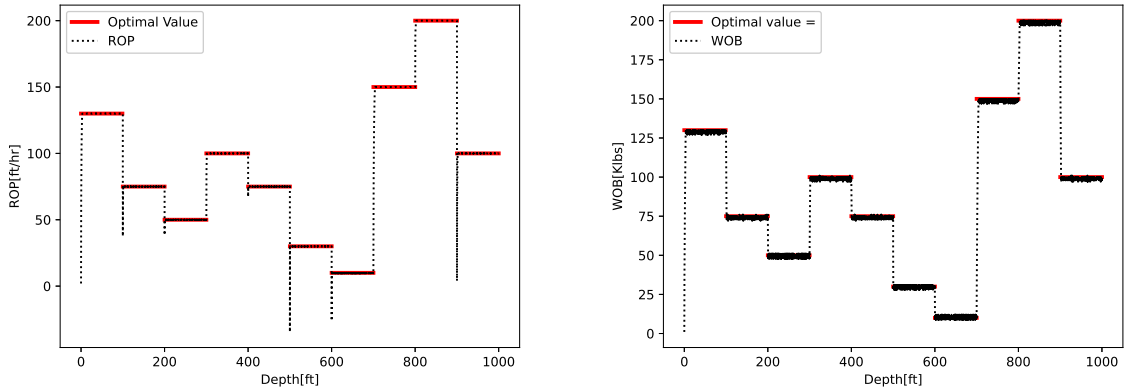
The agent generalizes well. Even though the model parameters are adjusted, the agent controls the input towards the optimal WOB. The agent learns a policy that replicates gradient ascent. The policy maps the difference in ROP_t and ROP_{t-1} to either an increase or decrease in input. The agent rarely maps any observations to a change in input of 0. This is because the agent receives the same reward for holding the optimum for two iterations as it does for adjusting input down and back up.

By construction of the model, a decrease in ROP will always lead to a worse return if that value is held throughout the episode. As a result of this, the agent can in reality be greedy, and never plan for future states. The action that gives the highest reward in the current state will always yield a higher return. The policy the agent learns is valid for all model configurations. The ROP model(section 4.3) is also convex in the entire domain. This makes the gradient ascent policy applicable to any parameter configuration of the

ROP model. The behaviour of the agent is comparable to that of the extremum seeking algorithm, which also is a model-free optimization technique that follows gradients to reach an optimum of the model through exploration of the state-space.

5.1.2 Drilling Test Case

Figure 5.4 shows a test case where the optimal value $WOB_*(d)$ varies with depth. The change in $WOB_*(d)$ occurs when the terminal depth for one formation is reached in the environment. The environment calls the reset function, and the parameter is changed. The agent detects formation changes through change of ROP in the observation vector, and decides what direction input is manipulated in response to the variable change. The agent controls the input to the optimal values for each case. Every segment of the drilling test case features previously unseen parameters. The agent does not learn in this simulation.



(a) ROP (b) WOB

Figure 5.4: Test case of 1000ft with varying model parameters.

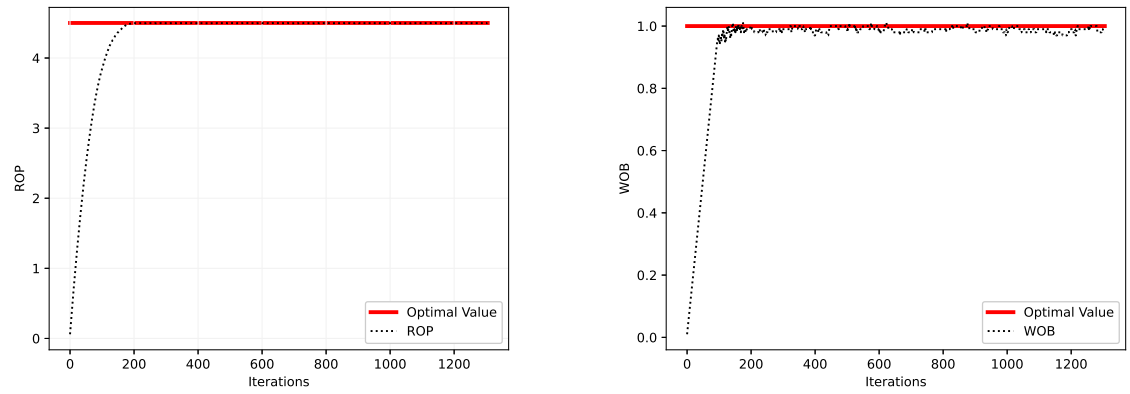
5.1.2.1 Discussion

The agent also generalizes well in the drilling test case. This test case brings three new challenges to the agent. The first is a change in model parameters during a simulation. The second is the initial observation the environment presents has when the model parameters change. The agent is trained in an episodic environment where the initial observation always is a vector of 0. Here the agents initial observation depends on the ending observation of the previous section. As the policy the agent has learned still is valid regardless of initial value, the closed-loop system converge to the optimal input for all parameter values. The third challenge is negative values for ROP. When the model parameters change, the initial observation for the agent can be a negative ROP value, which can sometimes be a challenge for algorithms that rely on probability distributions. However, the agent handles it well. Negative values for ROP is nevertheless unrealistic

in a drilling case. The purpose of this simulation however is not for it to be a realistic drilling case, but to test the RL agents capabilities of applying a learned model on unseen environment configurations. As the agent has essentially learned a gradient ascent like policy, it should generalize well to unseen problems, which it does. Arguably, optimization of a convex function with one input value is an easy task. In a machine learning setting however, generalization towards unseen data is never given, and this test visualizes the algorithms capability of generalizing well to unseen parameters. To increase complexity, a test case with three input varaibles, which is more realistic for drilling optimization will be analyzed.

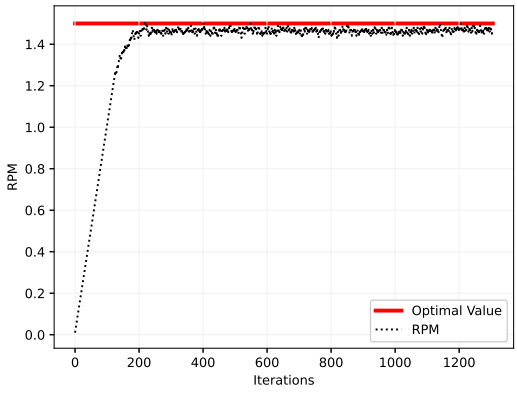
5.2 Environment 2: Multiple Input

Figure 5.5 shows a simulation of an agent trained on environment 2. The model parameters are identical to that of the training process. The agent manipulates all three input variables, WOB,RPM and q to approach maximum ROP, through individually choosing directions to adjust the input. Note that the Y-axis is scaled down by a factor of 0.01. This simulation shows the RL agents capability to find optimum input values on a model that can be accurately defined in an environment for training, with three input variables.

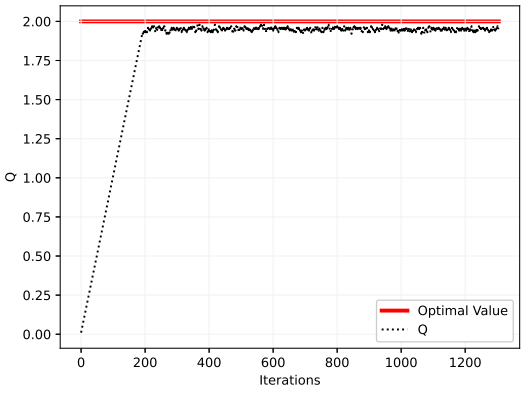


(a) ROP

(b) WOB



(c) RPM



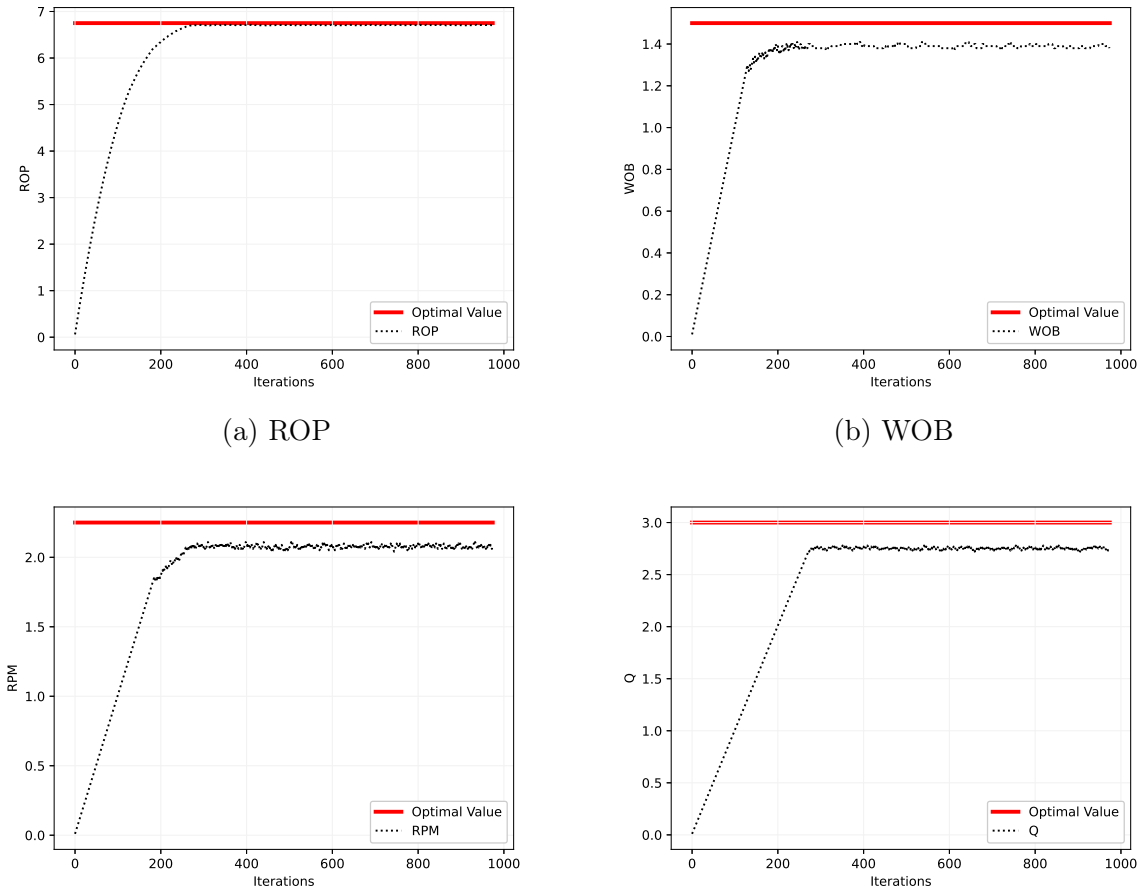
(d) q

Figure 5.5: Simulation of agent in environment 2 on a model configuration identical to in training.

5.2.1 Validation

Figure 5.6 shows the same agent without any further training deployed on a simulation with adjusted model parameters. The optimal values are directly placed in the model through Equation 4.7. In this simulation, the optimal input is adjusted by 50% compared to optimum in training, and the agent manipulates the input to a achieve ROP numerically

close to the optimal ROP. The agent does not learn during this simulation. The Y-axis is scaled down by 0.01.



(a) ROP

(b) WOB

(c) RPM

(d) q

Figure 5.6: Simulation of agent in environment 2 on unseen parameters.

5.2.1.1 Discussion

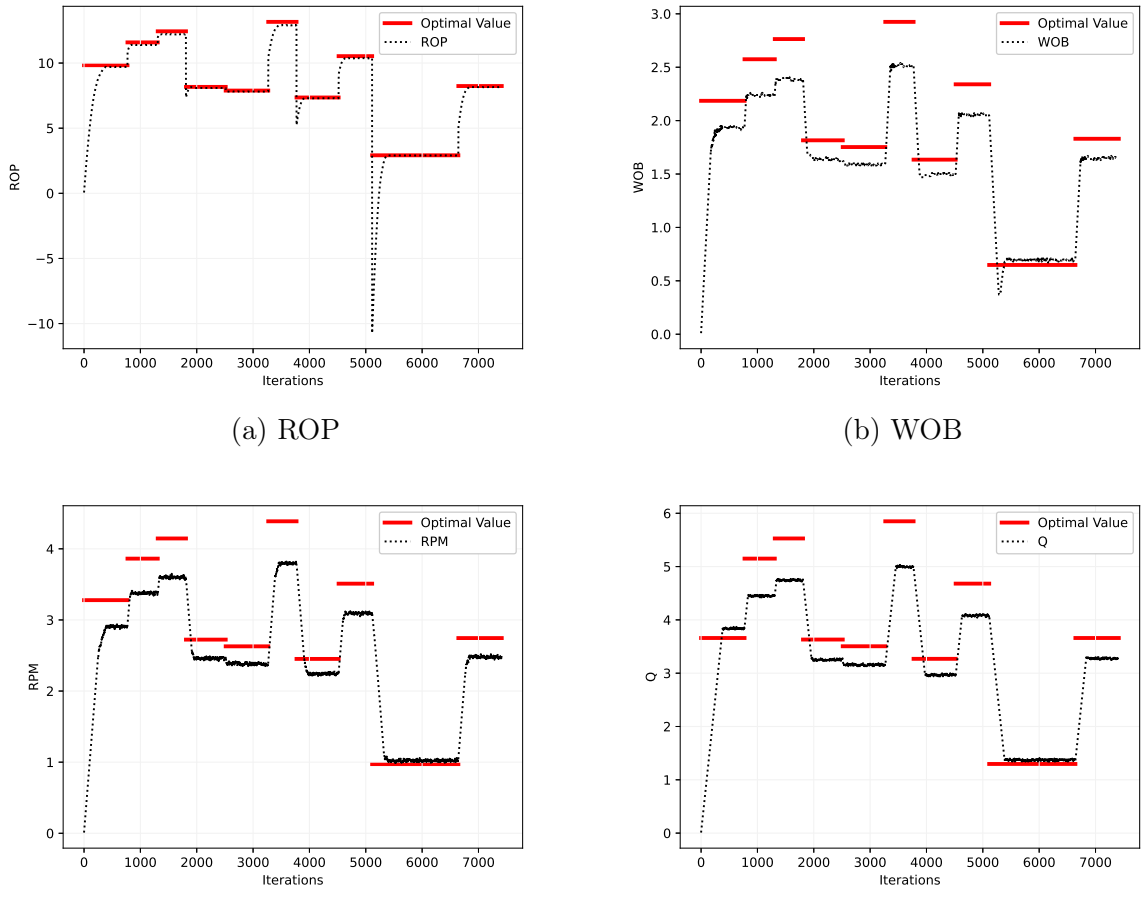
The ROP model of environment 2(section 4.4) has a flat optimal region by construction. This results in almost no gradients in the region close to the analytical optimum. As a consequence of this, the agent controls the input to achieve ROP numerically close to the optimum. The input values are however analytically not close to the optimum. The model is defined similarly as in environment 1, but it has more input variables to control in each iteration. The policy is still a gradient ascent policy, and due to the memory in the observation vector, the agent can relate decrease in ROP to a change in individual input parameters. The model is also convex for all parameter configurations.

One approach to reduce the discrepancy in the input values could be to introduce a varying λ(Equation 4.1), which is used to scale the actions from the agent. If the agent was allowed to take smaller or larger steps as it approached flat regions, more interesting policies could be discovered. A realistic drilling segment could have a really flat optimal

region in terms of ROP, while in reality some states introduce bit wear and drilling dysfunctions. In this environment however, the cost of excessive input is not modelled in the reward function.

5.2.2 Drilling Test Case

Figure 5.7 shows the same agent in a simulation generated by environment 2. The optimal input values, WOB_*, RPM_* and q_* are scaled by a constant drawn from a random distribution $K \backsim \mathcal{U}(0.1, 3)$ at an interval of 200ft. The agent manipulates the input such that ROP is numerically close to the optimal ROP. However, it is further away from the analytical optimum. Negative values for ROP is possible in the environment 2 model, as seen around 5000 iterations. The agent does not learn during this simulation.



(a) ROP

(b) WOB

(c) RPM

(d) q

Figure 5.7: Drilling test case where optimal input varies with depth.

5.2.2.1 Discussion

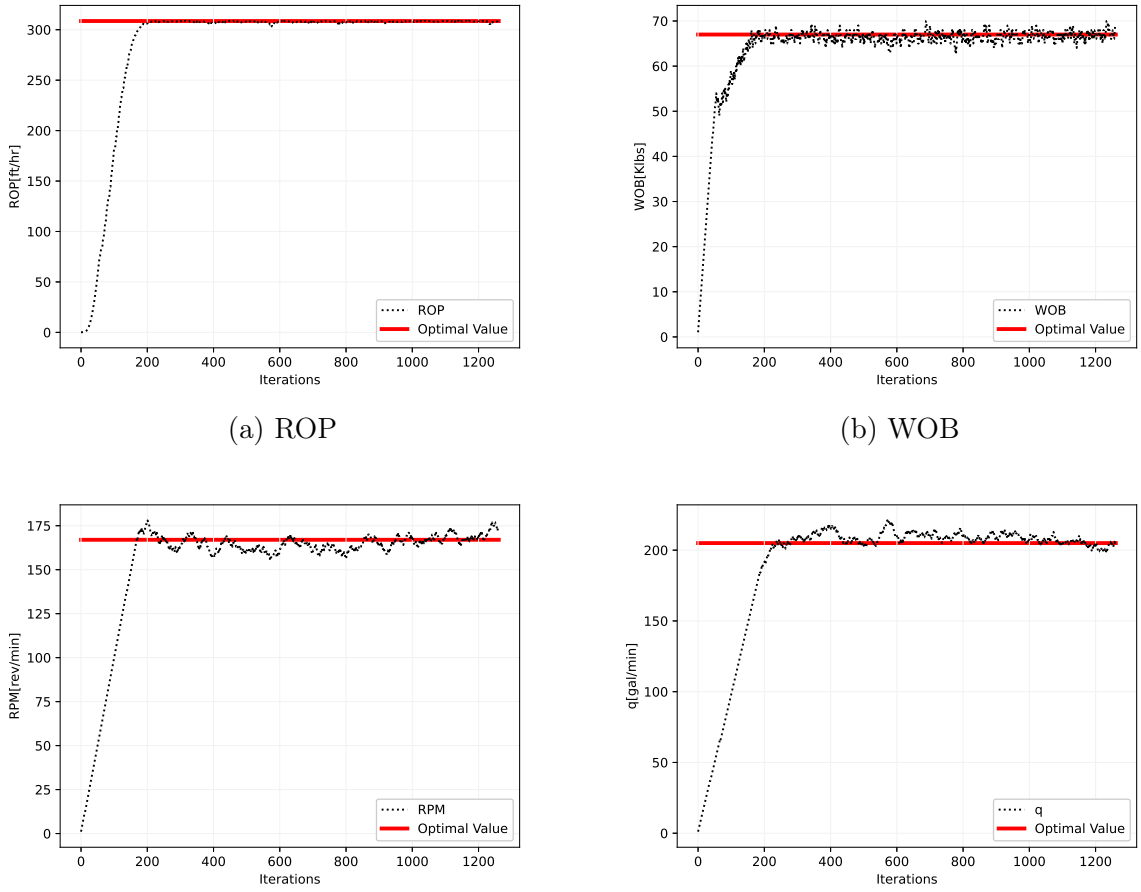
The simulation in Figure 5.7 is a very artificial case. As the optimal input values in the model are just scaled with a scalar, the RL agent can at each parameter change control the input values in the same direction. The purpose of this simulation is to see whether

or not the agent can control three input variables when the observation from the system suddenly change. The agent generalizes well to new data, and handles three separate input variables well. As the optimal region is flat, it is expected that a gradient ascent policy with a set step length λ will struggle to find the exact analytic optimum.

Although the agent does not find the analytical optimum, it can be seen that it manipulates the input to move ROP towards the numerical optimal ROP. The observation vector of the model also holds only one previous sytem state in the memory vector, which could provide difficulties for gradient search with three variables. The agent might struggle to relate change in output ROP to a specific input variable.

5.3 Environment 3: Eckel's model

Figure 5.8 shows an agent trained on environment 3 in a simulation identical to an episode in the training process. This means $K = 0.5$. In Figure 5.8, the agent does not learn. The agent controls the system to ROP values numerically close to the optimum, but has more variation in input than the previous sections. This simulation is the first test based on a published ROP model. It again shows that if the formations are known, and an accurate environment can be modelled for training, the RL agent can learn a policy to optimize the problem. The agent does not learn in this simulation.



(a) ROP

(b) WOB

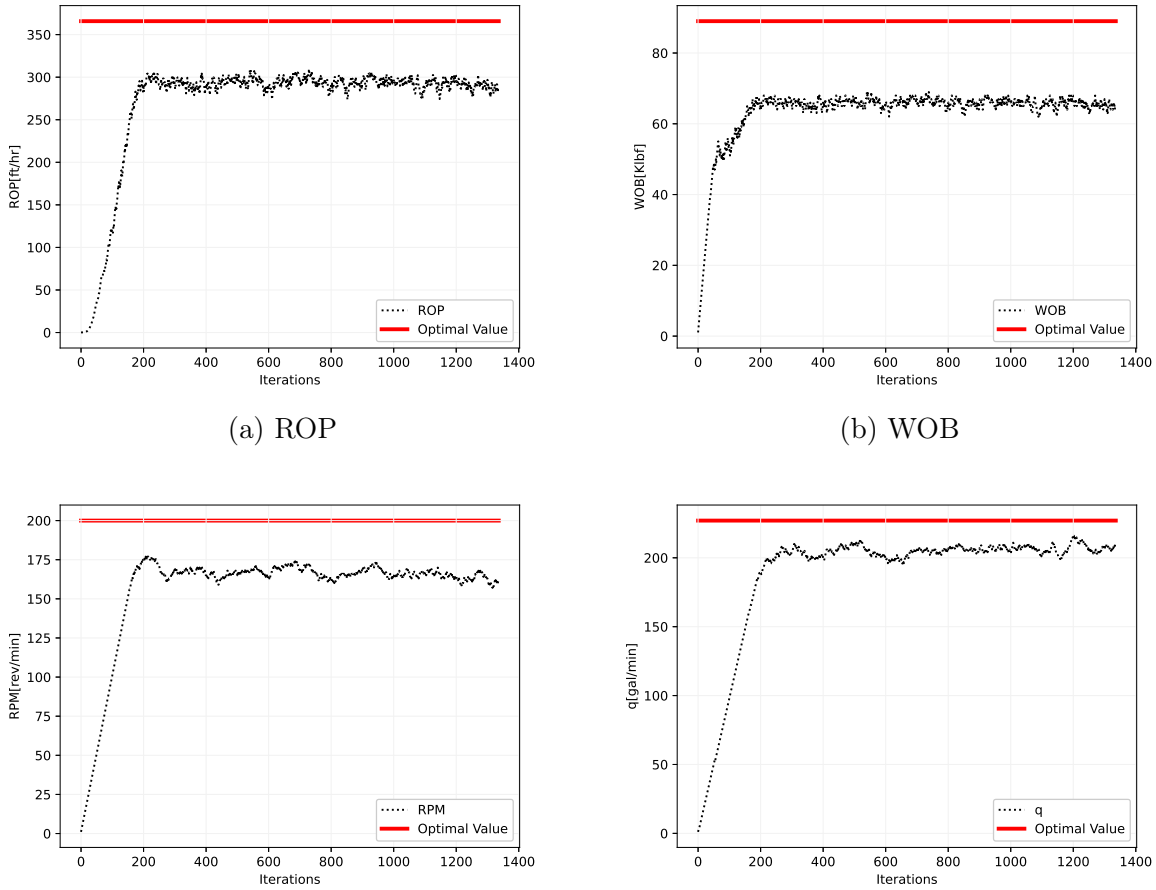
(c) RPM

(d) q

Figure 5.8: Simulation of environment 3 with a model configuration identical to training.

5.3.1 Validation

Figure 5.9 shows the same agent interacting with a simulation where the drillability constant is modified to $K = 0.25$. The agent generalizes poorly, and does not control the input to a value that is in the same range as the optimal value Figure A.2 in appendix A.2 shows another plot of poor generalization, where the agent overshoots. The agent does not learn in this simulation.



(a) ROP

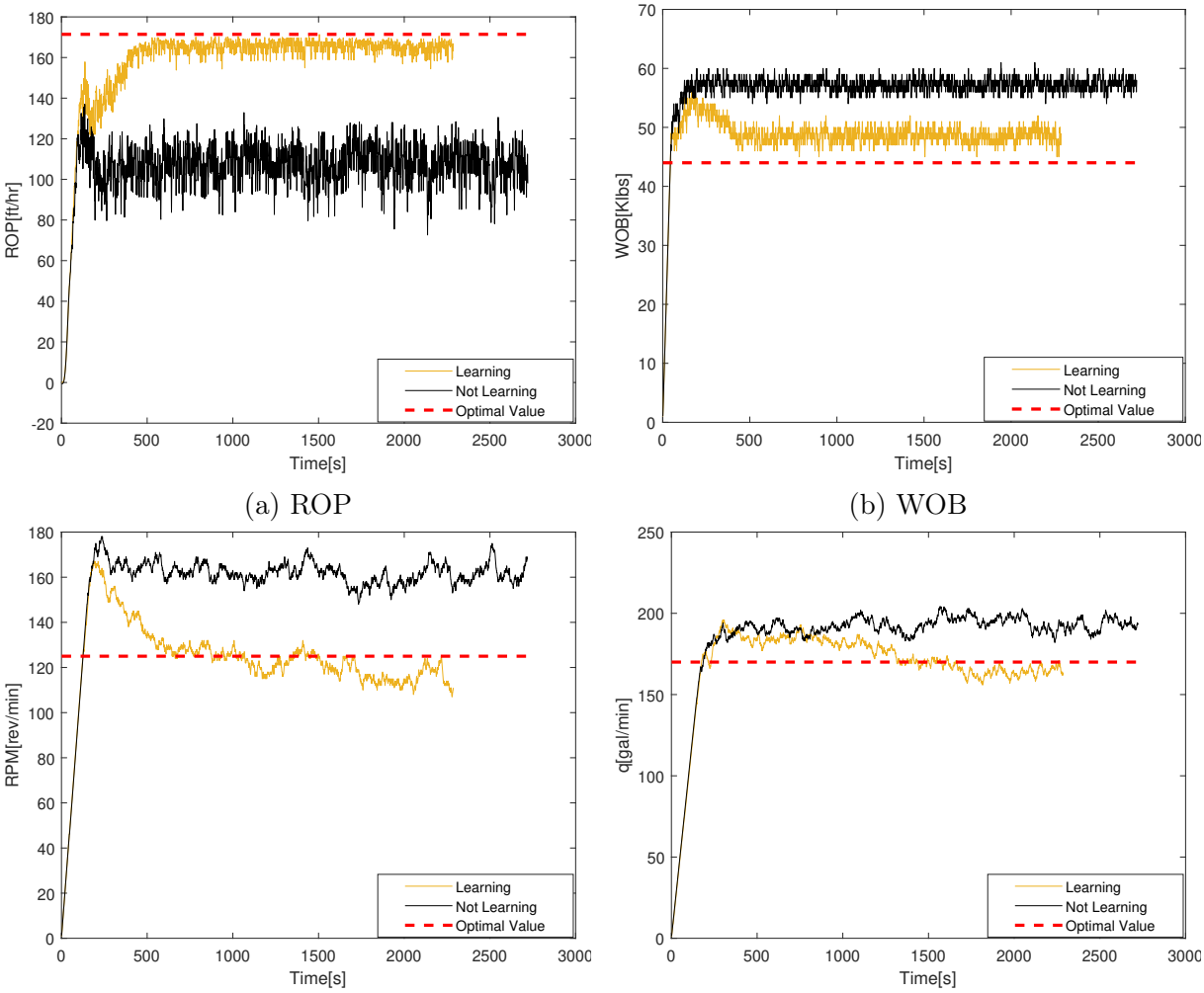
(b) WOB

(c) RPM

(d) q

Figure 5.9: Simulation with unseen model parameters.

Figure 5.10 shows two simulations of the same agent on identical instances of environment 3. In this simulation, $K = 1$. The difference between the two simulations is that the agent trajectory plotted in orange learns during the simulation. The agent trajectory plotted in black does not learn, and acts as a predictor like the previous sections. The simulation starts off with identical policy and value function network parameters, but the agent that learns updates these parameters according to the algorithm implementation. The agent with learning controls the input towards the optimum, while the agent that does not learn controls it to non-optimal values.



(a) ROP (b) WOB

(c) RPM (d) q

Figure 5.10: Two agents in an identical simulation. One agent learns, the other does not.

5.3.1.1 Discussion

As can be seen in Figure 5.9, the agent generalizes poorly. It has not converged to a pure gradient ascent policy in the training process, and performs relatively bad on any model configuration that is different to that of the training process. In a realistic setting, this translates to poor performance when the response from drilling is not as expected. There can be several causes of this. The reward function in environment 3(Equation 4.9), which quantifies desirability of states, has one more element than the simpler environments 1 & 2. The soft constraints on input parameters encourage a quite complex policy. The agent has to learn to follow gradients in some sub-sets of the state-space, and to strictly decrease the input in other sub-sets of the state-space, while not having any knowledge of the model it tries to optimize. This gives a policy which is different from strict gradient ascent, and can contribute to bad generalization.

To combat this, an agent that learned throughout the simulation was evaluated. The agent updates the policy to increase the return, and hence ROP through the definition of

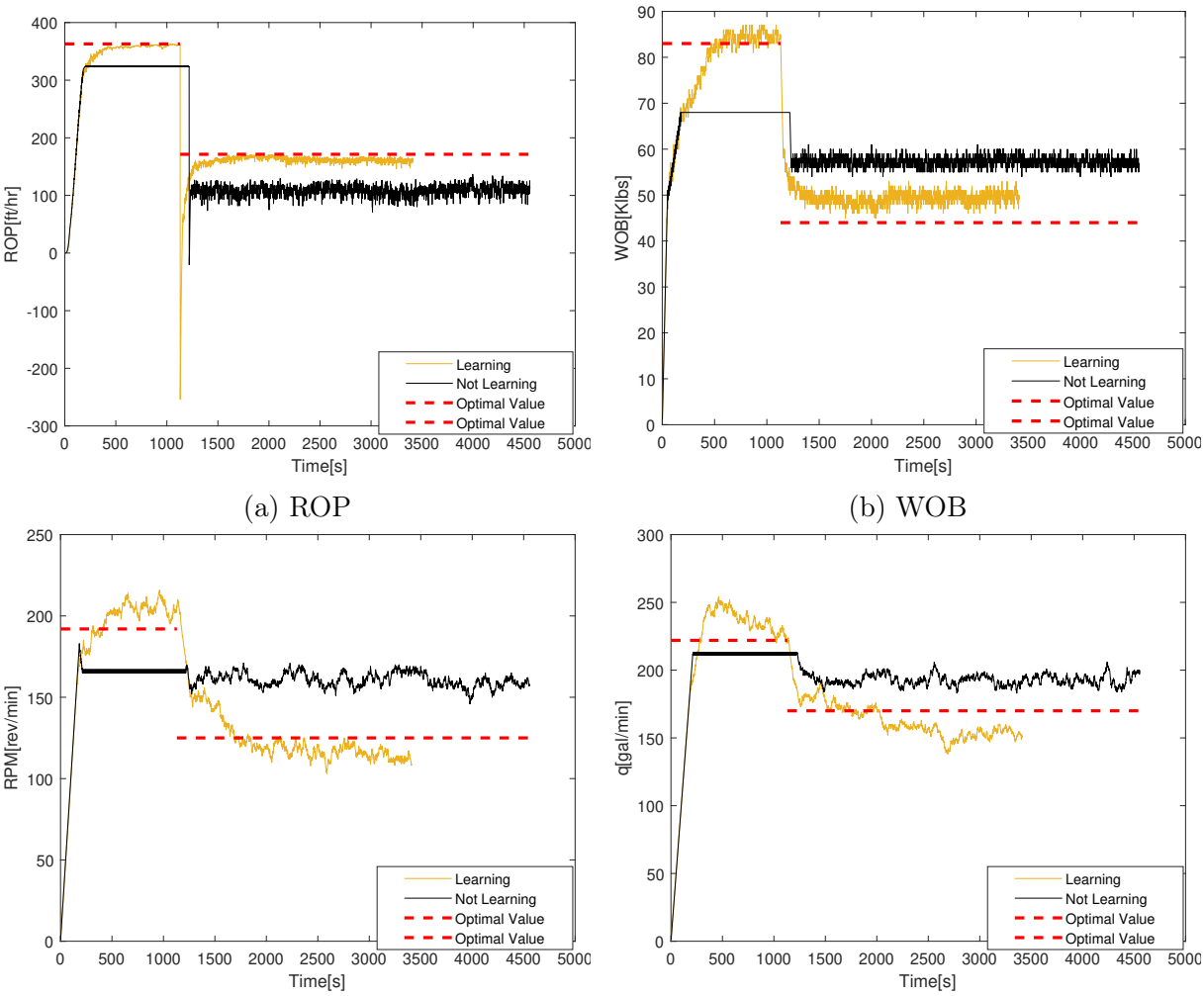
the reward function(Equation 4.9). The agents initially follow the same policy, but as the agent with learning experience negative reward, it searches towards the optimal input to maximize return. When the agent learns, it maintains some degree of exploration. The exploration is increased through the entropy regularization term in the algorithm loss function, described in section 3.8.4.2. As the agent continually explores, it never holds the exact optimum. It rather explores around the optimal region. This can be seen in Figure 5.10c and Figure 5.10d. The agent does however not completely control the input to the optimal region for WOB(Figure 5.10b). A cause of this can be that the agent manipulates all three input variable at each time-step. The memory vector, m_t in the observation vector the agent receives from the environment holds only one previous system state vector, x_{t-1}. It might not be enough for the agent to follow gradients and adjust three separate input variables. The algorithm might struggle to find a relationship between what variables cause increase or decrease in output ROP(and therefore reward), and introducing more memory in the observation vector can potentially solve this problem. In addition to this, the network architecture and hyper parameters are not tuned specifically towards the environment model, which can have huge impact. The network implementation for this agent follows the recommended stable baselines implementation described in section 4.7. Network architecture will be further discussed in section 6.3.

5.3.2 Drilling Test Case

Figure 5.11 shows a simulation where the drillability constant K changes with depth.

$$K(d) = \begin{cases} 1, & \text{if } d < 200 \\ 0.3, & d \geq 200 \end{cases} \tag{5.1}$$

The agents are initially identical, but the agent plotted in yellow updates the network parameters to continually learn. The learner controls the input parameters better than the non-learner. The plot shows the time each agent uses on the simulation. The learner finishes the drilling segment quicker than the non-learner. Note that the optimal value curves follows the time axis of the learning agent. The learning agent quickly updates the policy such that the input values approach optimum.



(a) ROP (b) WOB

(c) RPM (d) q

Figure 5.11: Drilling test case where drillability constant K changes at 200ft.

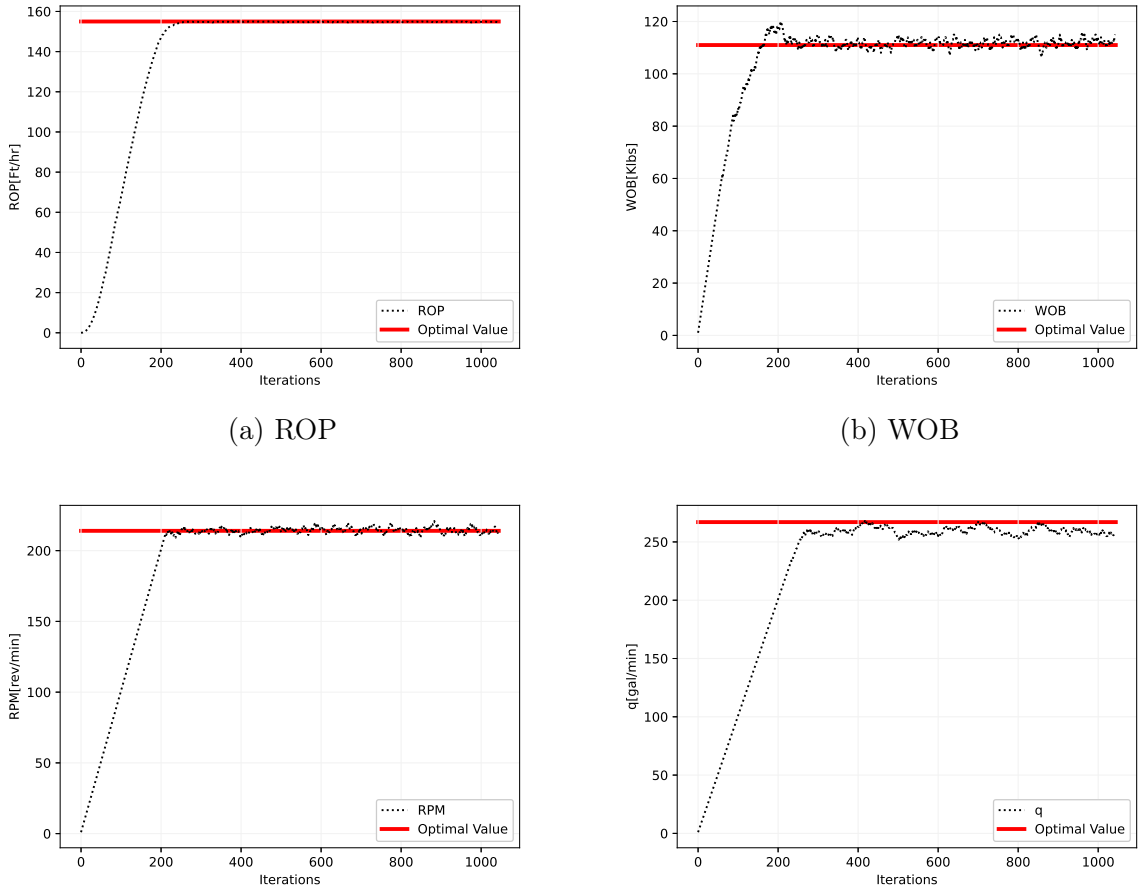
5.3.2.1 Discussion

The advantage of having the agent learn in real-time, and update the network parameters is more apparent in Figure 5.11. The agent without learning still manipulates input in a small range, but does not move towards values close the the analytical optimum. The agent that does learn however, controls the system input towards the optimum. Another property of the A2C implementation becomes apparent in this simulation. When the uncertainty in the function estimates increases, the exploration increases, through the entropy regularization. This is visible in the input plot. The agent adjusts the input according to the learned policy. When the observation is previously unseen, the input values are adjusted up and down in a region close to the optimum, to explore the state-space further. This is also visualized in the agent that does not learn. The agent picks actions from a more uniform probability distribution when the uncertainty in the model increases after the formation change at 1250 seconds. When the agent does not learn, the uncertainty in the model will not decrease, and actions are picked from a more uniform distribution over a longer period. When the agent learns however, the agent still explores,

but the overall trend of the actions are towards the optimum. The same issues as above apply here, where the network parameters are not tuned towards the model, and the agent might have too few previous measurements in the memory vector to follow gradients with three input values.

5.4 Environment 4: Bourgoyne and Young's model

Figure 5.12 shows the agent trained on environment 4. The simulation is the same test case that is used for training(Table 4.6). The agent controls the system to a value close to the numerical ROP optimum through manipulating input. The agent does not learn in this simulation.



(a) ROP

(b) WOB

(c) RPM

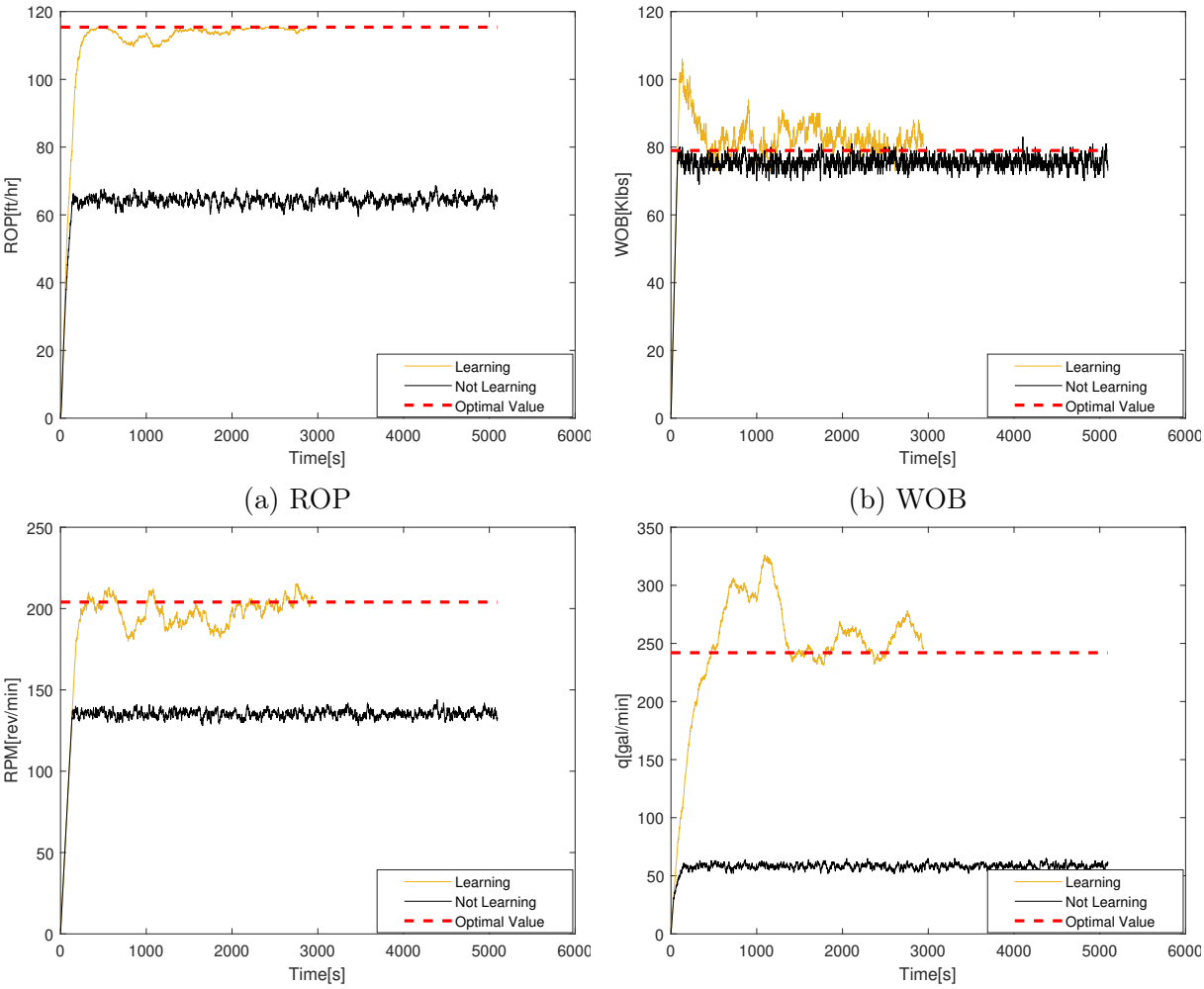
(d) q

Figure 5.12: Simulation of agent on environment 4 training case.

5.4.1 Valdiation

Figure 5.13 shows a simulation of two identical agents in environment 4. All formation specific parameters, a_1, a_5, a_6, a_8, in the Bourgoyne and Young(BY) model have been changed to parameters different from the training process. The yellow plot shows an agent that updates the parameters to continue learning. The black plot shows an agent that does not learn. The agent with learning manipulates input so that system approaches optimum, while the agent without learning controls the system towards sub-optimal values. The

agent with learning finished the drilling simulation earlier through maintaining a higher ROP.



(a) ROP (b) WOB

(c) RPM (d) q

Figure 5.13: Unseen parameters environment 4.

5.4.1.1 Discussion

The BY model that is the basis of environment 4 is one of the most comprehensive analytical ROP models that have been developed. It is the most complex model applied in this project. It has a larger degree of flexibility than the previous models, because of the four formation specific parameters. The modified implementation in environment 4 gives the possibility to adjust how the input parameters affects output, and how the output scales. Figure 5.13 illustrates a simulation where both the drillability constant a_1, and the parameters a_5, a_6 and a_8 are adjusted to change both the input-output relationship and the scaling of ROP from the training case.

The policy the agent learns in this environment is more complex than the previous environments. The agent has to learn gradient search in some sub-sets of the state-space,

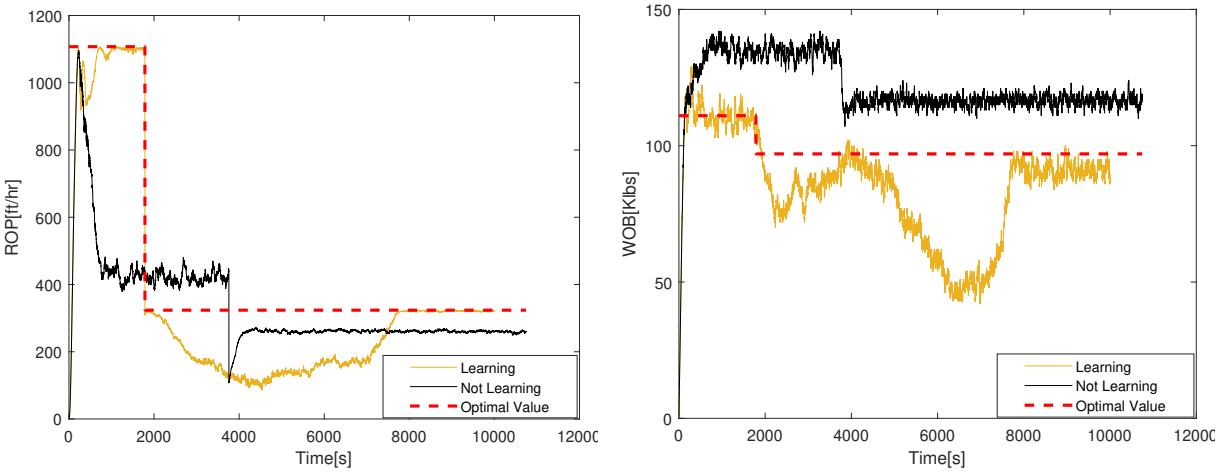
while staying within the soft constraints enforced through the reward function. In addition to this, the model is forced to be zero for all non-negative values for ROP. This leads to some sub-sets of the state-space resulting in actions not changing ROP where the model is zero, and naturally there is no gradient to follow.

As environment 4 has a larger memory vector, m_t, that holds the three previous system states. The agent seems to be more able to relate the effect of the individual input variables to ROP because of this, even though the model is more complex. The agent still applies relatively large adjustments to the input, but it trends towards the optimum. This variation can be a case of parametric bias in the artificial neural network, introduces through too many or too few artificial neurons. This will be further discussed in section 6.3.

As can be seen from Figure 5.13, the agent controls particularly WOB far above the optimal value in the initial phase of the drilling segment. This is in reality not something that is desired. Excessive application of input can cause oscillations, or other drilling dysfunctions according to the study by Dupriest(section 2.2.1). This can cause irreversible damage to the bit, which reduces the contact area between the bit and the rock formation. In turn, this reduces ROP according to models like that of Hareland and Rampersad(section 2.3.1). The use of excessive input in this case can be a cause of slow learning through badly designed ANNs. As the model is more complex than in the previous environments, with a larger number of elements in the observation vector, one can imagine that different network architectures should be used for different environment structures.

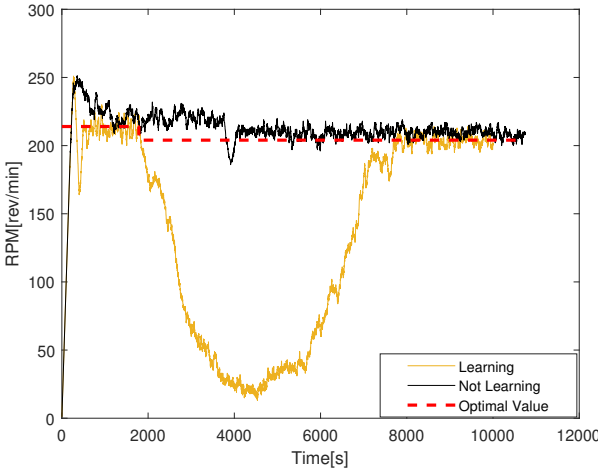
5.4.2 Drilling Test Case and Experimentation

Figure 5.14 shows a simulation where two initially identical agents are interacting with a drilling case constructed from environment 4. The BY formation specific parameters change at a set depth. The trajectory in yellow represents an agent with learning enabled, while the trajectory in black represents an agent without learning. The change in model parameters is designed to give a large step in ROP through the drillability constant a_1, while the optimal values for the input only change slightly, through small changes in a_5, a_6 and a_8. Figure 5.14 shows that the agent without learning controls the input to a sub-optimal state. The agent that learns does a search in the state-space before it controls the input towards the optimum after approximately 8000s. The agents finish the drilling simulation within 2000s of each other as a result of this.

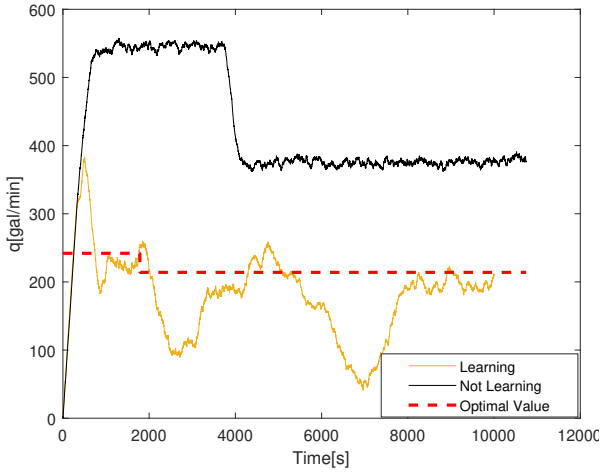


(a) ROP

(b) WOB



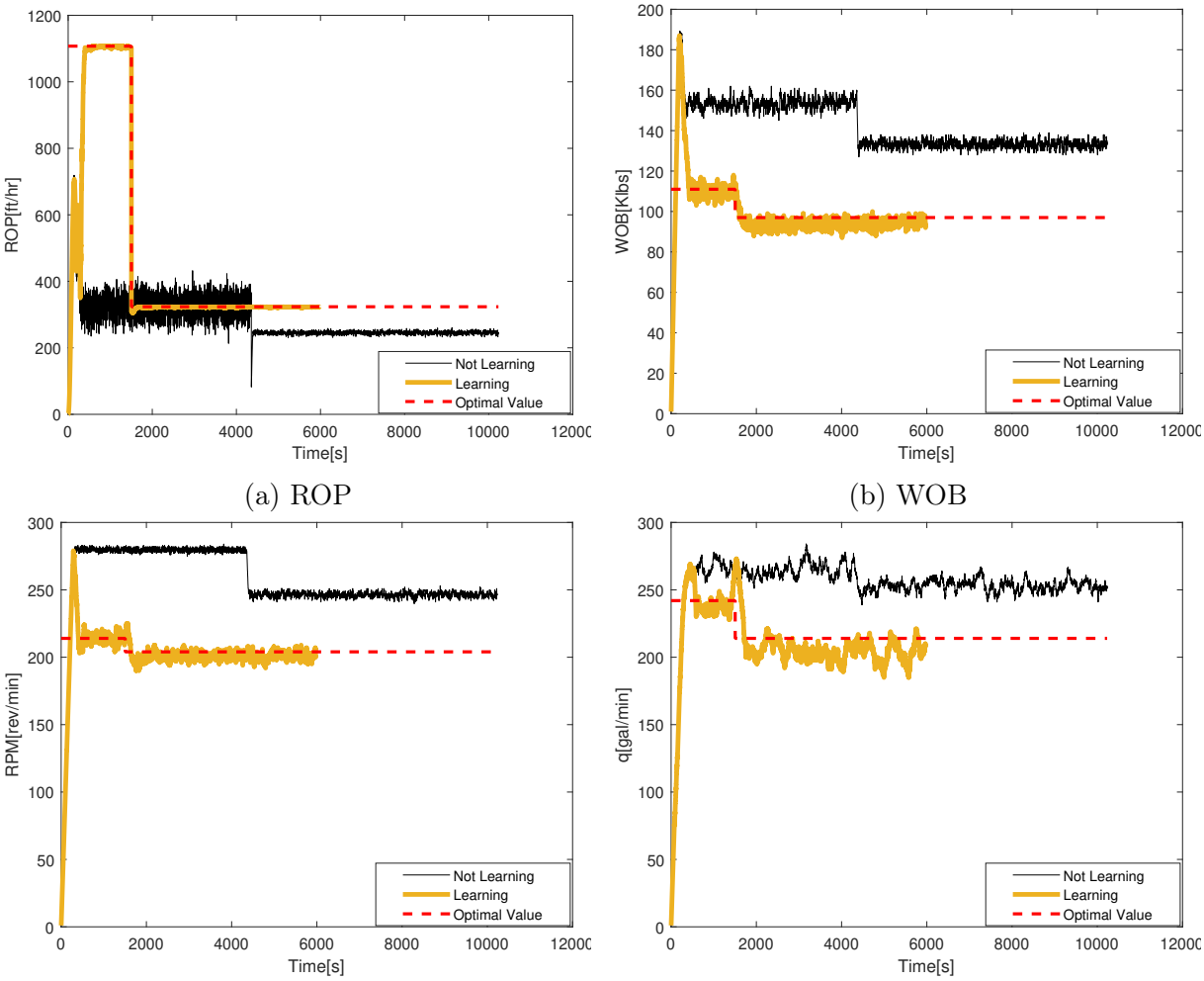
(c) RPM



(d) q

Figure 5.14: Drilling test case where all four formation specific constants change at 400ft.

Figure 5.15 shows the same simulation case as Figure 5.14. The only difference is a changed network architecture for the agent. The networks are changed to better reflect the number of output nodes, with a policy network with two hidden layers of 64-16 neurons and a value function network with three hidden layers of 128-64-32 neurons. The agent learns more efficiently before and after the parameter change, resulting in a finishing time around 6000s. This is significantly quicker than the non-learning agents and the learning agent of Figure 5.14, and is a result of maintaining a higher ROP through faster learning.



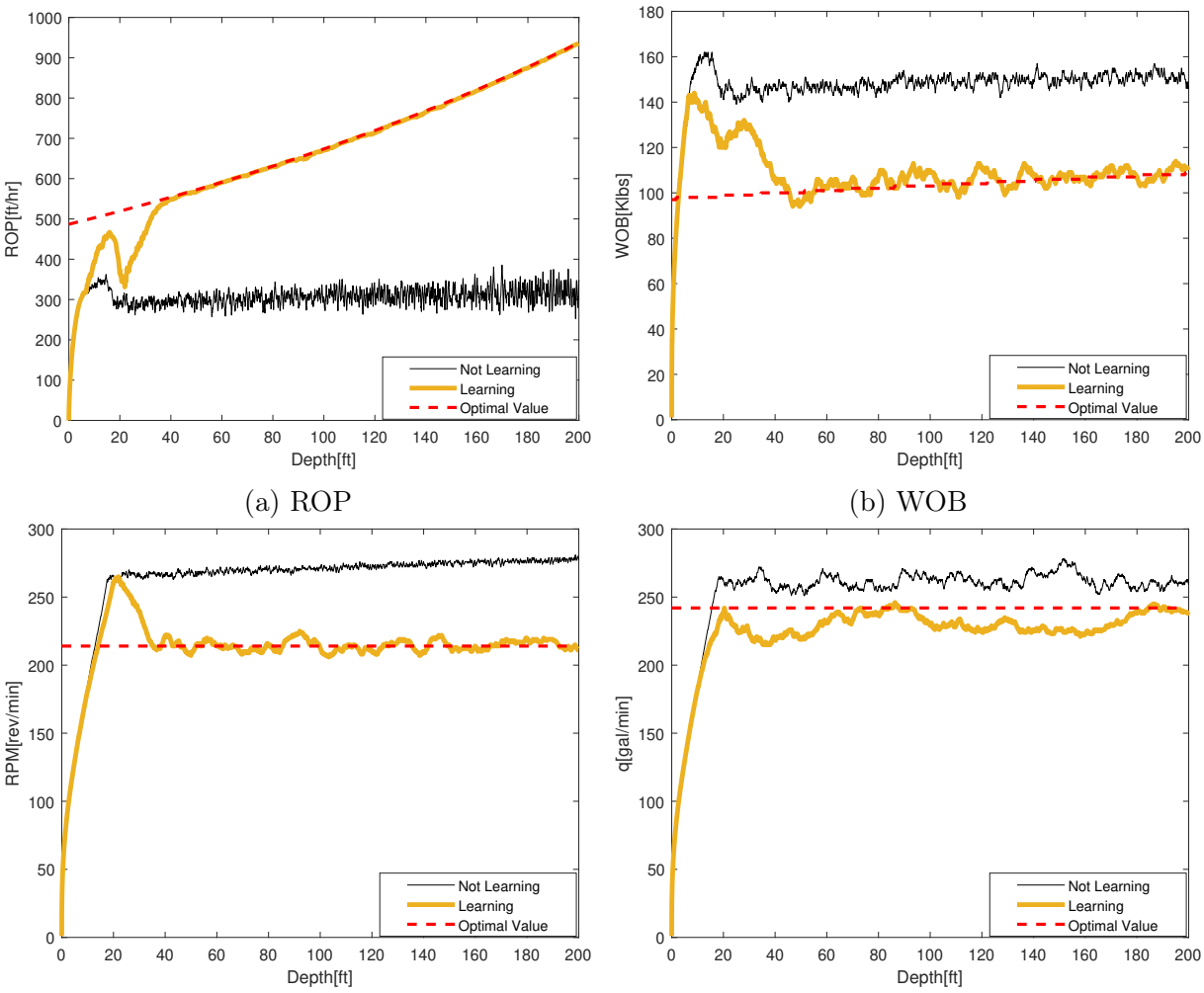
(a) ROP (b) WOB

(c) rpm (d) q

Figure 5.15: Tuned network architecture.

Figure 5.16 shows a simulation of environment 4 where the coefficient that decides the input-output curve for weight on bit(a_5) varies with depth. a_5 as a function of depth is given by Equation 5.2. The agent follows the trend of optimal WOB. The agent uses the adapted network structure described above.

$$a_5(d) = 0.75 + \frac{d}{1000} \tag{5.2}$$

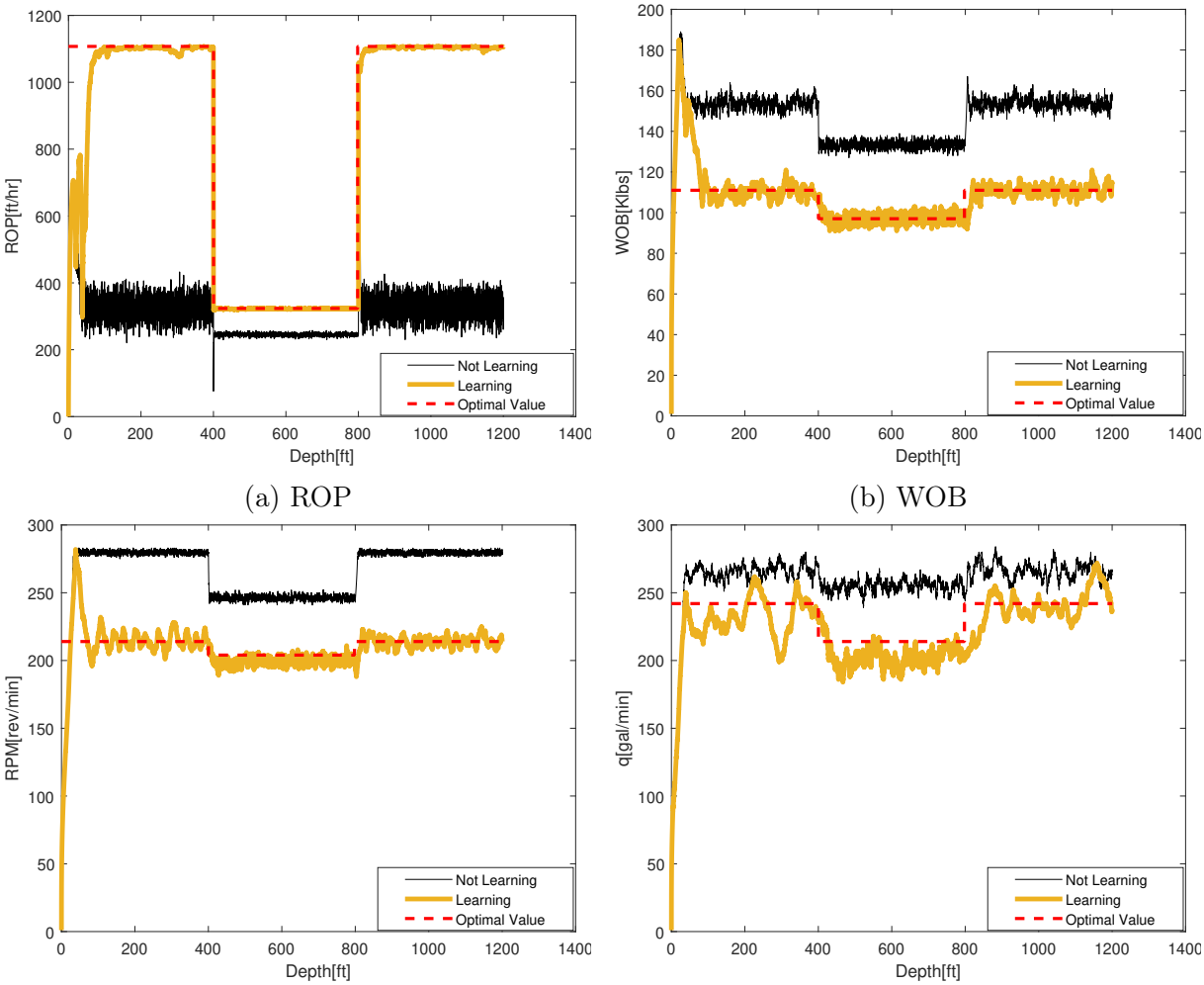


(a) ROP (b) WOB

(c) RPM (d) q

Figure 5.16: WOB-ROP interaction coefficient varies with depth.

Figure 5.17 shows a simulation that is similar to Figure 5.15, but the simulation has one more drilling segment. The last drilling segment has model parameters identical to the first drilling segment. The purpose of this test is to see whether the agent performs better the second time it encounters a given formation. The plot also follows the depth on the first axis to better visualize the the difference with and without learning.



(a) ROP (b) WOB

(c) RPM (d) q

Figure 5.17: Encounter of a previously seen formation during drilling.

5.4.2.1 Discussion

The simulation in Figure 5.14 is designed to be difficult for the agent to handle. After the formation change at 400ft the drillability coefficient a_1 is scaled significantly, while the input specific coefficient only change slightly. As a result of this, the optimal output ROP is reduced to about one third, while the optimal input values only shift by a few percentage. This is a challenge because of the large decrease in the estimated value functions, and subsequently advantage function. This affects the policy, which can get large parameter updates through Equation 3.33. Sudden large parameter updates make the overall training unstable.

The learning agent in figure 5.14 spends a significant amount of time adjusting the input values, and as a result of this finishes the drilling segment almost at the same time as the non-learning agent. This highlights that the agent in fact learns inefficiently. The agents in Figure 5.14 use the recommended network architecture of the stable baselines library of two separate networks with identical structure of two hidden layers of 64 neurons.

Figure 5.15 shows the difference in learning when the network architecture is more tuned towards the size of the observation and action vector. The tuned network architecture consist of:

- Value function network: Three hidden layers with 128, 64 and 32 artificial neurons.
- Policy network: Two hidden layers with 64 and 16 artificial neurons.

All nodes in the network use the hyperbolic tangent activation function(Equation 3.23b). During experimentation, other network architectures were tested. Smaller networks with two layers of eight neurons diverged during training, and larger network structures generalized poorly and introduced unstable learning. This will be further discussed in section 6.3. The reason for the asynchronous network architecture is the difference in the output layers in the network. The policy network outputs three values, making up the action vector. The value function network however, outputs a vector corresponding to the size of the observation vector of 16. As shown in Figure 5.15, the agent learns quickly and controls the input towards the optimum much quicker than the agent with the standard network implementation shown in Figure 5.14. The agent with the adapted network finishes about 4000s quicker through stable learning, and maintaining a higher ROP. The agent still initially spikes both RPM and WOB past the optimum, even though the agent learns more efficiently. This suggests that the reward functions needs a term that models the cost of using input, to prevent excessive use of input.

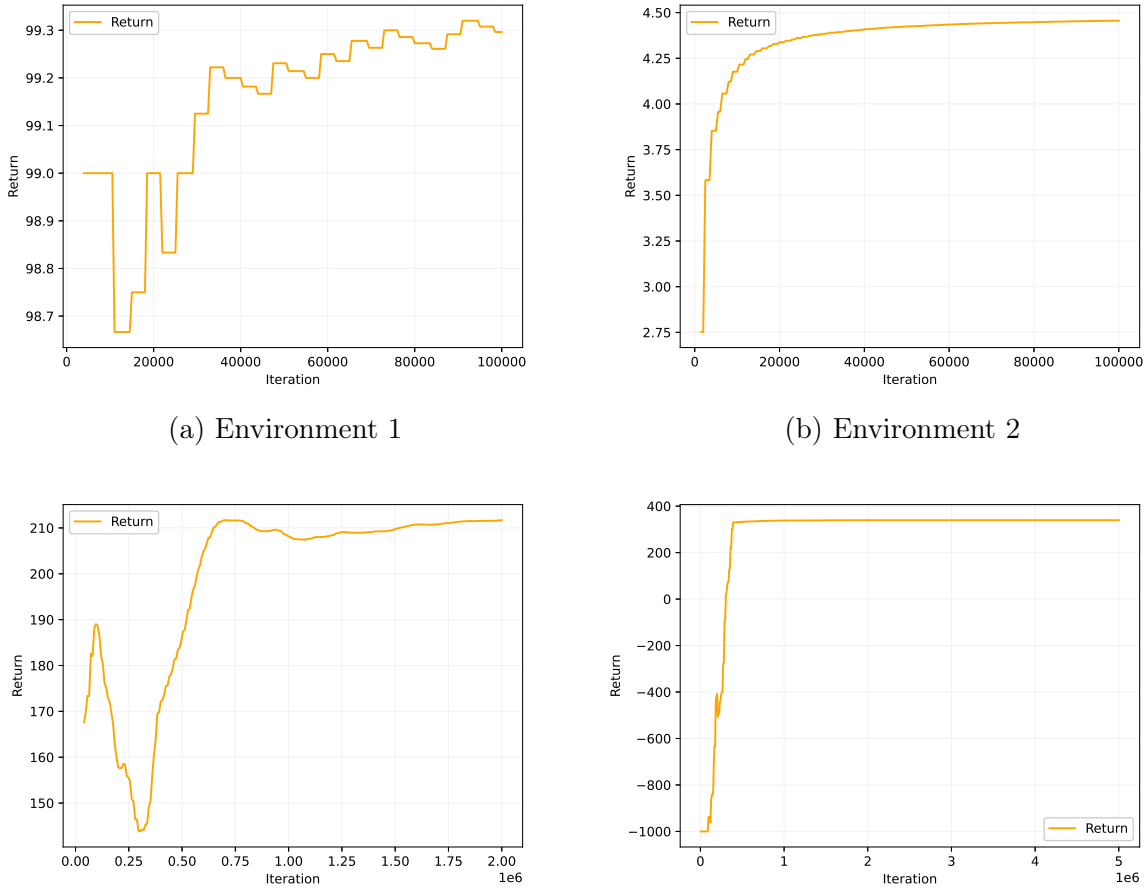
All simulations that are analyzed so far in this thesis is based on formations that change spontaneously. Figure 5.16 shows a case where the WOB coefficient a_5 changes according to Equation 5.2, giving a constant change optimal WOB. Even though the ROP models applied in this thesis assume constant model parameters for a given rock formation, one can imagine that in reality, a rock formation is rarely uniform. The hardness of the rock can vary within the formation itself, and the agent shows it is capable of learning this real-time. This simulation could also be used to illustrate the effect of depth on ROP directly. As mentioned in section 2.3.1(Batanee et al.), ROP decreases with increasing depth. This effect is removed from the BY model implemented in environment 4 as a simplification measure. Figure 5.16 shows that the agent is capable of following non-stationary optimal values over a drilling segment. This is essential both in realistic modeling of formation hardness, and to be able to follow an optimum that is dependant on depth.

Figure 5.17 visualizes that the agent performs better the second time it encounters a formations. The third drilling segment is identical to the first, and the large spike in input is eliminated in the encounter. This suggests that the agents perform better as they learn. It is however interesting to investigate how an agent would perform on a known formation after many simulations on different parameters. A solution where the network parameters are saved every time a new formation is encountered could be implemented. The algorithm

could then fetch these parameters if the formation is previously drilled.

5.5 Convergence

Figure 5.18 shows the convergence of each agent trained on the respective environemnts in this project. Figure 5.18a shows the training of environment 1. It can be seen that the agent achieves almost the maximum obtainable reward($G_{max} = 100$) in one training episode. The other environments require more training before reaching their maximum reward respectively. The first two environments converges steadily, as they learn to follow gradients. Environment 3 and 4 has the inclusion of soft constraints on rewards, and can achieve negative reward. As the model in environment 3 can be negative, the agent can always find a direction that improves ROP, and this results in all episodes leading to positive rewards. Environment 4 has sub-sets of the environment that have no gradients, and as a consequence of this, the agent receives the minimum possible reward for the first few iterations. After approximately 1000000 iterations, the agent converges.



(a) Environment 1

(b) Environment 2

(c) Environment 3

(d) Environment 4

Figure 5.18: Convergence of environments 1-4 in training process.

6 | Further Discussion

This chapter presents further discussion of results, implementation and the project in general. The chapter is structured in the following way:

- Section 6.1 discusses the implemented solution.
- Section 6.2 discusses simplifications in the models implemented in this project.
- Section 6.3 discusses choice of algorithm, network design and algorithm performance based on network design.
- Section 6.4 presents future work the author would like to investigate further.

6.1 Solution Method

As presented in section 2.3.3, research of optimization of drilling rate can be split into two categories:

- Pre-optimization on an ROP model in advance of operations.
- Real-time optimization of ROP based on data available while drilling.

Reinforcement learning is a flexible framework, that can be tailored to either of these approaches through design of the reward function and environment design for offline training. Optimization in advance of operations require a model that is precise enough to recreate an environment that the agent can train on. Figure 5.1, Figure 5.5, Figure 5.8 and Figure 5.12 show plots where the agent interacts with simulations where the environments are identical to what the respective agents are trained on offline. The models have increasing complexity, and the RL agents handle these cases well. However, research shows that modeling the drilling system is one of the major challenges in ROP optimization, as the phenomena that affect ROP is not fully understood. In addition to this, it is argued by several authors that the data from well drilling is location and condition specific. This limits the usefulness of data driven modeling such as ANNs and other machine learning algorithms for ROP prediction with the purpose of accurately forecasting new drilling segments and formation types. This limits the applicability of training an RL agent of-

fline in advance of operations, with the intention of teaching an RL agent the optimal sequence of input decisions. However, as shown in section 5.1 and section 5.2, RL agents can generalize well to previously unseen model configurations. If the drilling process could be modeled in such a way that a learned policy is valid for all model configurations, an agent could be trained on one model configuration, and optimize a sequence of other configurations without policy updates based feed-back from the environment.. This is seen in Figure 5.4 and Figure 5.7. According to the drilling curve presented by Dupriest, the input-output relationships in the ROP model is convex, and if that is valid for all rock formations, a similar approach could be used. Additionally, the reward function design is the deciding factor in shaping the policy. This is because the value function is the estimated expected cumulative discounted reward, and this is used to evaluate agent actions. Clever reward function design can contribute in making the RL agent more robust to unseen model configurations.

The agents trained on environment 1 and 2 learn a universal policy based on accurate modeling in the training process. In reality, this would be a large challenge, and a lot of research has gone into universal modeling of ROP without any major success. In the experiments done with environment 3 and 4, the agent generalizes poorly to unseen model configurations. To combat this, real-time learning based on feedback from the environment is introduced. As seen in section 5.3 and section 5.4, the agent learns quite efficiently, and converges towards the optimal solution for several different test cases. The agents have a pre-trained policy based on a single parameter configuration of the ROP model. The more accurate this training environment is, the more accurate the initial policy of the agent will be. This is a way of introducing knowledge about the system in advance of operations, as it will limit the time the agent has to explore to discover better policies in the initial drilling. However, as can be seen from Figure 5.13, the agent also handles large errors in initial policy. This is visualized through the non-learning agent plotted in black, who manipulates input to a sub-optimal solution. The agent that uses the feedback from the environment to learn however, approaches the optimum after some initial exploration.

The Bourgoyne and Young ROP model is quite flexible by construction. It can be modified through changing the constants a_{1-8}. The model has been used for optimization of drilling rate in several studies. The studies often revolve around finding the model coefficients that give the most precise model compared to real drilling data, and optimizing the model in advance. In a survey done by Soares et al(section 2.3.1), a simplified Bourgoyne and Young model was found to be the most accurate analytical model out of a selection of other models. The modified model implemented in this thesis(section 4.6) is similar to that of Soares, and still retains flexibility through the ability of changing the input-output relationship with a_5, a_6 and a_8. In addition, a_1 scales the output. This allows for testing on a model that varies significantly with changing the given parameters. The results

from experimentation with this model shows that the RL agent is capable of learning ROP optimization with changes in model parameters in real-time. Additional plots from experimentation on the BY model can be found in appendix A.3.

The primary driving factor which the agent evaluates is the reward signal. In this project, the reward function is implemented as a simple function of the change in ROP. In addition, penalties for exceeding the input limits are implemented in environment 3 and 4. The simple reward functions worked for the purpose, which was to approach the maximum ROP for a given model. Initially, a term for quick convergence was present in the reward function. This was implemented as a numerically large reward signal at the end of an episode, which decreased with the number of seconds the agent used on the drilling segment. This sparse and large reward did however only make training more unstable, and generalization worse. Sparse rewards require more future planning, and in turn makes learning slower. The reward function can be formulated to reflect more comprehensive objectives in drilling. It could feature bit-wear penalty, and the cost of applying input. One could also directly model the monetary cost of drilling in the reward function. An objective similar to that of Abughaban et al.(section 2.3.3, equation 2.11) could be formulated, where depth of cut, drilling specific energy, vibrations and other metrics decide the reward signal.

The agent learns different policies for the different environments. The policy the agents learn in environment 1 and 2 is valid for all model configurations. Because of the memory in the observation vector, the agent always maps a change in ROP to either an increase or decrease in input. This policy is valid for the entire domain of parameter configurations. However, in environment 3 and 4, the agents learn a policy that does not transfer between model configurations. This suggests that the agents do not learn a pure gradient ascent policy. With real-time learning however, the necessity for the agent to learn the optimal policy in initial training is not absolute. This is demonstrated through the agents evaluated in environment 3 and 4 continually updating the policies towards the optimal policies. The results from environment 3 suggest that the number of past system states in the memory vector is important. As the agent in environment 3 only has one past system state to base the decision on, it seems unable to manipulate three separate input values towards the optimum. When more past system states is introduced in the memory vector of environment 4, the agent seems more able to map output to change in the correct input.

Real-time optimization of the drilling process can be solved by model-free, data-driven approaches. This is demonstrated through promising implementations of for example the extremum seeking(ES) algorithm(section 2.3.3), and other data driven gradient approaches, like the Intelligent Drilling Advisory System(section 2.3.3). The ES algorithm explores the state-space through perturbation with a sine-like function. It then follows

gradients to optimize ROP or specific energy. A natural question would be why RL could be preferred as a real time optimizer compared to similar solution methods. Exploration in ES can be inefficient and slow in high dimensional spaces, while deep RL algorithms utilize ANNs that specialize in high dimensional function approximation. RL methods also provide several customized ways of handling exploration, and a variety of different algorithms that suit different problem types. In addition, the reward function of RL makes for a flexible framework for objective definition. Hard constraints can be implemented in RL algorithms through defining valid action and state spaces. This can be more difficult in ES. The ES algorithm might be more robust and sample efficient. The challenge with machine learning algorithms, and particularly algorithms that feature ANNs, is that they require large data samples, and can be non-robust to changes. However, the RL implementation analyzed in this thesis has been robust within the bounds of the conducted testing. An advantage of learning algorithms is that they continue to learn, and will get better and better as they are applied. Another strength of RL algorithms is that they plan for future states, and can weight long term benefit against short term reward. This can be an advantage if the function for optimization has local optimums. As an example, chess engines based on reinforcement learning, that continually learn during chess matches with real players learn the players characteristic strategies, and can optimize a policy based on this. That would be hard to model in other optimization algorithms. An interesting extension of this work would be to compare data-driven optimizer such as the ES algorithm to a continually learning RL agent as implemented in this project.

6.2 Simplifications

Several assumption have been made in the ROP models and environment implementation in this project. None of the models feature any dynamics. The implementation make the assumption that the sampling rate is large enough such that the dynamical behaviour of the ROP model has settled between measurements. In reality, input changes would induce oscillations, and the system would not change the input instantaneously as the current implementation does. However, the Bourgoyne and Young model have been used in several drilling optimization studies, where satisfactory precision for a given well is met.

The model does also not feature any noise. For a realistic implementation, noise filtering would be necessary. Additionally, the frequency of which the agent interacts with the environment would have to be large enough to measure changes in the states with noise present.

Another simplification that is made in this project is the properties of rock formations. The implemented ROP models assume constant rock properties for a given rock. The

model reflects this through constant formation specific parameters for the given rock. When the rock formation changes, these parameters change. In reality these rock properties can vary with depth. A single rock formation can as an example be softer on the edges and harder towards the middle. Figure 5.16 shows that the RL agent is capable of adjusting the input to compensate for constantly changing rock properties, where the WOB interaction coefficient $a_5 = 0.75 + \frac{depth}{1000}$ varies with depth.

As mentioned above, the models implemented in this thesis also assumes individual optimal values for the input. Surface plots of mechanical specific energy from some articles support this. In reality however, these optimal points could be dependant of each other, which introduce a multiple of local maximums in the ROP model. One could however assume that local optimal values are in a region that achieves satisfactory performance.

A realistic drilling case also has more constraints than featured in this implementation. Hard constraints would be needed on the input, in addition to pressure constraints for well stability purposes. These should not be modeled as soft constraints through reward, as the penalty on input implemented in this work, but hard constraints in the algorithm. Particularly pressure constraints are a limiting factor in a realistic drilling case.

6.3 Algorithm and design

In this project, the A2C algorithm was chosen for a combination of reasons, which will be outlined in this section. When choosing a DRL algorithm, there are many considerations to take. The A2C and A3C algorithms are more efficient on consumer hardware than its competitors, due to the parallelization of agents, which allows multiple instances running at once. This also completely replaces the replay buffer of the DQN algorithm, which saves computational resources. The A2C algorithm is also flexible, as it supports both discrete and continuous action spaces. The DQN algorithm for example only supports discrete action spaces. It is also an advantage to use an on-policy algorithm like the A2C for real-time optimization, as the agent has to approach optimal behaviour. When the agent learns real-time, it is not desirable to follow another policy than the optimal policy the agents tries to estimate. This can lead to unnecessarily much exploration as the drilling progresses.

The drilling problem that this algorithm aims to solve has no finite ending time. One can assume that the agent finds a non-zero value for ROP that eventually leads to a terminal depth, but there is no guarantees. Methods that are based on MC sampling of full returns is not ideal. Pure gradient based methods are therefore not the most robust option, as they use experienced return without bootstrapping. If this experience never occurs, the agent will not learn. The A2C algorithm uses an n-step temporal difference learning update, which is flexible as the number of steps the algorithm takes before bootstrapping

can be adjusted.

Another challenge in the drilling problem is the sudden model changes that occur when the drilling segment enter a new formation. This leads to a large change in the value function estimate, when suddenly the measured ROP jumps up or down. In pure value based methods, these large jumps in value function estimates make learning unstable, as the expected return suddenly change. This leads to large policy updates. Brief experimenting with the DQN algorithm confirmed that the algorithm did not converge towards an optimal policy. It in fact did not converge towards a policy that resulted in non-negative ROP.

Large changes in the value functions can also cause problems for actor-critc methods. Actor-critic methods often scale the policy update with a value-function estimate. This problem also occured in the DDPG algorithm, that adopts the Q-learning step of DQN, and uses this to scale the policy update. The policy update steps became too large, and the DDPG algorithm did not learn a policy that improved initial performance. There exists several methods for bounding the policy update in actor-critic methods. These include clipping and kl-regularization, which both essentially limits value function estimate impact on the policy update step.

The A2C algorithm, and advantage actor critic methods in general, uses the advantage function for this purpose(Equation 3.30a). This is different from other methods who use the action-value function(Equation 3.5) or state-value function(Equation 3.4). The advantage function is the action-value function subtracted the state-value function. This limits the size of the advantage function, and helps in preventing too large update steps because of exploding or diminishing value functions. In addition to this, the A2C algorithm from stable baselines implemented in this project normalizes the advantage function, which acts as an additional bound in the update step.

When designing the artificial neural networks, or network architecture for short, there are many parameters to consider. The size of the network, the type of network, and hyperparameters are the most essential. Recurrent neural networks carry an internal state that essentially introduces memory. Convolutional neural networks work well for problems with grid-like topology. This includes time series of measurements. Properties like these can introduce more robustness to the RL algorithm. Both recurrent neural networks and convolutional neural networks tend to perform better than feed-forward neural networks on dynamical systems. These networks are however more complex, and require more design and attention. The feed-forward neural networks are simpler to implement, and modify. Because of this, the feed forward NNs were chosen in this project.

Intuitively, one would think that different network architecture should be used for the policy and value networks, as they produce output of different dimensions. The standard

stable baselines implementation uses a set network architecture of two hidden layers with 64 artificial neurons. This number seems arbitrary, but works quite well for the simpler problems. This network architecture is used for all results from environment 1,2 and 3. However, as can be seen from Figure 5.14, when the model changes in a particular way the agent struggles to learn the new optimal values. The simulation is designed to drastically reduce the expected maximum return through lowering the optimal ROP, while maintaining optimal input values close to the initial formation. The default network architecture learned this new optimum quite slow, as the large changes in the advantage function changed the policy by a large amount. In reality, the policy only needs a slight change. When the networks were redesigned to better reflect the number of output units the separate networks have, the agent learned more efficiently. As the policy network only outputs 3 values, being the action vector, and the value function network outputs 16 values, it is intuitive to have more artificial neurons in the value function network. A structure of the policy network π was found through testing to be two hidden layers of 64 and 16 neurons. For the value function network V, a structure of three hidden layers of 128, 64 and 32 neurons was found to be best. Network structure with fewer neurons than this learned slowly, and much larger network structures generalized poorly and was unstable during learning. Network architecture design is based a lot on testing, but as a rule of thumb, the width of a layer should be a number in the series 2^n. Figure A.8 shows a plot of a simulation where the algorithm has two identical networks with two layers of eight artificial neurons. The networks does not have enough neurons to represent the policy parameterization efficiently. The agent does not learn an optimal policy. Figure A.7 shows another experiment with a network of two layers with 512 artificial neurons. The agent has an oscillatory behaviour, and the network could be overfitted, which is introduction of large parametric bias through too many neurons for the function it tries to approximate.

Choosing the optimization algorithm for the problem is also important. The optimization algorithm essentially decides how the network parameters are updated with the gradients from the RL algorithm, and can cause learning to be either unstable or too slow. If the initial steps taken by the optimizer algorithm are too large, the RL agent will struggle to learn the desired behaviour. If the steps are too small, the agent will learn slowly. The SGD algorithm(algorithm 1) was initially tested for simplicity, but the because of the large variance from the assumption made in Equation 3.26(algorithm picks one gradient at random), the A2C algorithm did not learn at all. The default optimization algorithm for the stable baselines implementation is the Adam optimizer. It is a specialized optimizer for ANNs, that combine adaptive learning rates with momentum. This did however also make training unstable. To combat this, the RMSprop algorithm(algorithm 2), which were used for the A2C and A3C in the original implementations was implemented. This stabilized training, and performed remarkably better than the adam optimizer.

6.4 Future Work

An interesting extension of this project is to adapt the agent and environments to evaluate the mechanical specific energy(MSE) concept(section 2.3.2). The MSE concept is more versatile in real-time optimization than the ROP models utilized in this work. The challenge is however that the MSE model is dependant on ROP as a measurement. A real-life simulator or laboratory scale drilling rig could be used to simulate ROP, or generate data used for training. As the author of this thesis had no access to such tools, and limited time, this remains an interesting future work. The formulation of the MSE forumla also makes it more universally applicable, as it takes ROP as an input.

Another approach could be to collect data from previously drilled wells. Data from one well could be used to generate a model for an environment to train the agent in. The agent could then be evaluated as it learns to optimize drilling based on data from another well with real-time learning. This will give good indication of efficiency and capabilities in the RL as a real-time optimizer for drilling through real-time learning.

This thesis has focused on model-free RL algorithms. Implementation of a model-based RL algorithm could be beneficial, even if the model is not a perfect representation of the drilling process. Anything the RL agent can use to plan or simulate experience online could improve sample-efficiency and overall learning capabilities.

It would also be interesting to investigate the sample efficiency of RL with real-time learning as an optimization method compared to other data driven optimization techniques such as the extremum seeking algorithm.

One important point that is not implemented in this work is hard constraints. Hard constraints needs to be implemented, both for input and pressure constraints for well stability. The models utilized in this thesis did not have a connection to pressure, and it was neglected from the work. Pressure constraints are however important for well stability, and is a limiting constraint while optimizing ROP.

7 | Conclusion

Optimization of rate of penetration(ROP) is a can pose a challenge for several reasons. First and foremost, the phenomena that affect ROP are not fully understood. This makes ROP modeling difficult, which limits the usefulness of optimization of operational parameters in advance of drilling. Research indicates that parameters for the ROP model has to be uniquely defined for specific formations and drilling operations. Real-time optimization of ROP is emerging as the most feasible solution.

Implementing a model-free reinforcement learning(RL) agent that is pre-trained on one model configuration, and allowing it to continually learn during operations is a possibility. It combines knowledge one has about the system in advance of drilling, with the possibility of continually improving drilling rate through learning from measurements, even though the actual system model is unknown. The A2C algorithm is a good fit for the purpose. The RL agent shows promise on the most comprehensive ROP model implemented in this thesis, the Bourgoyne and Young model. The agent handles validation cases where model parameters change with step functions, and when model parameters gradually change with depth. These are fundamental properties for ROP optimization.

The models implemented in this thesis are however simplified through ignoring dynamical behaviour and noise, amongst other things. These are challenges in a realistic setting. Whether an RL autodriller would be more efficient and robust than other existing solutions cannot be definitely concluded from the work done in this project. Testing with more realistic simulators, and/or data from previously drilled wells would be needed to analyze this. However, the RL autodriller concept shows promise as a method to implement a model-free, data driven optimization algorithm, with a large degree of flexibility towards objective, exploration, and future planning.

Bibliography

[1] Antonin Raffin et al. *Stable Baselines3*. https://github.com/DLR-RM/stable-baselines3. 2019.

[2] Adam T. Bourgoyne et al. *Applied Drilling Engineering*. First Printing. Richardson, TX: Society of Petroleum Engineers, 1986.

[3] Cesar Soares and Kenneth Gray. "Real-time predictive capabilities of analytical and machine learning rate of penetration (ROP) models." In: *Journal of Petroleum Science and Engineering* 172 (2019), pp. 934–959. DOI: https://doi.org/10.1016/j.petrol.2018.08.083. URL: http://www.sciencedirect.com/science/article/pii/S0920410518307563.

[4] Chirant Hedge et al. "Rate of Penetration(ROP) optimization in drilling with vibration control." In: *Journal of Natrual Gas Science and Engineering* 67 (2019), pp. 71–81. URL: http://www.https://www.sciencedirect.com/science/article/pii/S187551001930112X.

[5] Eren Tuna. "Real time optimization of drilling parameters during drilling operations." PhD thesis. Middle east technical university, 2010.

[6] W.C Maurer. "The "Perfect-Cleaning" Theory of Rotary Drilling." In: *Journal of Petroleum Technology* (1962), pp. 1270–1274.

[7] Kenneth K Landes. "How rock properties are related to drilling." In: *Oil Gas Journal* (1964), pp. 94–101.

[8] *Maximizing Drill Rates with Real-Time Surveillance of Mechanical Specific Energy*. Vol. All Days. SPE/IADC Drilling Conference and Exhibition. SPE-92194-MS. Feb. 2005. DOI: 10.2118/92194-MS. eprint: https://onepetro.org/SPEDC/proceedings-pdf/05DC/All-05DC/SPE-92194-MS/1835404/spe-92194-ms.pdf. URL: https://doi.org/10.2118/92194-MS.

[9] Sanjit Roy and G.A. Cooper. "Prevention of Bit Balling in Shales: Some Preliminary Results." In: *SPE Drilling & Completion* 8.03 (Sept. 1993), pp. 195–200. DOI: 10.2118/23870-PA. eprint: https://onepetro.org/DC/article-pdf/8/03/195/2088206/spe-23870-pa.pdf. URL: https://doi.org/10.2118/23870-PA.

[10] John R. Eckel. "Microbit Studies of the Effect of Fluid Properties and Hydraulics on Drilling Rate." In: *Journal of Petroleum Technology* (1968), pp. 541–546.

[11] G. Hareland and P. R. Rampersad. "Drag - Bit Model Including Wear." In: *Society of Petroleum Engineers* (1994).

[12] H.R. Motahhari, G. Hareland, and J.A. James. "Improved Drilling Efficiency Technique Using Integrated PDM and PDC Bit Parameters." In: *Journal of Canadian Petroleum Technology* 49.10 (Oct. 2010), pp. 45–52. DOI: 10.2118/141651-PA. eprint: https://onepetro.org/JCPT/article-pdf/49/10/45/2147023/spe-141651-pa.pdf. URL: https://doi.org/10.2118/141651-PA.

[13] Jr. Bourgoyne A.T. and Jr. Young F.S. "A Multiple Regression Approach to Optimal Drilling and Abnormal Pressure Detection." In: *Society of Petroleum Engineers Journal* 14.04 (Aug. 1974), pp. 371–384. DOI: 10.2118/4238-PA. eprint: https://onepetro.org/spejournal/article-pdf/14/04/371/2157414/spe-4238-pa.pdf. URL: https://doi.org/10.2118/4238-PA.

[14] H.I. Bilgesu et al. "A New Approach for the Prediction of Rate of Penetration (ROP) Values." In: SPE Eastern Regional Meeting All Days (Oct. 1997). SPE-39231-MS. DOI: 10.2118/39231-MS. eprint: https://onepetro.org/SPEERM/proceedings-pdf/97ERM/All-97ERM/SPE-39231-MS/1942900/spe-39231-ms.pdf. URL: https://doi.org/10.2118/39231-MS.

[15] David Moran et al. "Sophisticated ROP Prediction Technologies Based on Neural Network Delivers Accurate Drill Time Results." In: IADC/SPE Asia Pacific Drilling Technology Conference and Exhibition All Days (Nov. 2010). SPE-132010-MS. DOI: 10.2118/132010-MS. eprint: https://onepetro.org/SPEAPDT/proceedings-pdf/10APDT/All-10APDT/SPE-132010-MS/1719710/spe-132010-ms.pdf. URL: https://doi.org/10.2118/132010-MS.

[16] R. Jahanbakhshi, R. Keshavarzi, and A. Jafarnezhad. "Real-time Prediction of Rate of Penetration During Drilling Operation In Oil And Gas Wells." In: U.S. Rock Mechanics/Geomechanics Symposium All Days (June 2012). ARMA-2012-244. eprint: https://onepetro.org/ARMAUSRMS/proceedings-pdf/ARMA12/All-ARMA12/ARMA-2012-244/1600884/arma-2012-244.pdf.

[17] Seyed Babak Ashrafi et al. "Application of hybrid artificial neural networks for predicting rate of penetration (ROP): A case study from Marun oil field." In: *Journal of Petroleum Science and Engineering* 175 (2019), pp. 604–623. DOI: https://doi.org/10.1016/j.petrol.2018.12.013. URL: https://www.sciencedirect.com/science/article/pii/S0920410518310970.

[18] Mustafa M. Amer, Abdel Sattar DAHAB, and Abdel-Alim Hashem El-Sayed. "An ROP Predictive Model in Nile Delta Area Using Artificial Neural Networks." In: SPE Kingdom of Saudi Arabia Annual Technical Symposium and Exhibition Day 2 Tue, April 25, 2017 (Apr. 2017). D023S012R001. DOI: 10.2118/187969-MS. eprint: https://onepetro.org/SPESATS/proceedings-pdf/17SATS/2-17SATS/

D023S012R001/1294866/spe-187969-ms.pdf. URL: https://doi.org/10.2118/187969-MS.

[19] M.. Bataee and S.. Mohseni. "Application of Artificial Intelligent Systems in ROP Optimization: A Case Study in Shadegan Oil Field." In: SPE Middle East Unconventional Resources Conference and Exhibition All Days (Jan. 2011). SPE-140029-MS. DOI: 10.2118/140029-MS. eprint: https://onepetro.org/SPEUGM/proceedings-pdf/11UGM/All-11UGM/SPE-140029-MS/1689424/spe-140029-ms.pdf. URL: https://doi.org/10.2118/140029-MS.

[20] Abdolali Esmaeili et al. "ROP Modeling Using Neural Network and Drill String Vibration Data." In: SPE Kuwait International Petroleum Conference and Exhibition All Days (Dec. 2012). SPE-163330-MS. DOI: 10.2118/163330-MS. eprint: https://onepetro.org/SPEKIPC/proceedings-pdf/12KIPC/All-12KIPC/SPE-163330-MS/1661716/spe-163330-ms.pdf. URL: https://doi.org/10.2118/163330-MS.

[21] Xian Shi et al. "An Efficient Approach for Real-Time Prediction of Rate of Penetration in Offshore Drilling." In: (Sept. 2016). URL: https://www.hindawi.com/journals/mpe/2016/3575380/.

[22] Leo Breinman. "Random Forests." In: *Machine Learning(SpringerLink)* (2001).

[23] B.. Mantha and R.. Samuel. "ROP Optimization Using Artificial Intelligence Techniques with Statistical Regression Coupling." In: SPE Annual Technical Conference and Exhibition Day 3 Wed, September 28, 2016 (Sept. 2016). D031S041R007. DOI: 10.2118/181382-MS. eprint: https://onepetro.org/SPEATCE/proceedings-pdf/16ATCE/3-16ATCE/D031S041R007/1365037/spe-181382-ms.pdf. URL: https://doi.org/10.2118/181382-MS.

[24] R. Teale. "The concept of specific energy in rock drilling." In: *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts* 2.1 (1965), pp. 57–73. DOI: https://doi.org/10.1016/0148-9062(65)90022-7. URL: https://www.sciencedirect.com/science/article/pii/0148906265900227.

[25] Todd R. Hamrick. "Optimization of Operating Parameters for Minimum Mechanical Specific Energy in Drilling." PhD thesis. West Virginia University, 2011.

[26] A. Ebrahimi. "Optimizing milling-ROP by applying the concept of mechanical specific energy (MSE)." In: *Oil Gas European Magazine* 43 (Mar. 2017), pp. 35–37.

[27] Ajmed Hassan, Salaheldin Elkatatny, and Abdulaziz Al-Majed. "Coupling rate of penetration and mechanical specific energy to improve the efficiency of drilling gas wells." In: *Journal of Natural Gas Sciences and Engineering* 83 (2020).

[28] Miguel Armenta. "Identifying Inefficient Drilling conditions Using Drilling-Specific Energy." In: *Society of Petroleum Engineers* (2008).

[29] Kshitij Mohan, Faraaz Adil, and Robello Samuel. "Comprehensive Hydromechanical Specific Energy Calculation for Drilling Efficiency." In: *Journal of Energy Resources Technology* 137.1 (Sept. 2014). 012904. DOI: 10.1115/1.4028272. eprint: https:

//asmedigitalcollection.asme.org/energyresources/article-pdf/137/1/012904/6146935/jert_137_01_012904.pdf. URL: https://doi.org/10.1115/1.4028272.

[30] E.M. Galle and H.B. Woods. "Best Constant Weight and Rotary Speed for rotary Rock Bits." In: Drilling and Production Practice All Days (1963).

[31] Jr. Young F.S. "Computerized Drilling Control." In: *Journal of Petroleum Technology* 21.04 (Apr. 1969), pp. 483–496. DOI: 10.2118/2241-PA. eprint: https://onepetro.org/JPT/article-pdf/21/04/483/2222470/spe-2241-pa.pdf. URL: https://doi.org/10.2118/2241-PA.

[32] *Optimization of Drilling Performance Based on an Intelligent Drilling Advisory System*. Vol. Day 3 Thu, March 28, 2019. IPTC International Petroleum Technology Conference. D031S069R001. Mar. 2019. DOI: 10.2523/IPTC-19269-MS. eprint: https://onepetro.org/IPTCONF/proceedings-pdf/19IPTC/3-19IPTC/D031S069R001/1126942/iptc-19269-ms.pdf. URL: https://doi.org/10.2523/IPTC-19269-MS.

[33] Ulf Jakob Aarsnes, Ole Aamo, and Miroslav Krstic. "Extremum seeking for real-time optimal drilling control." In: July 2019. DOI: 10.23919/ACC.2019.8815162.

[34] Magnus Nystad, Bernt Sigve Aadnøy, and Alexey Pavlov. "Real-Time Minimization of Mechanical Specific Energy with Multivariable Extremum Seeking." In: *Energies* 14.5 (2021). DOI: 10.3390/en14051298. URL: https://www.mdpi.com/1996-1073/14/5/1298.

[35] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2020.

[36] Vincent Francois-Lavet et al. *An Introduction to Deep Reinforcement Learning*. now, 2018.

[37] Hyeong Soo Chang et al. *Simulation-Based Algorithms for Markov Decision Processes*. Springer, 2013.

[38] J.M Hammersley and D.C. Handscomb. *Monte Carlo Methods*. Chapman and Hall, 1965.

[39] Todd Hester. *TEXPLORE: Temporal Difference ReinforcementLearning for Robots and Time-Constrained Domains*. Springer, 2013.

[40] Kai Arulkumaran et al. "Deep Reinforcement Learning: A brief survey." In: *Deep Learning for Visual Understanding* (). URL: https://doi.org/10.1109/MSP.2017.2743240.

[41] Voot Tangkaratt, Masashi Sugiyama, and Abbas Abdolmaleki. "GUIDE ACTOR-CRITIC FOR CONTINUOUS CONTROL." In: ().

[42] Jang et al. Beakcheol. "Q-learning Algorithms: A Comprehensive Classification and Applications." In: ().

[43] Marc Deisenroth, Gerhard Neumann, and Jan Peters. *A Survey on Policy Search for Robotics*. Vol. 2. Aug. 2013.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[45] Shiliang Sun et al. "A Survey of Optimization Methods From a Machine Learning Perspective." In: *IEEE Transactions on Cybernetics* 50.8 (2020), pp. 3668–3681. DOI: 10.1109/TCYB.2019.2950779.

[46] Jiawei Zhang. "Gradient Descent based Optimization Algorithms for Deep Learning Models Training." In: *CoRR* abs/1903.03614 (2019). arXiv: 1903.03614. URL: http://arxiv.org/abs/1903.03614.

[47] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. URL: http://dx.doi.org/10.1038/nature14236.

[48] T. Lillicrap et al. "Continuous control with deep reinforcement learning." In: *CoRR* abs/1509.02971 (2016).

[49] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[50] Kun Shao et al. "Visual Navigation with Actor-Critic Deep Reinforcement Learning." In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489185.

[51] *OpenAI Baselines: ACKTR and A2C*. https://openai.com/blog/baselines-acktr-a2c/. Accessed: 2021-04-26.

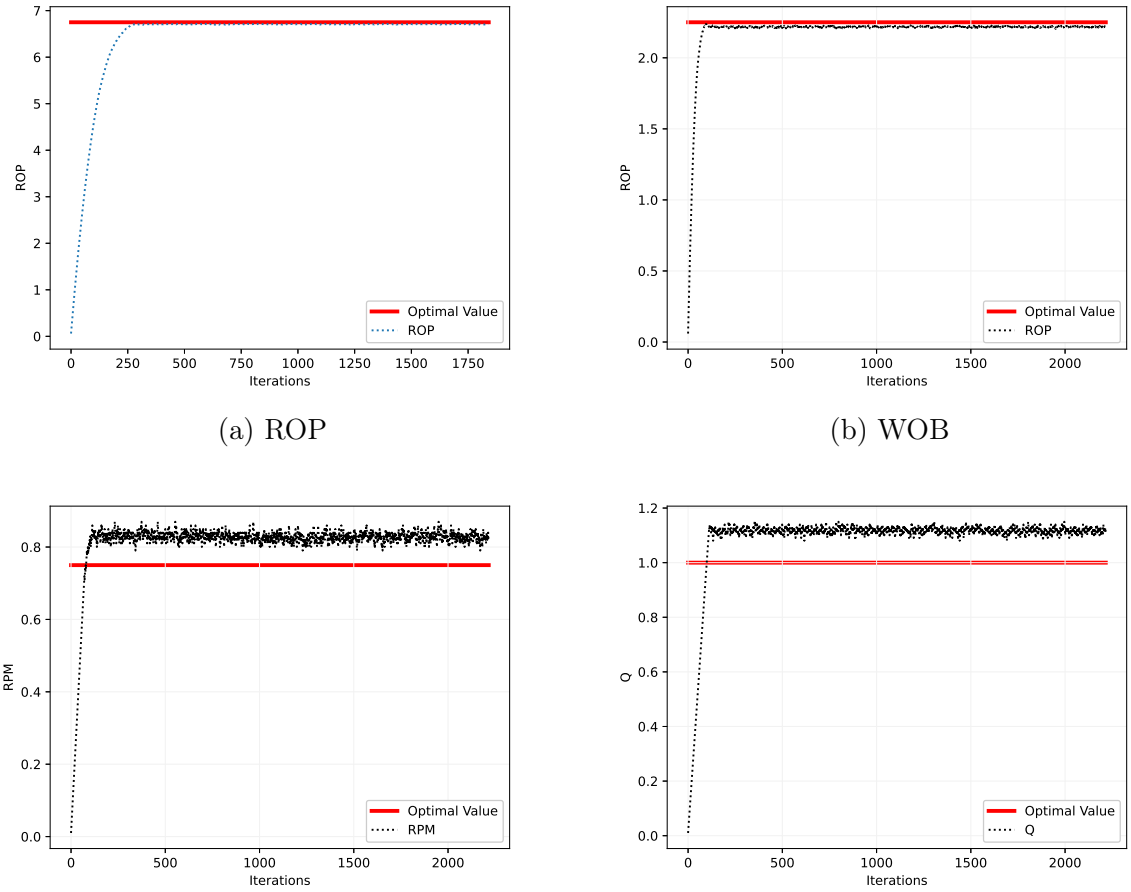
[52] Greg Brockman et al. "OpenAI Gym." In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

A | Plots

This appendix contains additional plots from experiments that might be interesting to see, but not directly necessary in the results section of the thesis.

A.1 Environment 2

Figure A.1 shows the agent trained on environment 2 interacting with a case where the optimal values are scaled individually rather than with a constant K.



(a) ROP

(b) WOB

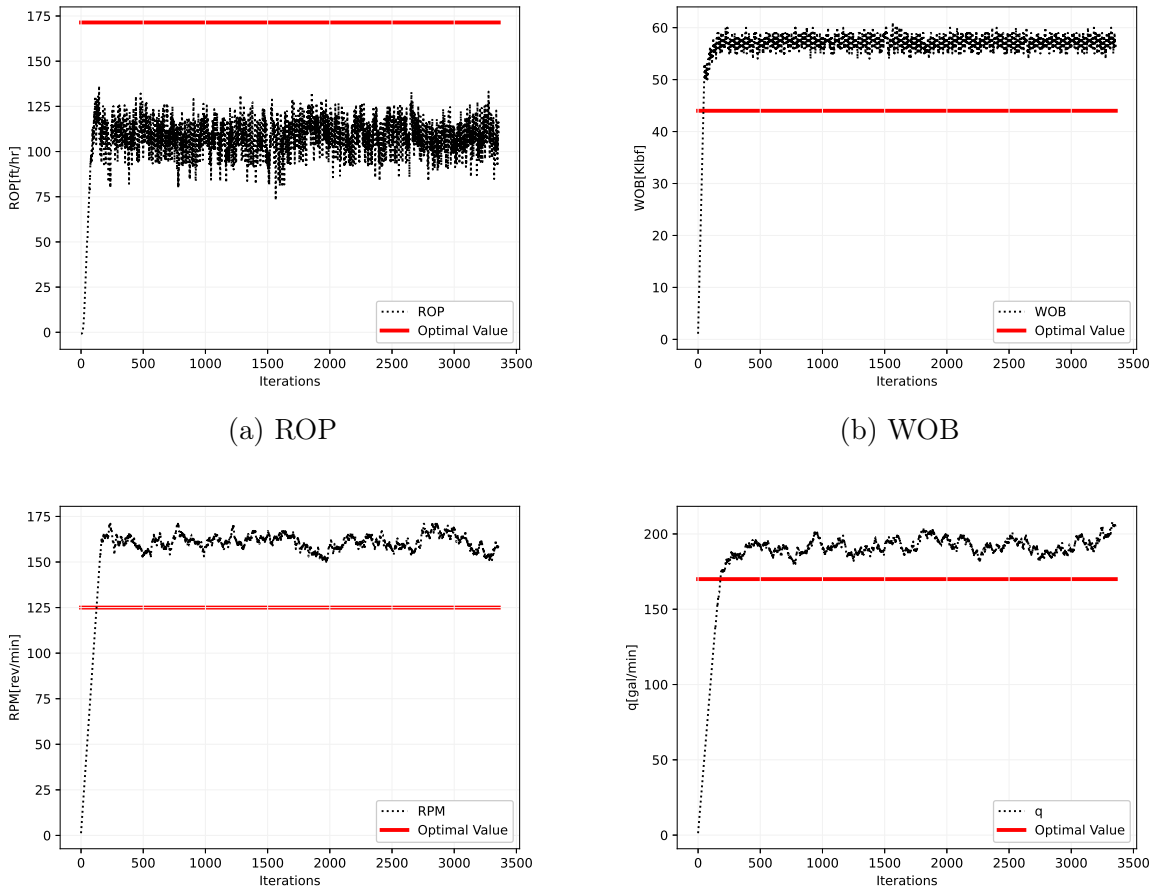
(c) rpm

(d) q

Figure A.1: Additional plot of validation of agent in environment 2.

A.2 Environment 3

Figure A.2 shows a plot where $K = 1$. The agent generalizes poorly, and the plot shows that the agent passes the optimal values, and does not find the optimal values even though ROP decreases after exceeding the optimum.



(a) ROP

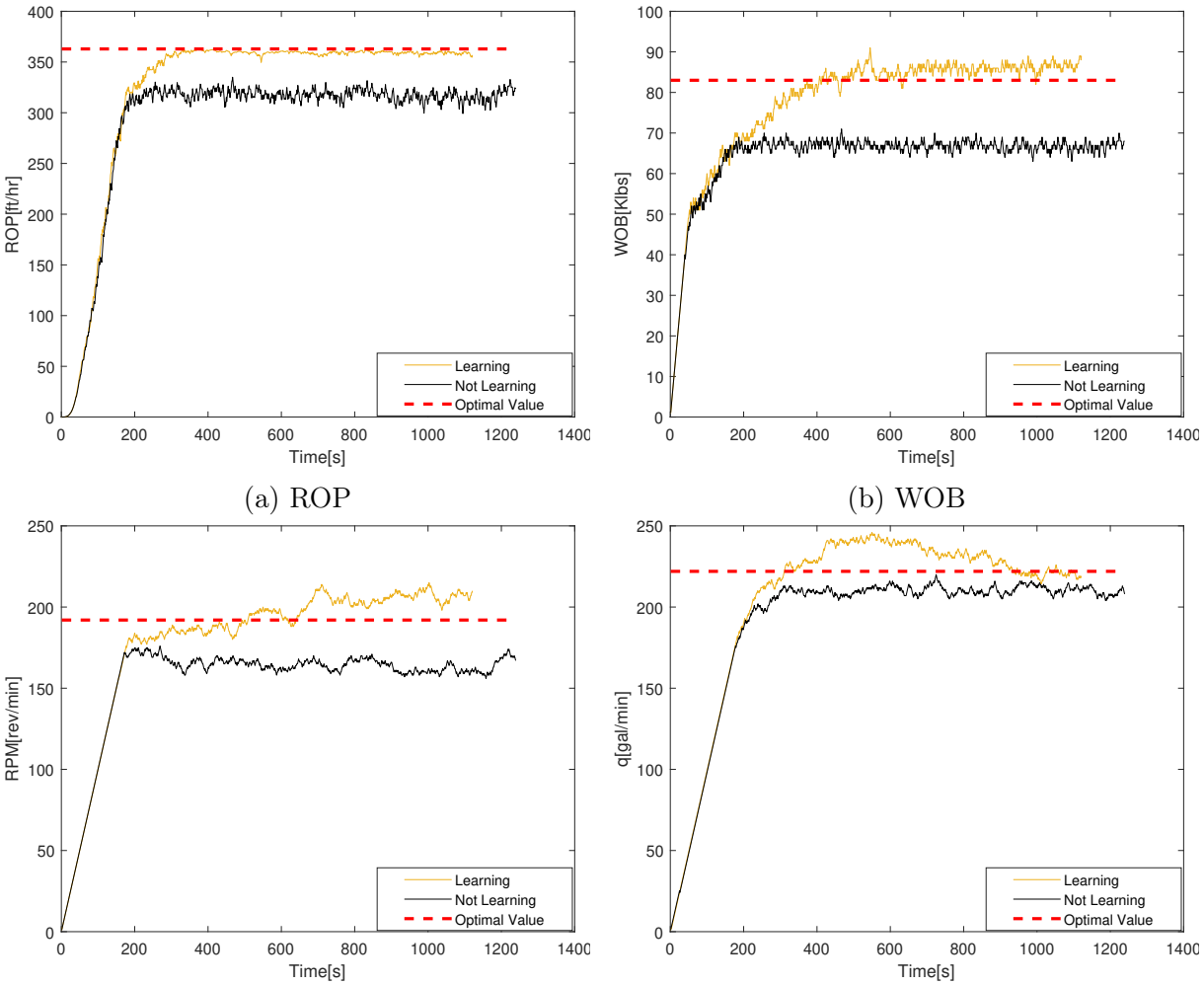
(b) WOB

(c) rpm

(d) q

Figure A.2: Additional plot from validation test of environment 3.

Figure A.3 shows a simulation where $K = 1$. The agent trajectory plotted in yellow shows an agent that continually learns, while the trajectory in black shows an agent that acts as a predictor.



(a) ROP (b) WOB

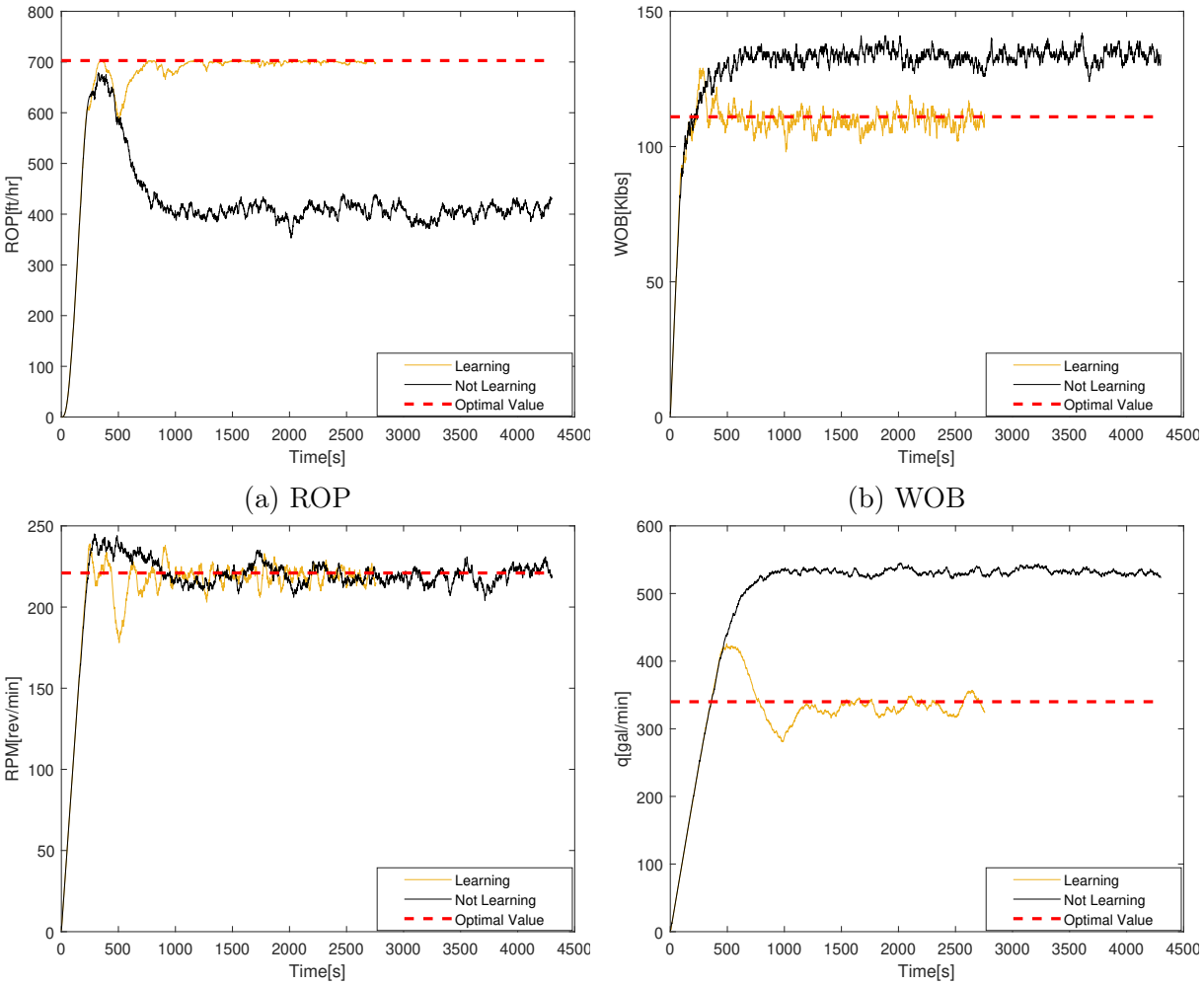
(c) rpm (d) q

Figure A.3: Additional plot of learning in environment 3.

A.3 Environment 4

This section contains plots from experimentation with the BY model. As the BY model is the most flexible ROP model used in this project, plots of different model compositions have been tested.

Figure A.4 is a simulation constructed to mimic a soft formation. The optimal input values are high, and particularly q to resemble a large volume of cuttings who needs to be cleaned of the bottomhole. The agent does not use the optimized network structure.

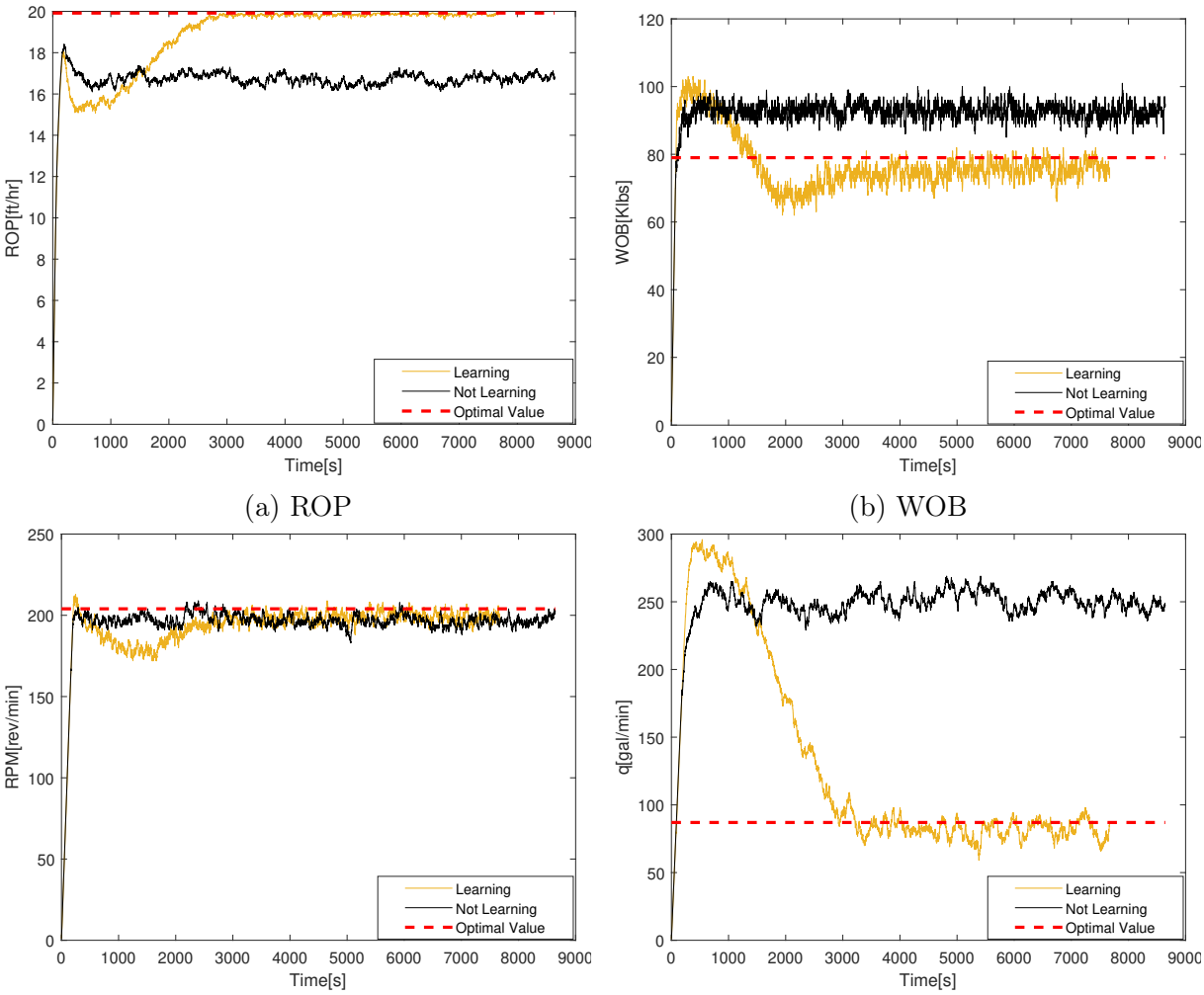


(a) ROP (b) WOB

(c) RPM (d) q

Figure A.4: Drilling in a formation resembling a soft rock type.

Figure A.5 mimics drilling in a hard formation. The optimal input, and particularly q is lower to resemble little cuttings to be cleaned. The agent does not use the optimized network structure.

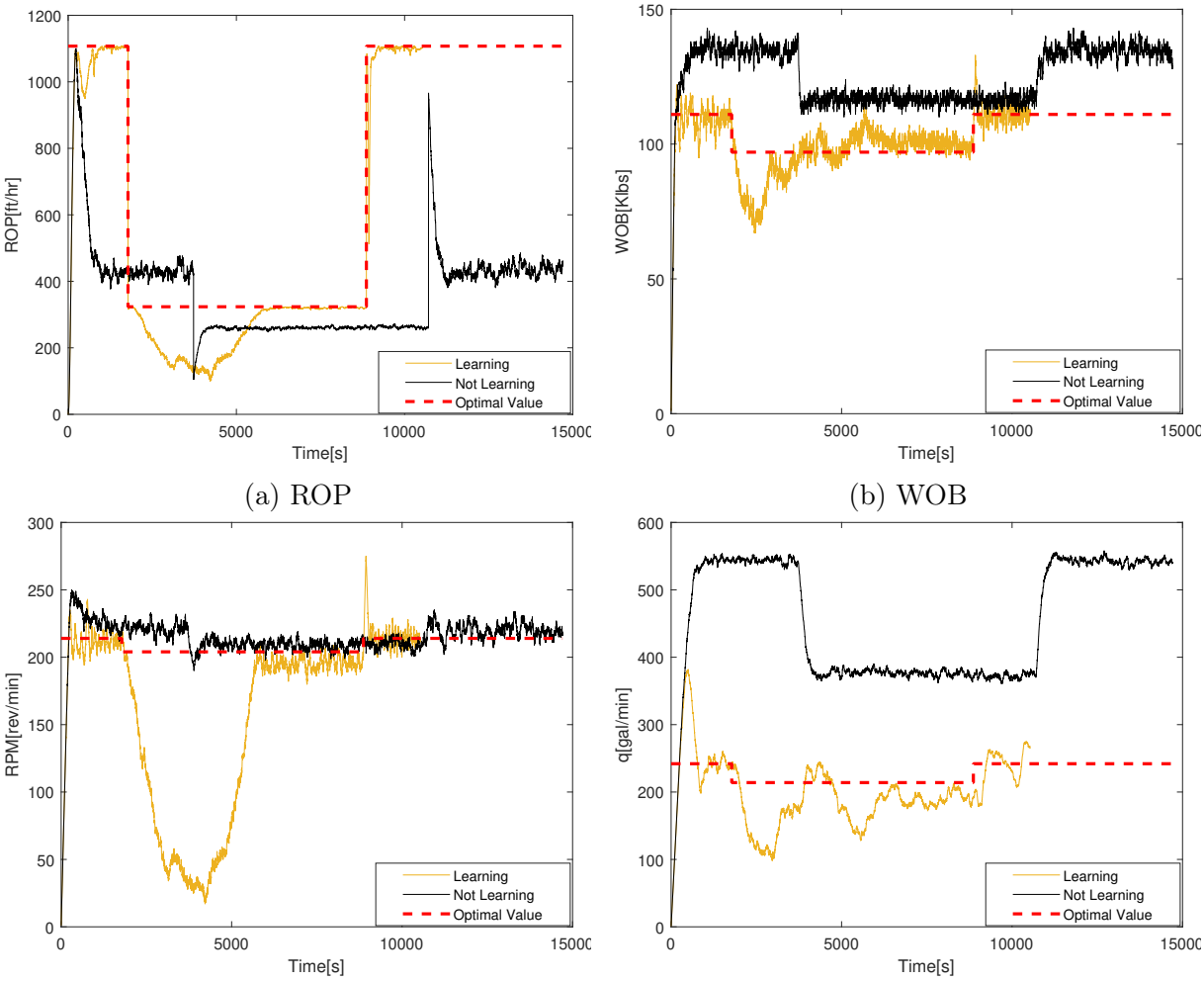


(a) ROP (b) WOB

(c) RPM (d) q

Figure A.5: Drilling in a formation resembling a hard rock type.

Figure A.6 is an extension of Figure 5.14 where the formation parameters change back to the initial value. The experiment was done to investigate if the agent "remembers" the previous formation after updating the policy on a different model configuration. The agent learns adjusts quicker to the initial formation the second time, but due to the unoptimized network architecture, the learning is inefficient.

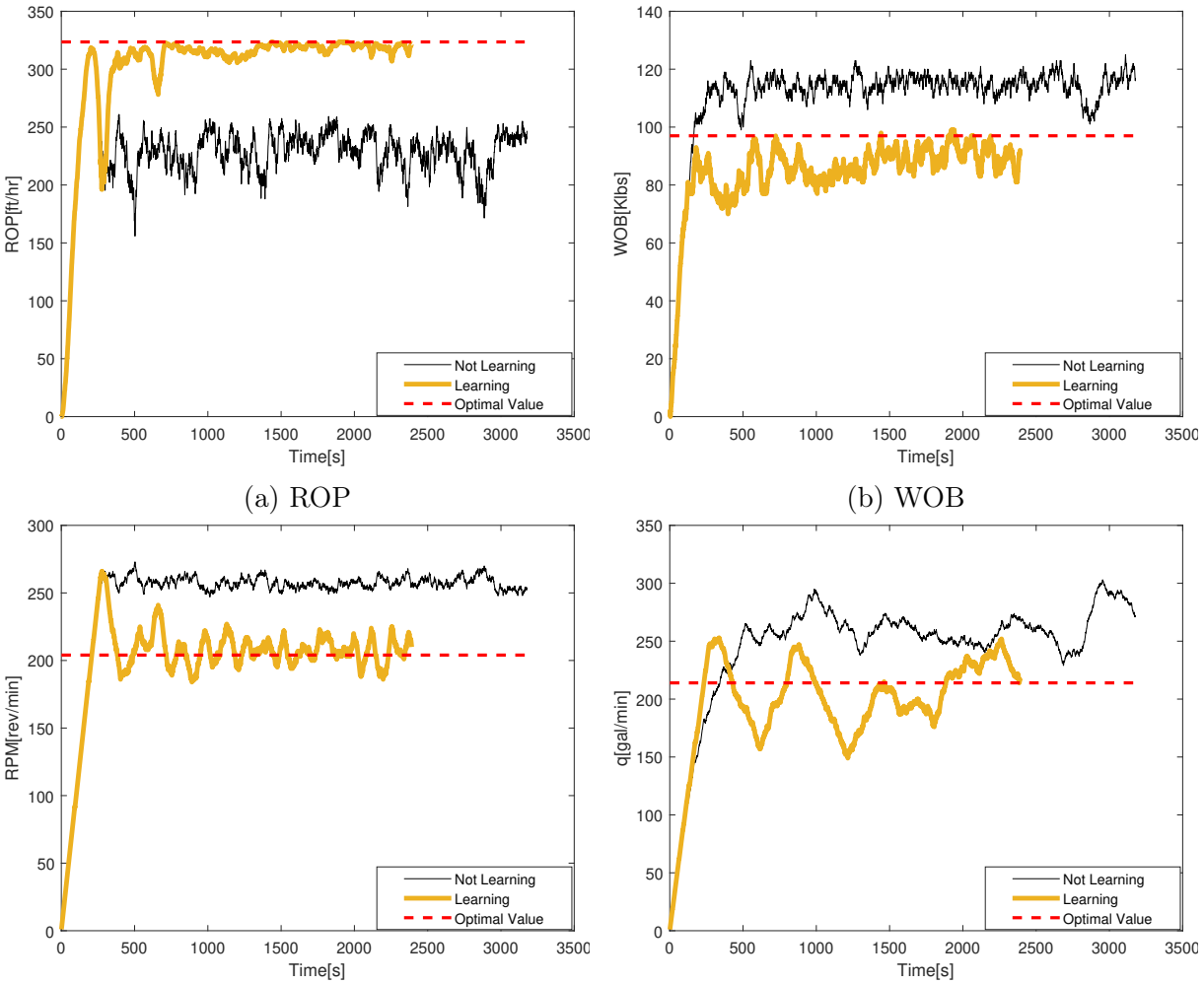


(a) ROP (b) WOB

(c) rpm (d) q

Figure A.6: Case of three drilling segments with sub-optimal network architecture.

Figure A.7 shows an experiment where the RL algorithm uses two identical network for policy and value function networks. The networks have two hidden layers with 512-512 neurons. As can be seen, the learning is unstable, and the agent over adjusts the input during the entire simulation.

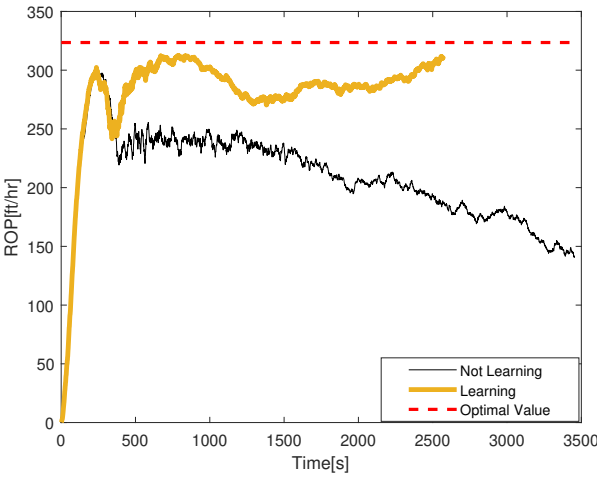


(a) ROP (b) WOB

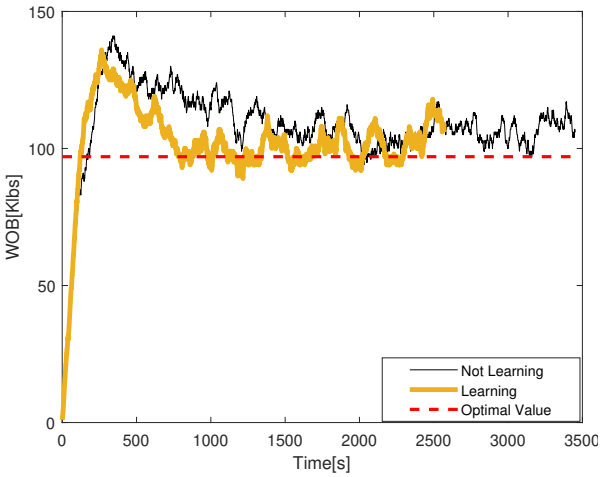
(c) RPM (d) q

Figure A.7: Network architecture of two hidden layers with 512 neurons.

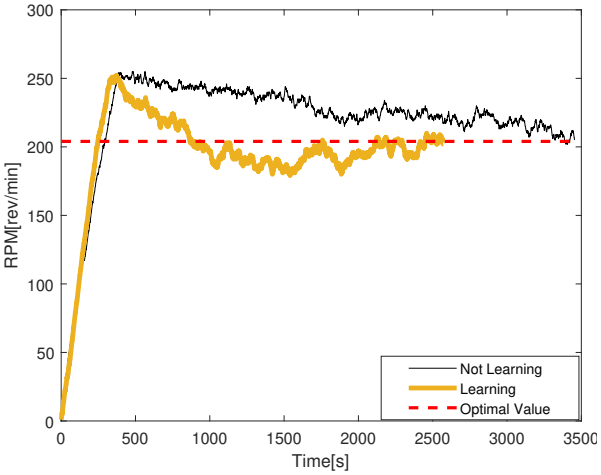
Figure A.8 shows an experiment where the RL algorithm uses two identical network for policy and value function networks. The networks have two hidden layers with 8-8 neurons. The agent is not capable of learning the optimal input values over the simulation



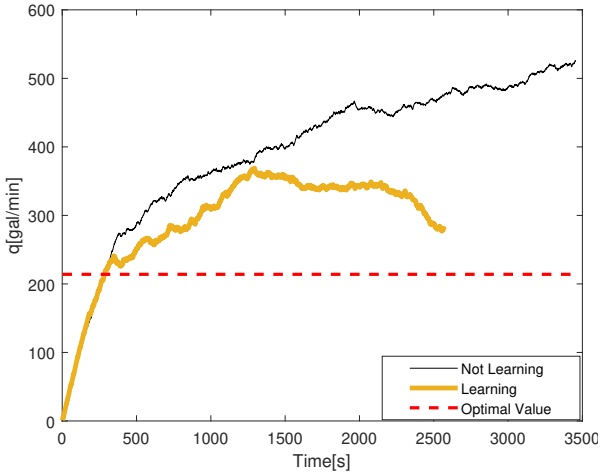
(a) ROP



(b) WOB

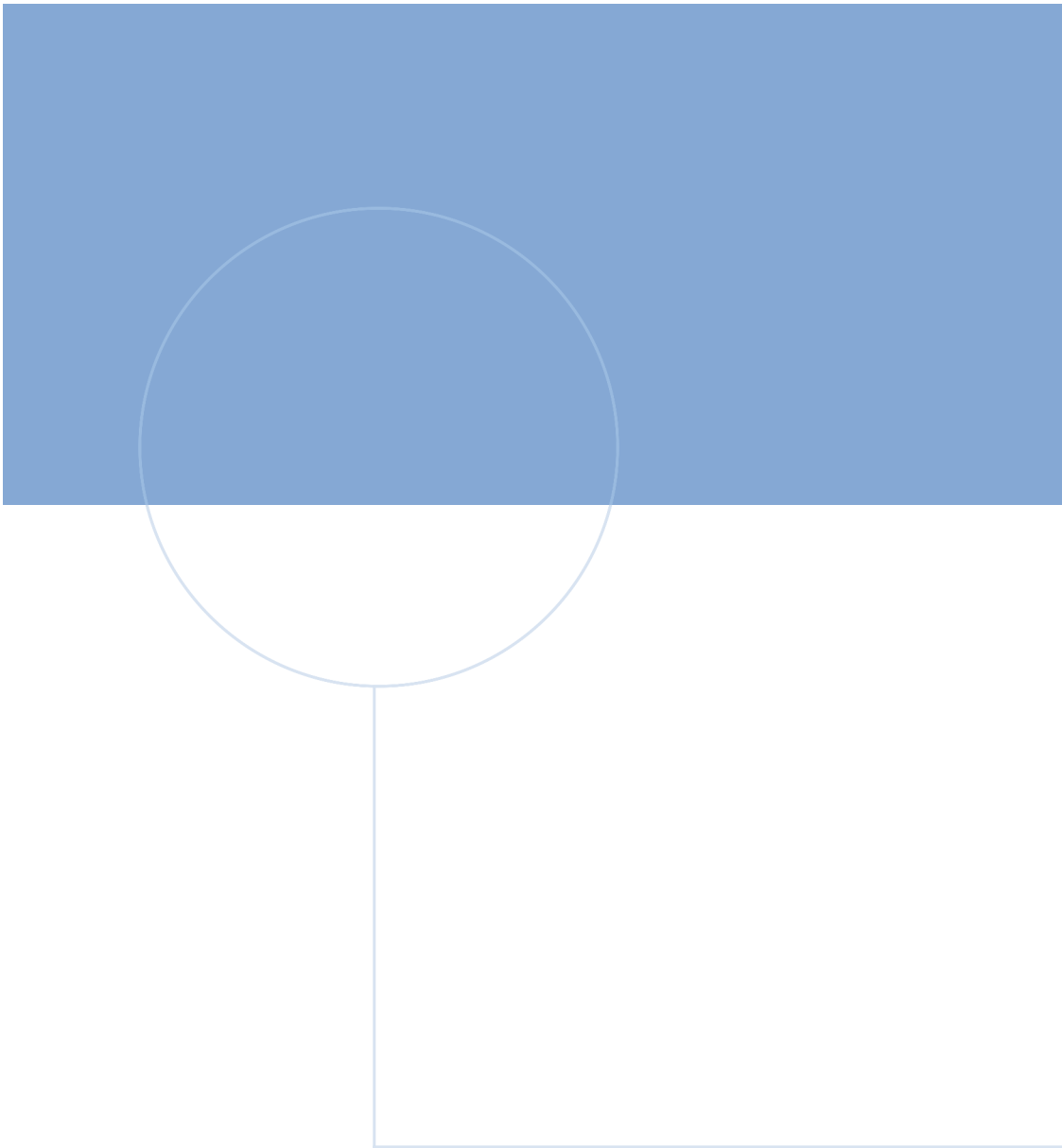


(c) rpm



(d) q

Figure A.8: Network architecture of two hidden layers with 8 neurons.



NTNU

Kunnskap for en bedre verden