USING ANOMALY DETECTION WITH MACHINE LEARNING BY ASSESSING THE BOTTOM HOLE PRESSURE CHANGES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMED MAGDY A. E. KHALIL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
PETROLEUM AND NATURAL GAS ENGINEERING

DECEMBER 2023

Approval of the thesis:

USING ANOMALY DETECTION WITH MACHINE LEARNING BY ASSESSING THE BOTTOM HOLE PRESSURE CHANGES

submitted by **AHMED MAGDY A. E. KHALIL** in partial fulfillment of the requirements for the degree of **Master of Science** i**n Petroleum and Natural Gas Engineering, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Assoc. Prof. Dr. İsmail Durgut
Head of the Department, **Petroleum and Natural Gas Engineering Dept., METU** _____

Assoc. Prof. Dr. Çağlar Sınayuç
Supervisor, **Petroleum and Natural Gas Engineering Dept., METU** _____

Examining Committee Members:

Asst. Prof. Dr. Mehmet Onur Doğan
Petroleum and Natural Gas Engineering Dept., METU _____

Assoc. Prof. Dr. Çağlar Sınayuç
Petroleum and Natural Gas Engineering Dept., METU _____

Asst. Prof. Dr. Doruk Alp
Petroleum and Natural Gas Engineering Dept., METU NCC _____

Date: 08.12.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name: Ahmed Khalil

Signature:

ABSTRACT

USING ANOMALY DETECTION WITH MACHINE LEARNING BY ASSESSING THE BOTTOM HOLE PRESSURE CHANGES

Khalil, Ahmed Magdy A. E.
Master of Science, Petroleum and Natural Gas Engineering
Supervisor: Assoc. Prof. Dr. Çağlar Sınayuç

December 2023, 77 pages

For achieving sustainability in producing from petroleum systems, it is vital to have a field model as close as possible to the real situation in order to monitor production and injection performance. In this study, it is aimed to develop a machine learning model that can be used as a representative performance indicator and coupled with an anomaly detection method. Machine learning consists of a supervised learning DNN model with four inputs and one output. Inputs are time, oil, water, and gas rates from well production. Output is the forecasted pressure performance of the well. Anomaly detection is developed by observing the predicted next day forecast and thirtieth day forecast given the previous data, after which it is compared with the actual pressures corresponding with the dates of prediction. Using this algorithm, it will help us forecast pressure or production, evaluate proposed scenarios, and identify anomaly pressures due to unaccounted reservoir characteristics between wells (like faults, permeability, or porosity changes). Additionally, the given model algorithm will be able to be tested on many different wells in a relatively short time, giving headroom to build an efficient development plan. Proposed scenarios can consist of new infill wells, changes in operational conditions, or changes in injection/production patterns. The thesis developed and tested an algorithm to forecast and find anomaly. It showed accurate and robust results that match with the results of a commercial simulator in spite of the complexity of the reservoir and the fitting procedure of the machine learning model.

Keywords: Petroleum systems, Pressure Performance, Machine Learning, Anomaly Detection, Proxy Model.

ÖZ

MAKİNE ÖĞRENME İLE ANOMALİ TESPİTİ UYGULAYARAK KUYU DİBİ BASINÇ DEĞİŞİKLİKLERİNİ DEĞERLENDIRMEK

Khalil, Ahmed Magdy A. E.
Yüksek Lisans, Petrol ve Doğal gaz Mühendisliği
Tez Yöneticisi: Doç. Dr. Çağlar Sınayuç

Aralık 2023, 77 sayfa

Petrol sistemlerinden üretimde sürdürülebilirliğin sağlanması için üretim ve enjeksiyon performansının izlenebilmesi amacıyla gerçeğe mümkün olduğunca yakın bir saha modelinin olması hayati önem taşımaktadır. Bu çalışmada, temsili bir performans göstergesi olarak kullanılabilecek ve bir anormallik tespit yöntemini kullanan bir makine öğrenme modelinin geliştirilmesi amaçlanmaktadır. Makine öğrenimi, dört girdi parametresi ve bir çıktı parametresi olan denetimli öğrenme DNN (derin sinir ağı) modelinden oluşur. Kuyu üretiminden elde edilen girdiler zaman, petrol, su ve gaz oranlarıdır. Çıktı kuyunun tahmin edilen basınç performansıdır. Anormallik tespiti, eldeki verilere göre tahmin edilen ertesi gün ve otuzuncu gün tahmininin gözlemlenmesi ve ardından tahmin tarihlerine karşılık gelen gerçek basınç değerleri ile karşılaştırılmasıyla geliştirilir. Bu algoritmayı kullanarak, basınç veya üretimi tahmin etmemize, önerilen senaryoları değerlendirmemize ve kuyular arasındaki hesaba katılmayan rezervuar özelliklerinden (faylar, geçirgenlik veya gözeneklilik değişiklikleri gibi) kaynaklanan anormal basınç değerlerini belirlememize yardımcı olacaktır. Ek olarak, verilen model algoritması nispeten kısa bir sürede birçok farklı kuyuda test edilebilmekte ve verimli bir geliştirme planı oluşturmak için imkan tanımaktadır. Önerilen senaryolar yeni enjeksiyon kuyularından, işletme koşullarındaki değişikliklerden veya enjeksiyon/üretim modellerindeki değişikliklerden oluşabilir.

Bu tezde, anormalliği tahmin etmek ve bulmak için bir algoritma geliştirilmiş ve test edilmiştir. Rezervuarın karmaşıklığına ve makine öğrenimi modelinin eşleşme prosedürüne bağlı olarak ticari simulator sonuçları ile karşılaştırıldığında doğru ve sağlam sonuçlar göstermiştir.

Anahtar Kelimeler: Petrol sistemleri, Basınç Performansı, Makine Öğrenmesi, Anomali Tespiti, Proxy Modeli

To my Grandmother

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Assoc. Prof. Dr. Çağlar Sınayuç for their guidance, advice, criticism, encouragements, and insight throughout the research.

I present great thanks to Asst. Prof. Dr. Mehmet Onur Doğan, Asst. Prof. Dr. Doruk Alp, Asst. Prof. Dr. Emre Artun for their valuable insights. As well as Prof. Dr. Serhat Akın for using his geophysical model.

I want to express my deep gratitude to all my instructors at METU Ankara and METU NCC, particularly those in the Department of Petroleum & Natural Gas Engineering. Their invaluable contributions, their patience with my shortcoming, and encouragement that have been instrumental in shaping my academic journey and professional development. We did not only take knowledge, but also absorbed morals and manners from you. As you plant the seed of knowledge into us, shall it grow and mature to be harvested and benefit the whole world.

I extend my heartfelt thanks to all my colleagues from METU, specifically those in the Department of Petroleum & Natural Gas Engineering, for their unwavering support throughout my undergraduate and graduate studies.

Finally, I also acknowledge METU Graduate School of Natural and Applied Sciences for allowing me the opportunity to obtain my M.S. degree.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
DNN	Deep Neural Network
SGD	Stochastic Gradient Descent
Adam	Adaptive moment estimation
LR	Learning Rate
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
bbl/d	Barrel per day
ft	feet
m	meter
BHP	Bottom Hole Pressure
CMG	Computer Modelling Group Ltd.
psi	Pounds per square inch
M unit	A thousand of the unit
MM unit	A million of the unit
MAE	Mean Absolute Error
MSE	Mean Square Error
RMSE	Root Mean Square Error

CHAPTER 1

INTRODUCTION

Time is a valuable resource. As time moves in one direction, we are presented with many decisions to take. Some may be small and intuitive, others are life changing, but the only truth is that they cannot be taken back. That is why all resources are used to make the right decision. Engineering simulation programs are used for example, to simulate actual data where it gives us an opportunity to test decisions before applying it in real life. However, it comes at the cost of more time. Here a proxy model is presented where an algorithm is designed to reduce the time used on simulation and increase the decision window.

AI and ML have recently been the focus to be integrated in every industry to cooperate with big data analytics as more data is available frequently thanks to technological advancements in equipment's, apparatuses, and automation. Oil and gas industry is no different. As it has even more data, it is integrated into the oil and gas industry as well. From well placement (Akin et al., 2010) to economic decisions and reservoir management (Ertekin & Sun, 2019) as seen more and more lots of sectors are joining and adopting AI in their work. Learning and applying AI becomes a useful tool for engineers cooperating with conventional methods to assist them designing and developing more sophisticated solutions.

Multiple proxy model studies have been presented in literature that predict future performance. Forecasting the Water-Cut in carbonate reservoir, oil recovery of a single cyclic nitrogen injection, or cumulative gas recovery from shale wells are all successfully implemented machine learning models (Artun, 2022). Additionally, a heat-map generated for optimizing well placement (Akin et al., 2010), and flow rates optimizations in a geothermal field (ARITURK, 2019). A second issue is identifying

anomalies early before it becomes a major problem. This is an issue of safety and economics that solving it means a safe and smooth non-interrupted operational environment that increases overall economical sustainability for longer. In petroleum engineering, anomaly detection has been used to identify irregularities in gas flow rate (Alharbi et al., 2022) and unplanned downtime (Feder, 2020).

Presenting here a hypothetical model with data obtained from a numerical simulation model and given to an algorithm. The algorithm includes a machine learning proxy model trains and forecasts, and an Anomaly detection that works coherently with the proxy model and tries to alarm you as soon as an anomaly is risen to take the right decisions. It can also be scaled to any number of wells. With the detection method for sudden and slight changes in pressure data.

Here a hypothetical model is introduced and tested within the bounded complexity of the geological model created for it. The introduced scenarios are meant to simulate different development plans that can be applied and extended in the field. The ML model is developed in a progressive way to show the stages of implementing a proxy model. Finally, anomaly detection scans through the forecasted and actual pressure data to find any potential anomalies not introduced during training.

CHAPTER 2

LITERATURE REVIEW

2.1 AI and ML

Artificial intelligence (AI) is a field of computer science that focuses on creating intelligent machines that can mimic human behavior and thought processes such as learning, reasoning, adapting, and self-correction (Noble & Noble, 2023). It involves the development of algorithms and software that enable computers to process large amounts of data, learn from patterns or features in the data, and make decisions or predictions based on that information.

AI encompasses a wide range of techniques and technologies, including machine learning, neural networks, expert systems, and natural language processing (Covarrubias-Moreno, 2022). The development of AI has had a significant impact on various sectors, including government, healthcare, transportation, and security (Ashri, 2020). In medicine, AI has been used to improve diagnosis, prognosis, and treatment, leading to greater accuracy and reliability in healthcare (Freitas, 2018). AI has also found applications in forensic science, aiding in crime scene investigation, DNA analysis, pattern recognition, and image processing (Ahmed Alaa El-Din, 2022). While AI has its limitations, it serves as a supplementary tool to human specialists and has the potential to contribute significantly to various fields.

Machine learning is a branch of artificial intelligence that uses data and algorithms to enable machines to learn from experience and perform tasks that would normally require human intelligence. For example, machine learning can help machines recognize faces, translate languages, recommend products, and drive cars (Yadav et al., 2022).

There are different types of machine learning, depending on how the machines learn from the data (IBM, 2023). Some of the most common types are:

- Supervised learning: The machine learns from labeled data, which means the data has predefined categories or outcomes. The machine uses the data to learn a function that maps the input to the output. The goal is to make accurate predictions or classifications for new data. For example, supervised learning can be used to classify spam emails, predict house prices, or recognize handwritten digits.
- Unsupervised learning: The machine learns from unlabeled data, which means the data has no predefined categories or outcomes. The machine uses the data to discover patterns, structures, or features that are not obvious to humans. The goal is to find hidden insights or groupings in the data. For example, unsupervised learning can be used to cluster customers, detect anomalies, or compress images.
- Reinforcement learning: The machine learns from its own actions and feedback from the environment, which means the data is generated by the machine's interaction with the environment. The machine uses the data to learn a policy that maximizes a reward or minimizes a cost. The goal is to find the optimal behavior for a given situation. For example, reinforcement learning can be used to play games, control robots, or optimize traffic (Coursera, 2023).

Machine learning has many applications in various domains, such as business, healthcare, education, and entertainment. Some of the benefits of machine learning are (Coursera, 2023):

- It can automate tasks that are tedious or complex for humans.
- It can improve the quality, efficiency, and accuracy of products and services.
- It can enhance the creativity, innovation, and personalization of human endeavors.

- It can generate new knowledge and insights from large and diverse data sources.

However, machine learning also poses some challenges and risks, such as (Coursera, 2023):

- It can be biased, unfair, or unethical if the data or algorithms are not properly designed, tested, or regulated.
- It can be vulnerable, unreliable, or harmful if the data or algorithms are corrupted, manipulated, or hacked.
- It can be complex, opaque, or unpredictable if the data or algorithms are not well understood, explained, or verified.
- It can be disruptive, competitive, or threatening if the data or algorithms are not aligned with human values, goals, or interests.

2.1.1 AI applications in petroleum engineering

AI algorithms, including machine learning (ML), have been used to integrate data-driven modeling and ML algorithms in different petroleum engineering challenges, such as exploration and development, reservoir engineering, and well logging. The use of AI in petroleum engineering aims to enhance efficiency, optimize production, and provide valuable insights for decision-making in the industry. Some examples where AI has been applied in petroleum engineering are,

1- Gas injection techniques, CO2 injection enhanced oil recovery (EOR) procedures (Hadavimoghaddam et al., 2023). “Despite their black-box nature, ANN models can understand the non-linear patterns that underpin large datasets, making them suitable for a wide range of subsurface issues” Hadavimoghaddam explains showing proxy model prediction accuracy for oil production rate that stayed within 5% error (Figure 2.1).

2- Reservoir management (Lobut & Artun, 2023). In their paper titled "Machine-Learning Based Selection of Candidate Wells for Extended Shut-In Due to Fluctuating Oil Prices" with the use of an unsupervised machine learning they categorized which well groups should shut-in given the information data from an old (50+ years) production and over 150 wells.
3- Forecasting reservoir performance. Artun (2022) after testing through different hyperparameters such as number of layers and neurons, he landed on a neural network with 2 layers and neurons of 40-25. Mostly seen a higher R^2 accuracy with more layers and neurons (Table 2.1). The chosen neural network had the most accurate for both mainly focused variables (Net Present Value and Incremental Recovery) (Figure 2.2).

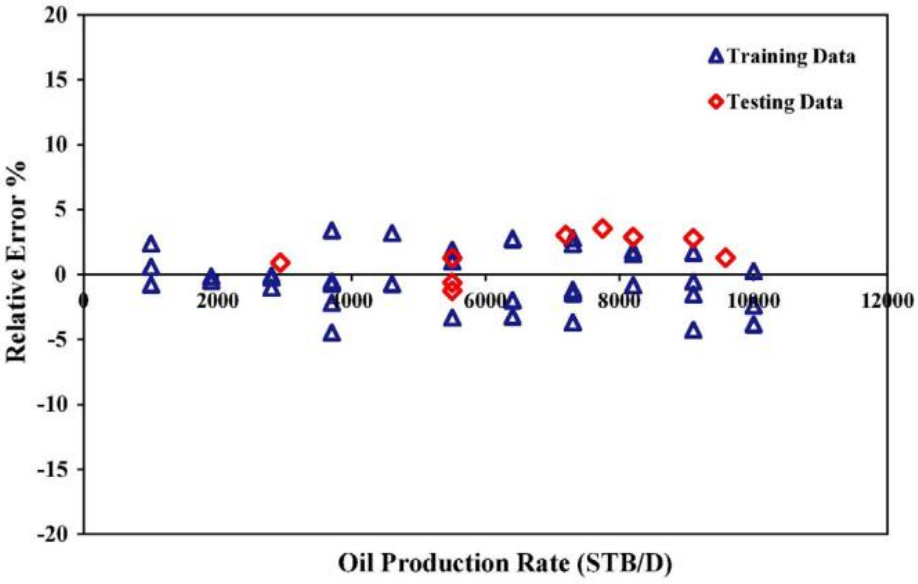

Figure 2.1. Investigation of the proxy model and (CMG) results based (Ahmadi et al., 2018).

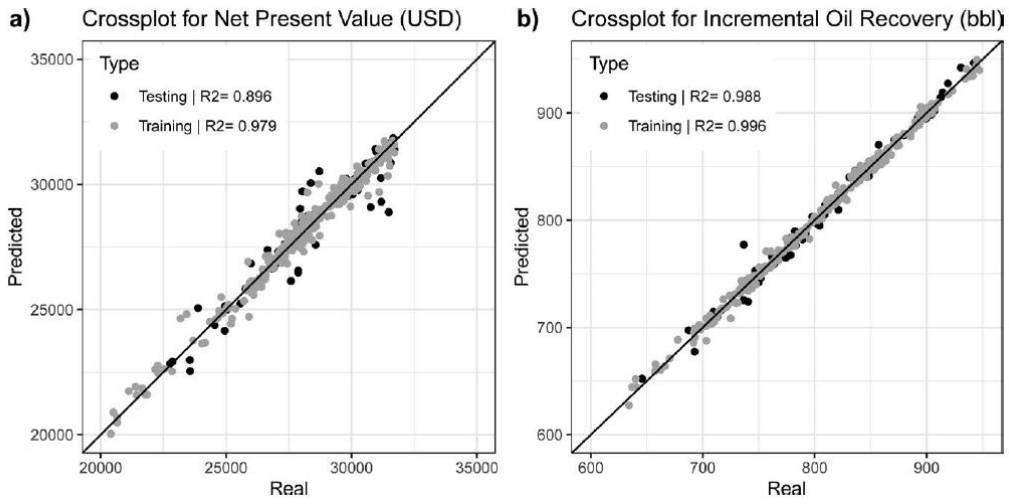

Figure 2.2. Real vs predicted cross-plots of (a) net present value (NPV) and (b) incremental oil recovery for the cyclic nitrogen injection problem (Artun, 2022).

Table 2.1.Training and Testing Performances of Different Neural Network (Artun, 2022).

		Training				Testing			
		Net Present Value, (US$)		Incremental Recovery, bbl		Net Present Value, (US$)		Incremental Recovery, bbl	
No. of layers/neurons		R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE
1 layer	10	0.93	11,287	0.99	137.2	0.84	7,913	0.92	169.1
	20	0.95	9,364	0.99	114.2	0.89	6,530	0.95	137.7
	30	0.97	7,577	0.99	104.4	0.90	6,101	0.96	115.1
	40	0.97	7,613	1.00	82.5	0.85	7,579	0.94	149.8
2 layers	10–5	0.96	8,936	0.99	92.5	0.83	8,072	0.95	138.8
	20–10	0.90	5,087	0.99	92.4	0.68	11,071	0.99	69.1
	30–20	0.98	6,011	1.00	74.4	0.81	8,535	0.98	79.4
	40–25[a]	**0.98**	**6,192**	**1.00**	**74.5**	**0.90**	**6,301**	**0.99**	**66.5**
3 layers	15–10-5	0.97	7,260	0.99	100.8	0.87	7,013	0.96	117.8
	30–20–10	0.97	6,903	0.99	94.1	0.88	6,852	0.97	109.8
	40–30–20	0.98	5,899	1.00	72.9	0.83	8,081	0.98	93.3
	50–35–20	0.98	5,263	1.00	67.4	0.88	6,860	0.98	85.3

[a] Selected model.

2.1.1.1 Well placement.

Akin et al. (2010), in their paper "Optimization of well placement geothermal reservoirs using artificial intelligence", introduces a framework that employs artificial neural networks and an optimization algorithm to determine the optimal injection well location in complex carbonate geothermal reservoirs. The results indicate that this approach effectively narrows down potential regions for optimum well placement, offering a feasible alternative to exhaustive searches. Additionally, the study underscores the importance of considering design parameters and injection flow rates when optimizing well locations within such reservoirs.

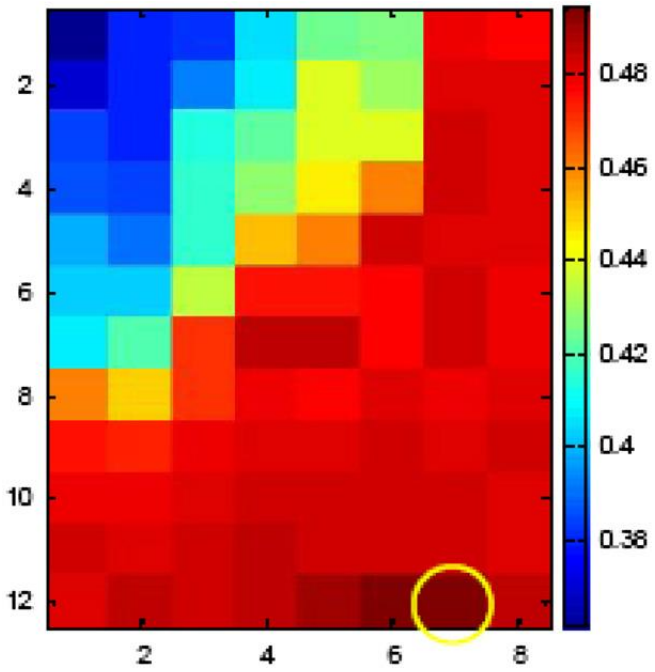

Figure 2.3. Evaluation surface for search-2. Optimum location is found to be (x-grid 7) (y-grid 12) (Akin et al., 2010)

2.1.1.2 Optimizing Flow Rates.

Arıtürk (2019), in his thesis "Optimizing the Production and Injection Wells Flow Rates in Geothermal Field Using Artificial Intelligence", demonstrates that employing Artificial Intelligence (AI) and machine learning (ML) techniques can

effectively predict future production and injection flow rates in geothermal fields. By utilizing reliable field data, AI-based models outperform conventional methods, addressing challenges such as gas presence, uncertain reservoir boundaries, and non-isothermal fluid flow (ARITURK, 2019). These findings offer a data-driven approach for optimizing power plant efficiency and continuous energy generation in geothermal systems. An example of flow rates prediction in KD-3 well and data set splitting for training and testing (Figure 2.4).

Figure 2.4. KD-3 Well Future Flow Rates Prediction (Ariturk, 2019).

2.2 Anomaly Detection

Anomaly detection is the process of identifying unexpected patterns in data. It is a topic of growing interest due to its applicability in various fields such as intrusion detection, fraud detection, fault detection, and system health monitoring. These anomalies can be either positive or negative, but in all cases, their detection is crucial for decision-making processes. This is particularly true in the petroleum industry, where heavy extraction machinery like turbomachines is monitored by numerous sensors to prevent damage (Martí et al., 2015). The authors introduce a new approach for efficient anomaly detection in turbomachines, combining the Yet Another Segmentation Algorithm (YASA) with a one-class support vector machine. This method addresses the challenge of limited labeled training data. The effectiveness of

this approach is demonstrated through comparative studies with other methods on benchmark problems and a real-life application related to oil platform turbomachinery anomaly detection.

2.2.1 Anomaly Detection in petroleum engineering

Alharbi et al. (2022), in their paper “Explainable and Interpretable Anomaly Detection Models for Production Data” paper concludes that establishing trust in machine-learning models is essential for driving the fourth industrial revolution (Alharbi et al., 2022). While white-box models offer transparency and understandability (e.g., as if-then rules), explaining the decisions of black-box models remains challenging. Through a comprehensive analysis of various models on production data sets, the study emphasizes the significance of identifying anomalies to ensure operational safety and well performance. Metrics such as F1 score, and complexity were used to compare model performance. The results reveal variations in performance across different models and data sets. The study also highlights the importance of local and global analysis for understanding model decisions. The findings underscore that model selection should consider both prediction performance and interpretability, aligning with the assertion that the highest holdout accuracy doesn't necessarily indicate trustworthiness.

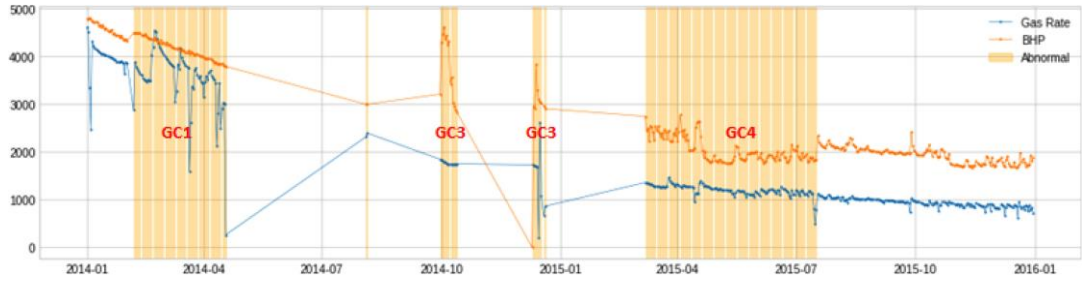

Figure 2.5. Visualization of normal and abnormal records in the gas data set (Alharbi et al., 2022).

In the paper it shows how white-box models sometimes outperform black-box models in anomaly detection tasks. Therefore, in this study, a white-box algorithm for anomaly detection is built from scratch to fit and work with the ML model. Anomaly detected regions colored is also inspired to be shown in the results plot.

2.2.2 Deepwater Facility

The deepwater facility has multiple sensor data for many aspects of their system, including equipment measurements and health, trip sittings (Feder, 2020). Often those sensors can falsely trigger the alarm system that causes unplanned shutdown. Unplanned downtime is contributed by automation hardware failure, equipment failure, process trips, and production ramp-up. According to the alarm database, there were several incidents of unexpected shutdowns around these critical components that caused negative consequences such as delayed production, total facility closure, reduced sales volume, and higher operational costs. The ML solution is then introduced to:

- Ingest numerous sensor data.
- Generate a single alarm indicating the health of a particular system or piece of equipment.
- Predict abnormal events that could lead to a shutdown.
- Potentially provide insight to prevent upcoming shutdown through root-cause analysis.

The anomaly detection based on autoencoder, and principal component analysis algorithms were found to outperform other algorithms given the type of data. Using statistical analysis on historical alarm data, several subsystems were identified for the purpose of achieving reliable and robust predictions. This process incorporated both process knowledge and critical equipment sensor data into machine learning models for anomaly detection. These models, trained on historical records, were designed to monitor patterns in sensor data in a multivariate setting and represent

system health through a single indicator known as an anomaly score. This real-time prediction of anomalous behavior is crucial. When the goal is to reduce operational and surveillance costs, it's essential to proactively detect and diagnose unplanned shutdowns. In this context, optimizing the maintenance of critical equipment can significantly contribute to this effort.

This ML framework (Figure 2.6) is somewhat like this study algorithm where after processing the real-time data the algorithm is then automatically choose whether to be trained or deployed for prediction. The choice is based on the amount of error found from the last observation.

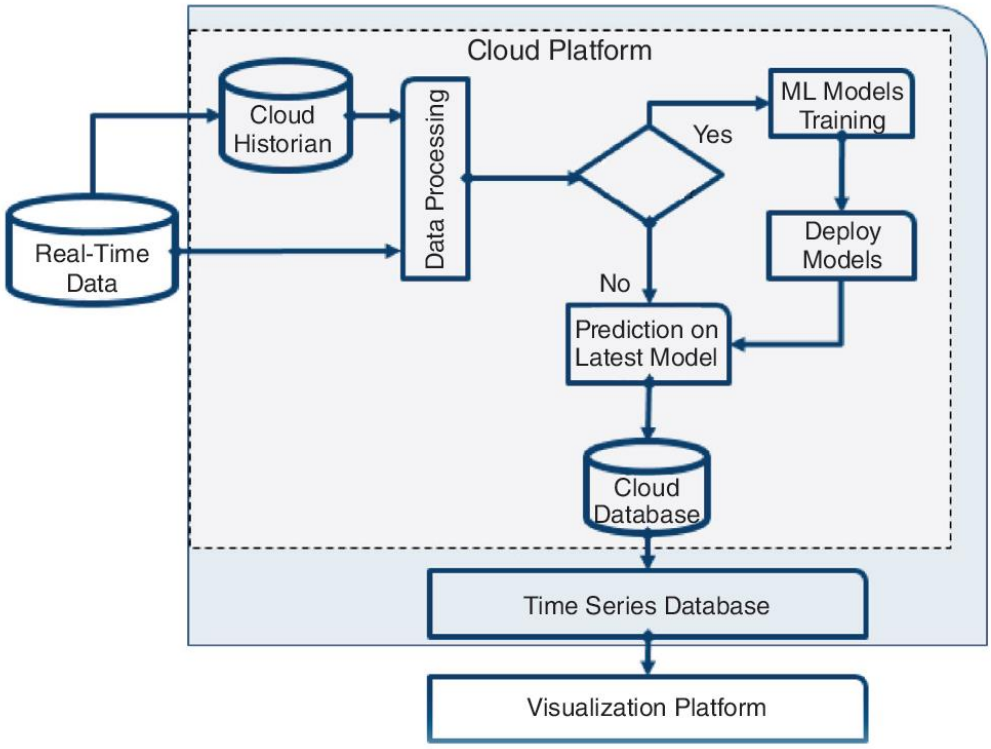

Figure 2.6. Model deployment solution using cloud computing platform (Feder, 2020).

2.3 Conventional approaches

Some conventional methods for predicting the production of hydrocarbon resources, such as decline-curve analysis and material balance, are not reliable in some situations because of the complex features of these systems. Researchers have tried to improve these methods, but there are still many challenges to overcome. Factors such as adsorption/desorption, turbulent flow in tiny pores, flow dynamics in fractures, and geomechanical effects due to fracturing make it difficult to build accurate numerical models of the reservoirs (Al-Alwani et al. 2019). However, recent advances in data collection, processing, and data-driven model building have encouraged new attempts to enhance the modeling of these problems. Therefore, there has been a growing number of publications that demonstrate the use of machine learning algorithms to develop forecasting models for hydrocarbon resources(Artun, 2022).

AI models demonstrate their benefits in terms of rapid computational efficiency and robust adaptability. Nonetheless, it's important to note that intelligent systems cannot entirely supplant traditional reservoir engineering approaches, including high-fidelity numerical simulation models and analytical tools.

While AI models significantly outpace high-fidelity numerical models in terms of computational speed, it's crucial to recognize that intelligent systems require pre-training before they can be employed to address a reservoir engineering challenge. Even for well-established expert systems, there are always inherent error margins that must be considered (Ertekin & Sun, 2019).

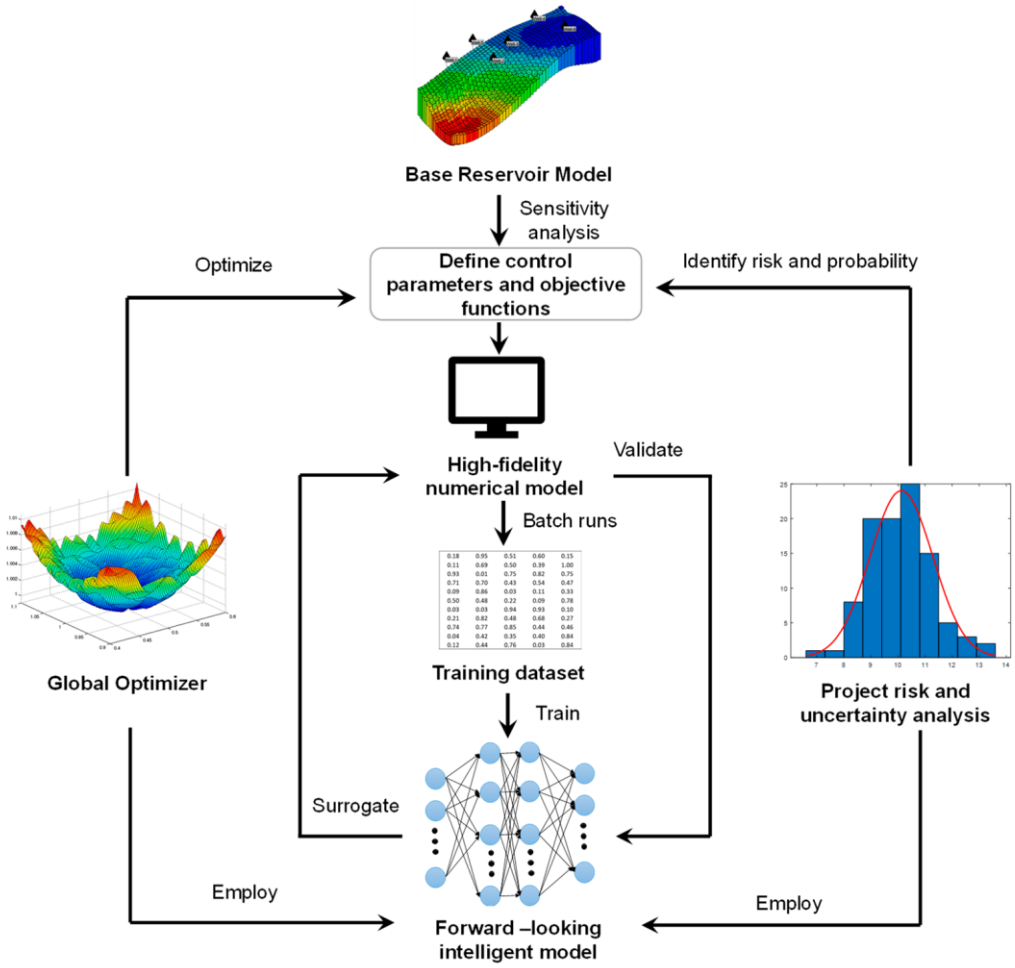

Figure 2.7. Structures of forward and inverse-looking AI model (Ertekin & Sun, 2019)

CHAPTER 3

STATEMENT OF THE PROBLEM

"Information is the oil of the 21st century, and analytics is the combustion engine." A quote by Peter Sondergaard can describe the data-driven world we live in. As technology advances, we see more tools and equipment that facilitate data extraction. Still, they are imprecise noisy-data, due to human and machine errors, that can lead to ambiguous interpretation. The main objective is to develop a machine learning model that can assess reading the pressure data of a petroleum field. It can detect any anomaly introduced in pressure readings by forecasting the near future pressure data then comparing it with the corresponding actual pressure data that measured from wells. This model will help notify engineers about any sudden and, to a certain extent, the subtle change in pressure from any of the existing wells.

CHAPTER 4

Hypothetical model

The main idea here is to design a well-fit proxy model to predict the bottom hole pressure of a well and is sensitive enough to catch anomalies through pressure readings. The proxy model built is a Deep Neural Network (DNN) that takes input data from field production (time, oil rate, water rate, gas rate) and its output forecast the production pressure for the near future. Firstly, in order to fit and train the model, a simulation test environment that can be controlled is built to be the source of the actual data. For that a complex geophysical system was designed on CMG reservoir simulation that is capable to produce multiple scenarios that involve multiple wells and geophysical properties. Secondly, a general machine learning model is built using Tensorflow that will be trained on the data and will be the solution for a fast reservoir performance metric for daily use. The model is applied, monitoring the performance of each well individually. Thirdly, manually changing the hyper-parameters of the model to fit the performance and match complexity of the well and reservoir. Finally, the ML output is compared to the actual data from the simulation and tweaked so that it matches perfectly and can show anomaly as early as possible.

4.1 Reservoir model construction

The geophysical reservoir model was imported from Middle East Technical University, Petroleum and Natural Gas Engineering department course (Optimization of petroleum recovery processes) and edited accordingly. The reason this model is used is because of its structural complexity to be as close as real field situation.

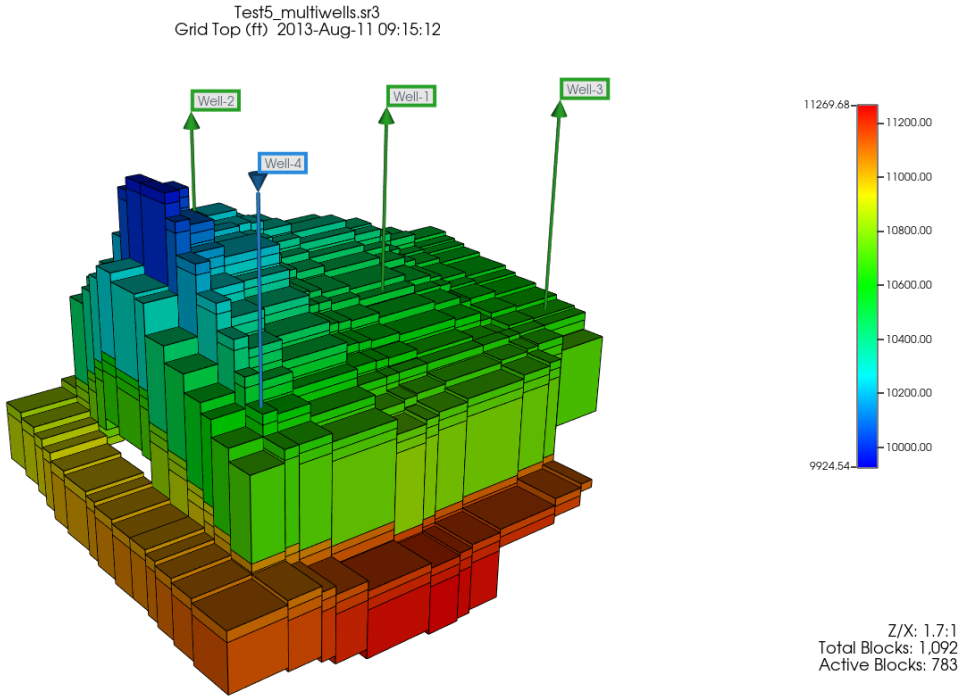

Figure 4.1. Simulated reservoir model

4.1.1 Geological design

4.1.1.1 Reservoir dimensions

There are mainly 6 layers, each differing in its own thickness and reservoir properties. Grid type is Cartesian 13 x 14 x 6 that makes a cuboid shape of 1092 blocks. However, some blocks were deactivated to give us the irregular shape in (Figure 4.1) with total 783 blocks active. The reservoir measures bulk area of 770,000 m^2 (1040 m $*$ 740 m), with a gross thickness 460 m.

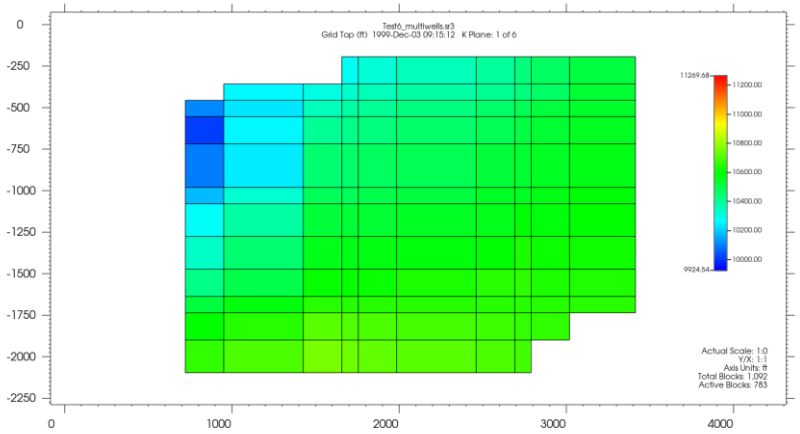

Figure 4.2. Grid-top first layer

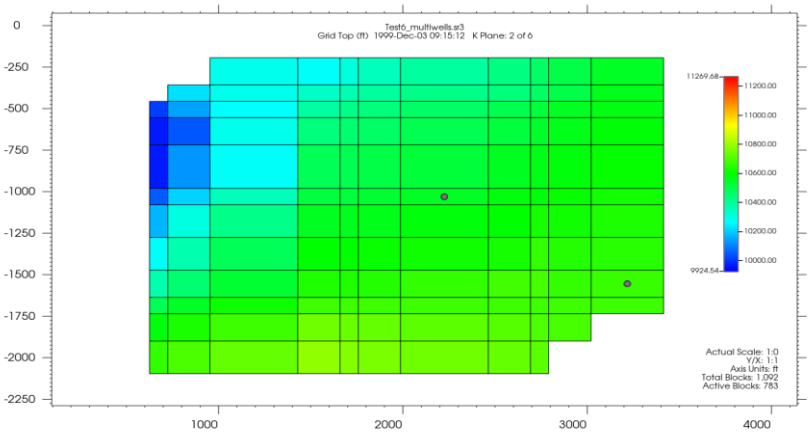

Figure 4.3. Grid-top second layer

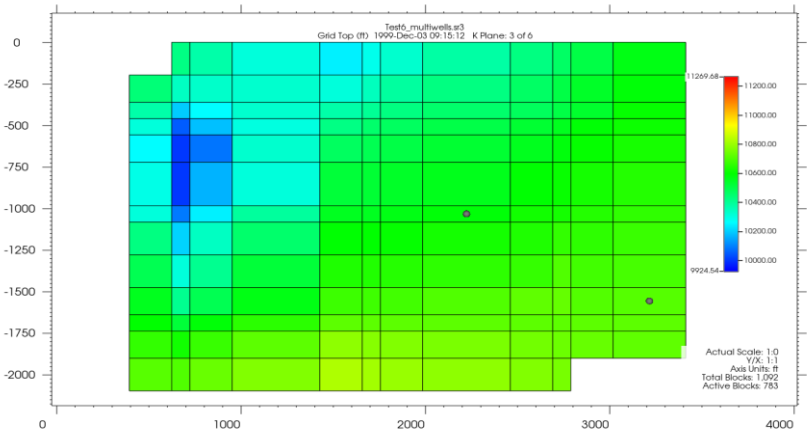

Figure 4.4. Grid-top third layer

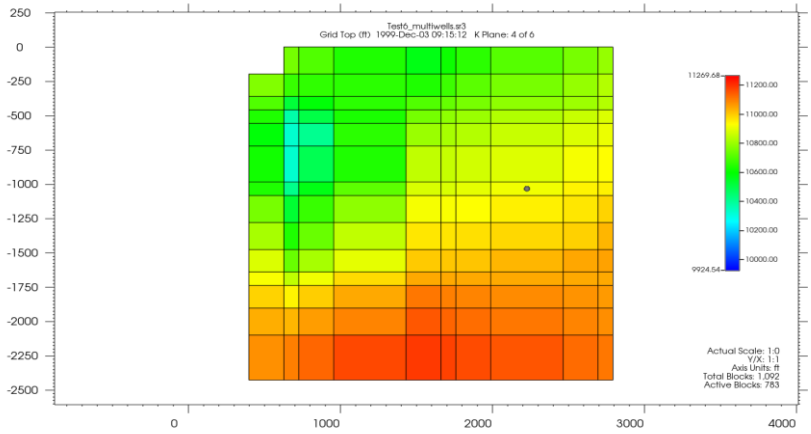

Figure 4.5. Grid-top fourth layer

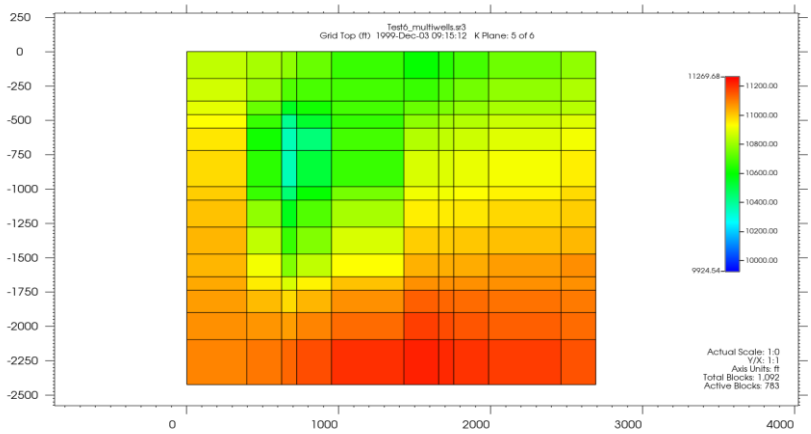

Figure 4.6. Grid-top fifth layer

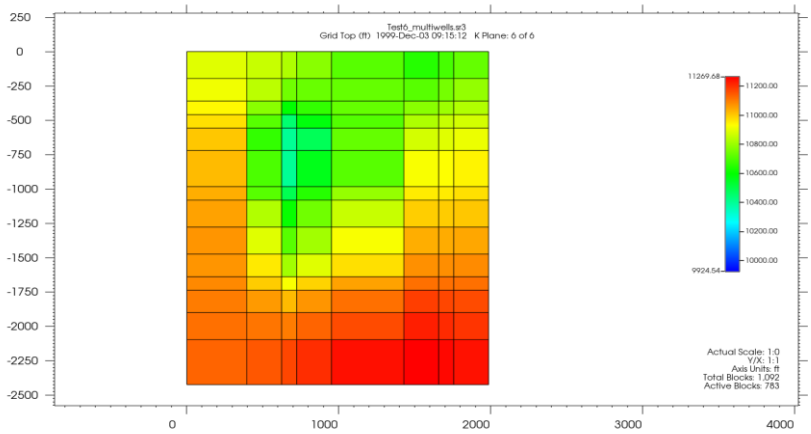

Figure 4.7. Grid-top sixth layer

4.1.1.2 Reservoir properties

Permeability is isotropic in each layer by itself but there are isolated sections where permeability differs (Figure 4.8). The overall porosity, permeability, and thickness of each layer is different (Table 4.1). The gas-oil contact and water-oil contact were defined at (3100 m) and (3385 m) respectively. The reservoir is initially saturated as pressure is less than bubble point pressure. There is Carter-Tracy (Infinite extent) aquifer support attached to the last layer of the reservoir.

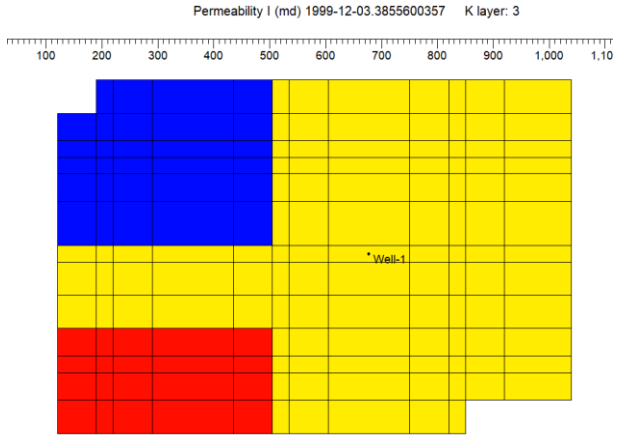

Figure 4.8. Permeability distribution

Table 4.1. Grid Array properties

	Grid Thickness	Porosity	Permeability I
UNITS:	m		md
SPECIFIED:	X	X	X
HAS VALUES:	X	X	X
Whole Grid			
Layer 1	10	0.15	100
Layer 2	12	0.17	300
Layer 3	100	0.25	1000
Layer 4	10	0.1	50
Layer 5	13	0.15	200
Layer 6	60	0.1	20

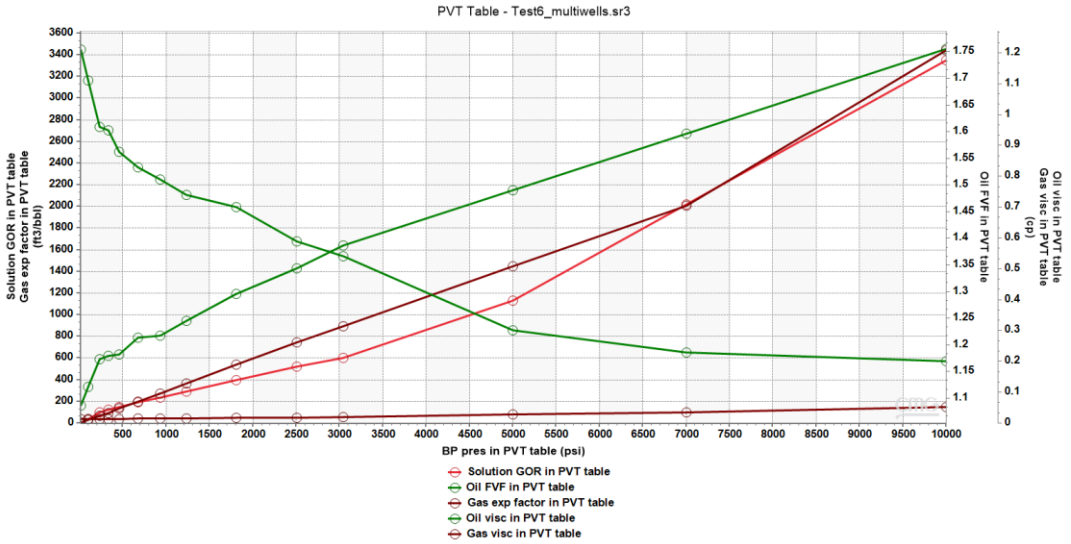

Figure 4.9. PVT table oil-gas

Table 4.2 Reservoir fluid volume

Item	Unit	Value
Total oil in place	m^3	1.20E+07
Total water in place	m^3	6.45E+06
Total gas in place	m^3	2.11E+09
HC. Pore Volume	Mm^3	17520
Total Pore Volume.	Mm^3	23944

4.1.2 Development plan

Here, the idea is to have each well producing oil at a fixed rate different from other wells, while observing pressure change in BHP. Production rates were chosen to be the controllable variable, so well production and injection values are fixed and pressure performance during field life is observed. After that, 5 main parameters are exported to an Excel file to be used later by the ML model.

Many scenarios were developed for testing to make the proxy model more complex and prove that it can handle multi-well design field. However, it summarizes the scenarios development into 3 main algorithm progression check points.

4.1.2.1 Simulated scenarios

In order to advance in the proxy model to handle a verity of different fields with their different needs, multiple scenarios are made in ascending order of complexity on the commercial reservoir simulation.

4.1.2.1.1 Scenario 1

Starting with one well producing at a constant rate and observe the pressure performance at the well location.

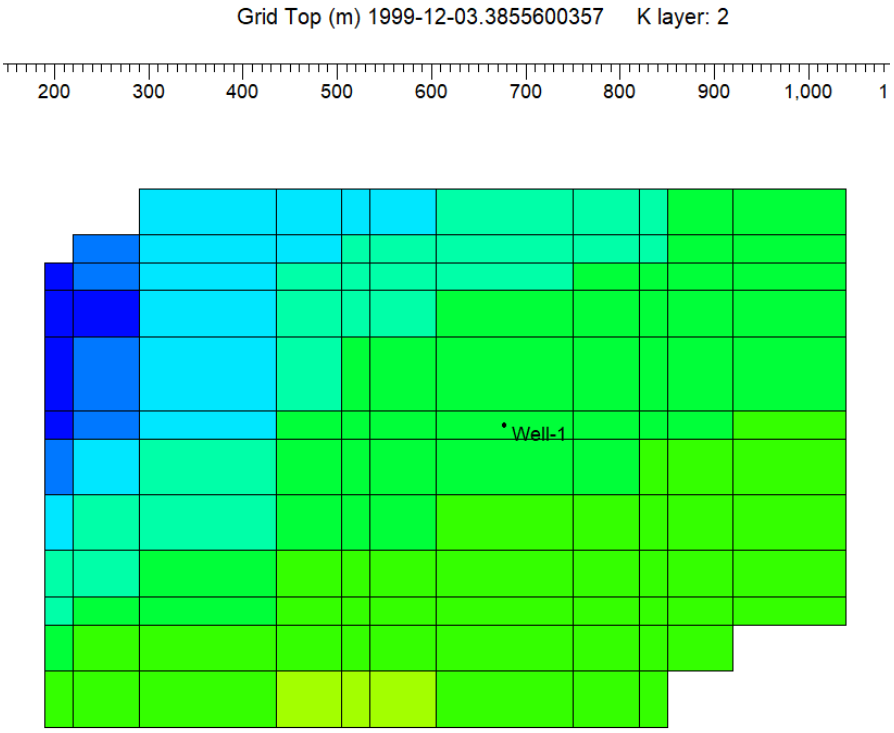

Figure 4.10. Scenario 1: complexity

4.1.2.1.2 Scenario 2

We increased the number of wells into two but fixed their production rates and observed pressure performance from well-1.

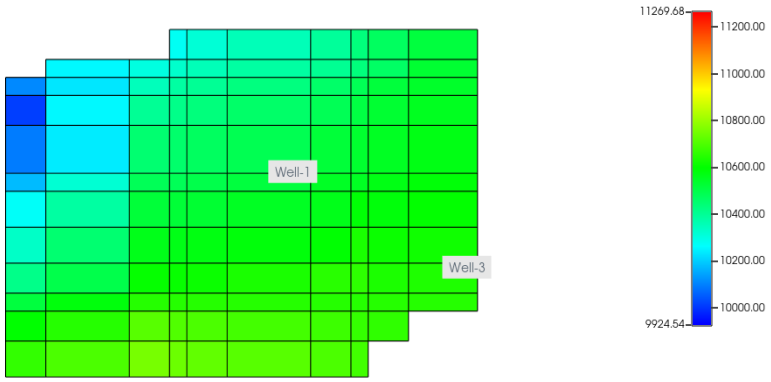

Figure 4.11. Scenario 2: complexity

4.1.2.1.3 Scenario 3

Increasing even further the number of wells, with different rates changes during the lifetime of the reservoir. Here, the problem will be:

1- well-1 and well-3 continue the fixed rate production.
2- Different rates and operational conditions for well-2.
3- Injection from well-4.
4- Well-5 is considered an anomaly with a small production rate that affects pressure on the long run.

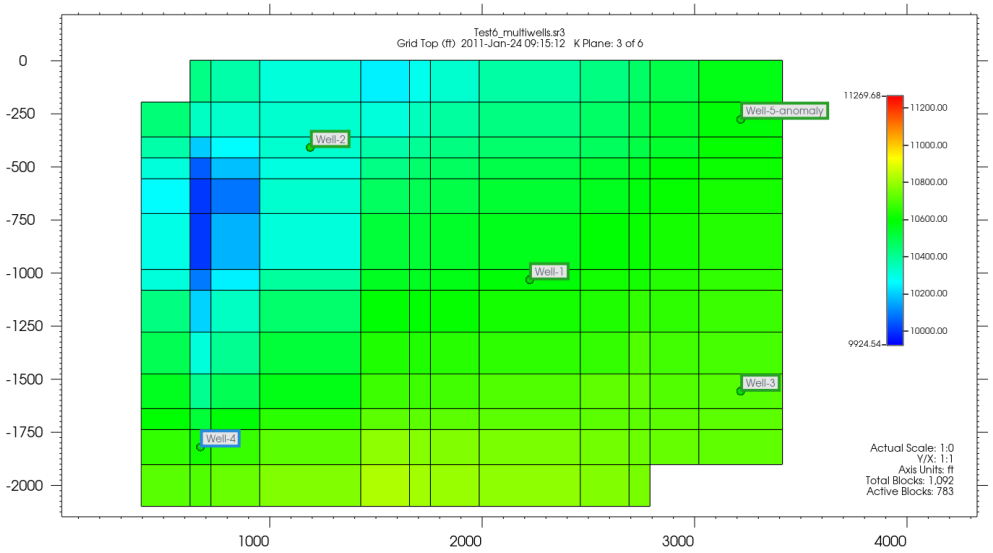

Figure 4.12. Scenario 3: complexity

4.1.2.2 Locations of the wells

Well locations chosen were following the 5-spot pattern for optimum production and injection. Injection location was chosen for its relative higher depth and areal sweep efficiency.

Table 4.3. Wells perforation and depth

	Type	Perforated layers	Depth (m)
Well-1	Producer	1-2	3337
Well-2	Producer	1-2	3257
Well-3	Producer	1	3364
Well-4	Injector	1-2-3	3420
Well-5	Anomaly Producer	1-2	3327

4.2 Machine Learning Model

The Machine learning proxy model is built using the latest version of Tensorflow on a python programming IDE (Spyder). But it can be imported and be used on any machine given the fact that it has the libraries required for the program to run (Appendix B). The model can be tweaked to fit and run on different wells on different fields. A supervised machine learning DNN model takes input data (i.e. time, oil, water, gas rate) and output data (i.e. pressure) and tried to fit its weights accordingly to mimic its forecast to the actual output with respect to the loss function. An input layer with a number of neurons presenting the number of input variables. A number of (hidden) layers, sandwiched between the input layer and the output layer, are presenting the term deep and complexity of our model. An output layer with a single neuron representing the given point of focus variable for performance (pressure). Additional layers can be added to the model to make it more robust and can be added in-between as well. In this case, a gaussian noise layer is added after the input layer to add a noise amount needed.

4.2.1 Input data

Importing the exported Excel data from the CMG simulated model output. Inputs for the model are time, oil production rate, gas production rate, and water production rate. After preprocessing the inputs, they are fed to the algorithm in order to train or predict the future pressure within a time window.

4.2.1.1 Time window

Future pressure is predicted within a specific time window. That time window is the resolution that the user can determine for the model to use. A longer time window will allow the model to predict for a further time in future but will be less accurate. A shorter time window will make the prediction more accurate within a small error

boundary but will debilitate the model of its capabilities. Time window can be set based on the frequency of data that comes, i.e., minutes, hours, days. In this case, the time window is set as 30 days to be predicted in future, and this main job is to catch slight changes presented in the system that can only be seen in the long term. The time window is also set for 1 day to catch any sudden changes and usually has a smaller error margin than the 30^{th} day time window.

4.2.1.2 Preprocessing

Applying preprocessing is an essential part to make sure it is compatible with TensorFlow and for training to be efficient. Preprocessing is defined as:

a. Applying data difference from initial state.
b. Data splitting.
c. Add noise that corresponds to human and pressure gauge error (Figure 4.13).
d. Normalizing data.

4.2.1.2.1 Noise

Noise is manually added using a function to represent the equipment gauges and human error. Figure 4.13 shows noise data resulted. A gaussian noise layer added after the input layer gives the same results.

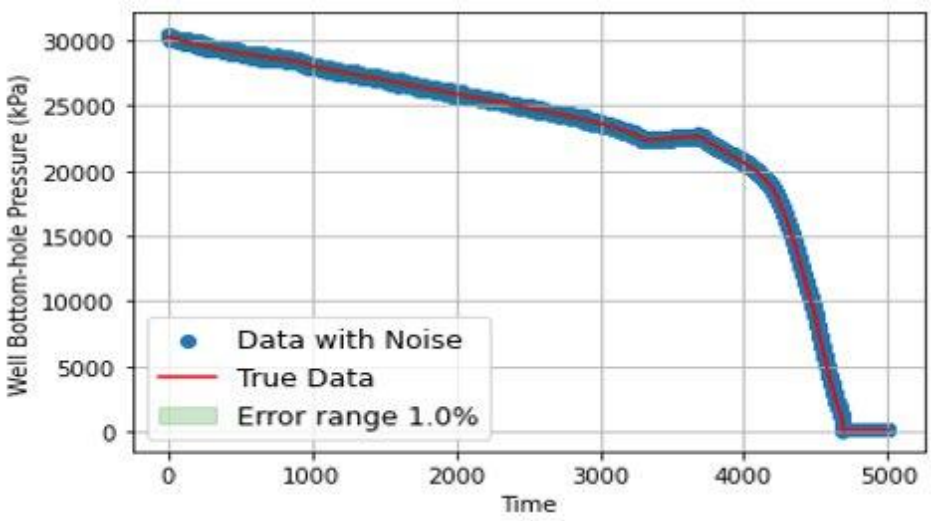

Figure 4.13. Added 1.0% gaussian noise to pressure output.

4.2.1.2.2 Normalization

Normalizing is applied using a Min-Max Scaler function, that records highest and lowest value then normalizes any values in-between 0 and 1.

The scaler factors can be fitted based on expected maximum and minimum values of the parameter. This won't affect the model as long as the scaler being applied consistently during fitting and forecasting of the model. The scaler has already been fitted and used across all wells.

The initial state for the model is set at no production at the beginning. This state can be changed for matured fields where it has already production.

Table 4.4. Preprocessed data stages

State	*Time (day)*	*Oil rate (bbl/d)*	*Gas rate (ft3/d)*	*Water rate (bbl/d)*	*Pressure (psi)*
Initial	0	0	0	0	4395.49
New observed	30	1258	$1.24 * 10^{6}$	$6.69 * 10^{-7}$	4383.65
Added noise (3%)	30	1267	$1.23 * 10^{6}$	$6.68 * 10^{-7}$	4458.28
Difference	30	1267	$1.23 * 10^{6}$	$6.68 * 10^{-7}$	62.79
Normalized	0.0058	0.5636	0.8746	$2.35 * 10^{-10}$	0.9604

4.2.2 Extrapolating new inputs

As mentioned, in order for the model to forecast in the future, it requires the 4 inputs to be present and representative of that time. Assuming to have a control over what oil rate will be, that leaves us with 2 variables (gas and water rate) to be determined. For that it is observed within a short period of time the rate change versus time follows a linear pattern that can be easily extrapolated with suitable Numpy or Scikit library functions.

4.2.3 ML Model construction

The construction here refers to how the model is built and compiled for the optimum learning and testing results. Multiple models were tested including LSTM and autoencoder anomaly detection.

4.2.3.1 Layers and nodes

The number of layers and nodes adds to the complexity of the model. As you increase those numbers you are increasing the dimensionality and capacity. This adds more

weights that need fitting which prolongs training time. In this study, there are in total 4 layers used to build the proxy model. The input layer has 4 nodes for each of the input parameters. The output layer has 1 node for the pressure forecast. Importantly, 2 hidden layers in between representing the depth of the model, and they have 60 each. Each node is connected to all other nodes in the adjacent layers (Figure 4.14). In addition to the noise added to the data, Tensorflow also offers a gausian noise layer that can be added to the model which builds tolerance for the model when noisy data introduced in inputs.

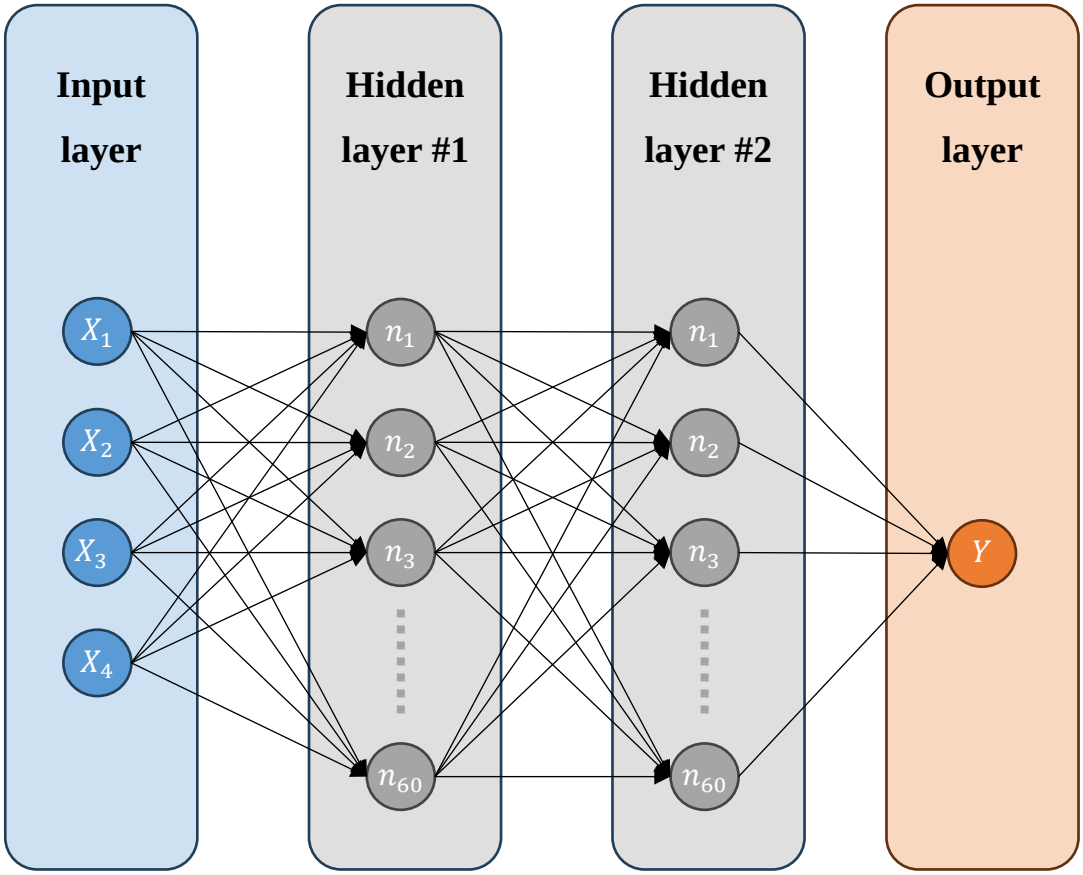

Figure 4.14. Example of layers constructed shapes.

4.2.3.2 Loss function

The loss function job is to calculate the error difference in output from the model and actual data during training. The model goal is to minimize the loss function as

much as possible without overfitting data. It can be choosen which function to be used from the available supported list by Tensorflow. MAE and MSE were both tested and gave similar results. MSE were chosen as loss function.

$$MSE = \frac{\sum_0^n (y_i - \hat{y}_i)^2}{n}$$

4.2.3.3 Optimization method

The optimization method is the technique responsible for updating the gradient during fitting process of the model so it can reach the local minimum error, reported by the loss function. Multiple methods are supported by Tensorflow, only two methods were tested (SGD and Adam). Those are the most widely known methods used in the industry. Adam gave a better fitting overall (Figure 4.15), so went with it.

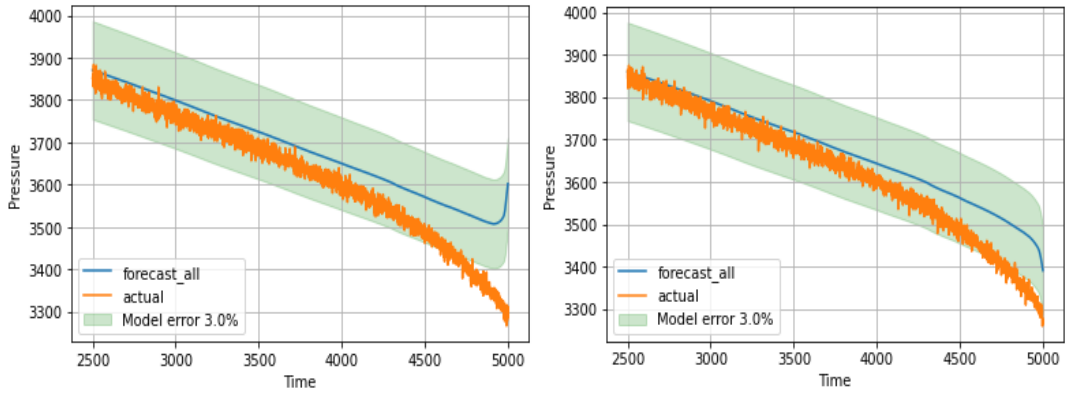

Figure 4.15. SGD (left) versus Adam (right)

4.2.3.4 Learning rate

Learning rate is another hyperparameter that defines the amount of step or adjustment the gradient can take. A very high LR allows the model to learn faster but can result in making it miss the global minima of error. A very low LR will take

the model too long to reach the global minima. So here it is testing the suitable learning rate by increasing it periodically and testing it against loss.

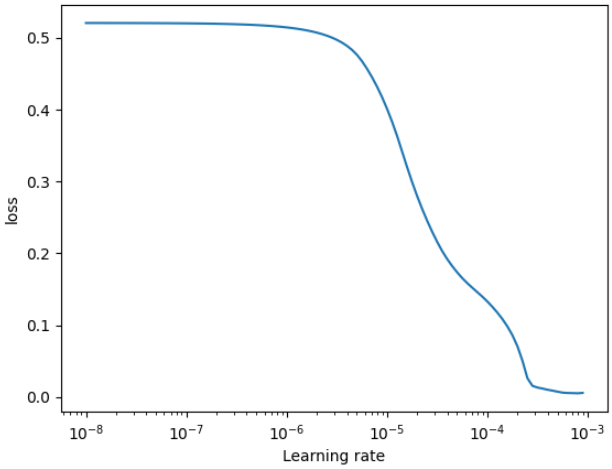

Figure 4.16. Loss vs Learning rate

If needed to continue training the model when new fresh data becomes available, it is important to carefully choose the learning rate here as it could alter weights significantly those results in mismatching the earlier pressure performance.

4.2.3.4.1 Dynamic LR

Another idea introduced is adaptive or dynamic LR. Basically, a threshold is set for the model to compare new errors introduced. Each threshold has its own LR predefined. If it is higher than this threshold, it means that the model forecast is far away from actual and needs to adjust the LR for a higher value to learn faster. Thus, underfitting the model problem is reduced.

4.2.3.5 Model optimization

A couple of methods were used to optimize the model for its hyperparameters. Including number of epochs used, number of units in each layer, learning rate, and

more. By utilizing step wise change in LR and observing at which point the curve is at its lowest, that should give the sweet spot for the suitable LR that result into smallest error (Figure 4.18). Tensorboard utility is available for checking the model performance. By preparing the observation points of interest in the model, hyperparameters can be checked against the model performance (Figure 4.20).

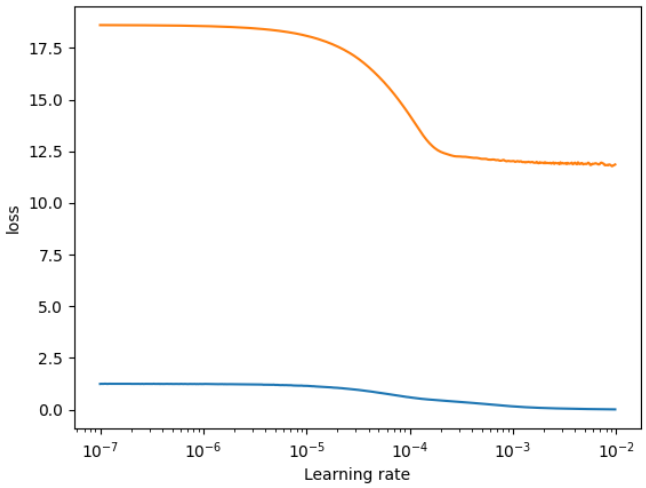

Figure 4.17. Loss vs learning rate

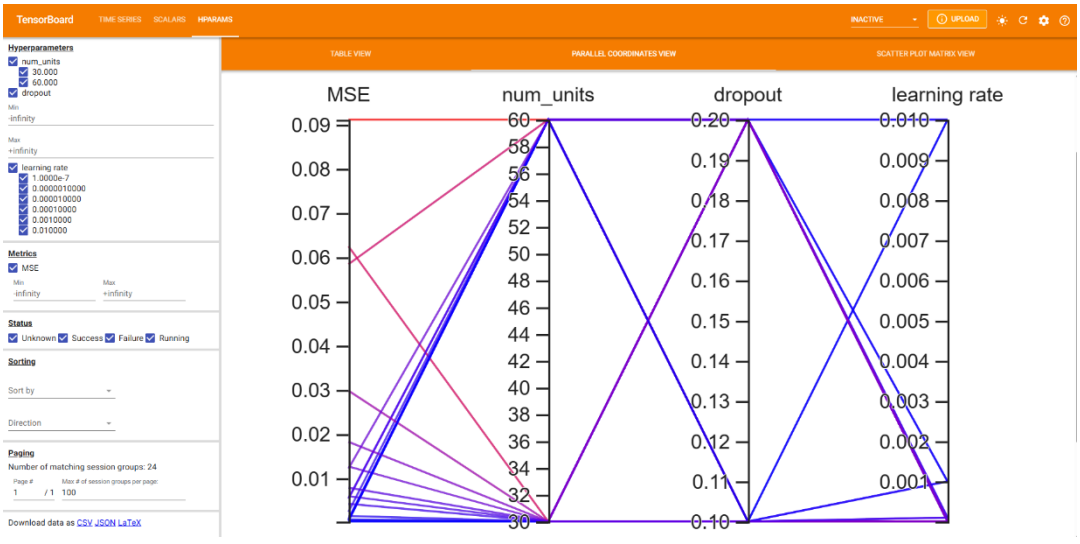

Figure 4.18. Tensorboard utility for hyper-parameters observation

4.2.4 Output data

The raw output from the proxy model needs to go through post-process in order to be a representative pressure data that can be compared against actual data. Post-process is composed of de-normalizing and adding back the difference in pressure.

4.2.4.1 Training and Forecasting

The first-time training process is crucial to have a representative model. The amount of data available for training can determine some hyperparameters, like LR and epochs. As soon as having a trained model, it can be saved, exported, or used to forecast the inputs accordingly. Forecasting takes a fraction of a second.

4.2.4.2 Anomaly detection

This algorithm approach for detecting anomaly is by calculating the expected model error after forecasting and comparing it to true error after actual data is available at the time. In order to do that, three things were established:

1- The model keeps updating and training when each new value is available given that the new forecast error is bigger than a threshold.
2- Defining variables that memorize the model expected forecast at some specific times within the “time window”.
3- Comparing and extrapolating errors from those variables against actual pressures

Forecasting error performance is bound to increase with time. The frequency at which the model updates when new data is available keeps the model in check and the threshold error at its accepted rate.

There are two variables defined. One variable collects the next day predicted forecast. Another variable collects the end of time window predicted forecast (i.e.,

predicted forecast after 30 days passing). The reasoning behind this is when those variables are compared to actual data, they produce percentage errors. Those errors are then memorized and extrapolated to estimate how much error is to be expected in the future forecasts.

Setting up the anomaly detection algorithm this way allows detection of sudden and smooth changes in pressure performance that is not a local change in well conditions that caused it. To name the possible range of pressure for day 1 prediction to be "Range 1", and the possible range of pressure for day 30 prediction to be "Range 30". Consequently, three keys emerge where it can identify anomaly:

1- Actual pressure $\notin$ "Range 1" & "Range 30". That means sudden change in pressure not expected by both. **(Condition A)**
2- Actual pressure $\notin$ "Range 30" but Actual pressure $\in$ "Range 1". That means it was a gradual pressure change that day 1 prediction adapted to the change but day 30 didn't. **(Condition B)**
3- "Range 1" $\notin$ "Range 30". It means the day 1 forecast does not follow same pattern as day 30 forecast. **(Condition C)**

4.2.4.3 Anomaly region

As part of the algorithm, I wanted to show where was first the anomaly is detected by the model and the duration it took to adapt on the anomaly. Hence, a region was highlighted on the plot graphing interface representing where the model has detected any anomalies.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Reservoir simulation scenarios pressures

Scenario 1 had a 1 well producing at a constant rate, it produced pretty much a linear decline. An abnormal pressure was introduced to see how the model will adapt to the sudden pressure change (Figure 5.1).

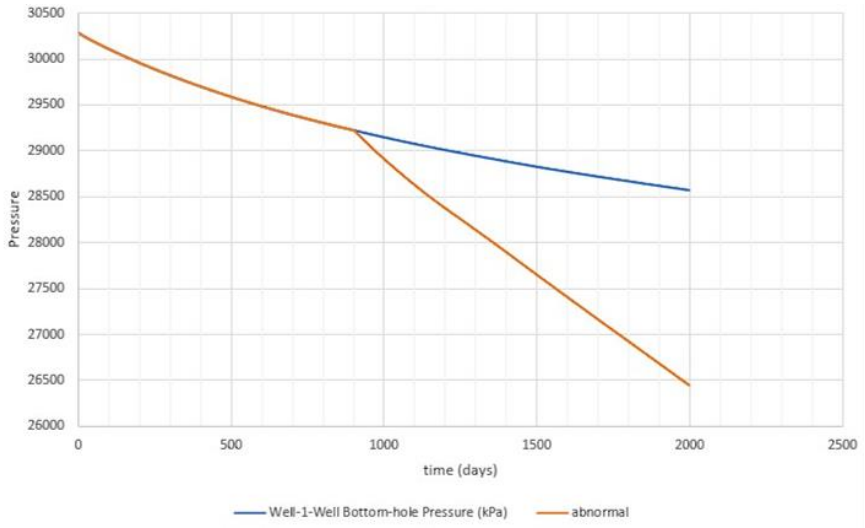

Figure 5.1. Normal expected pressure (Blue) versus Abnormal pressure (Orange)

As two wells, in scenario 2, are producing at a fixed rate within a finite reservoir area, BHP performance can be seen in (Figure 5.2).

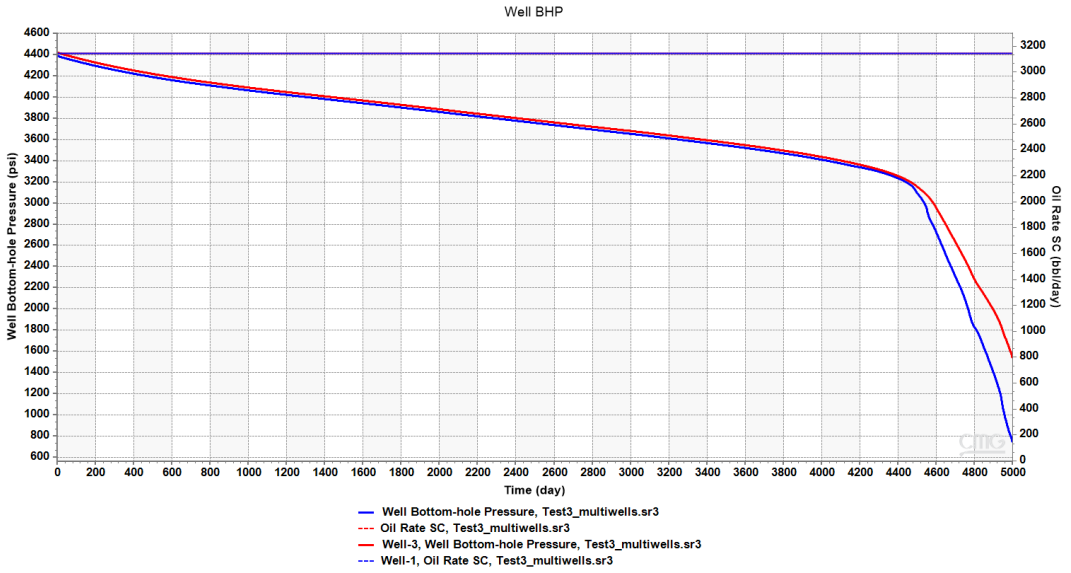

Figure 5.2. Scenario 2: Fixed oil rate production of two wells.

Scenario 3 increases the complexity of 5 wells operating at the same time with different rates (Figure 5.3).

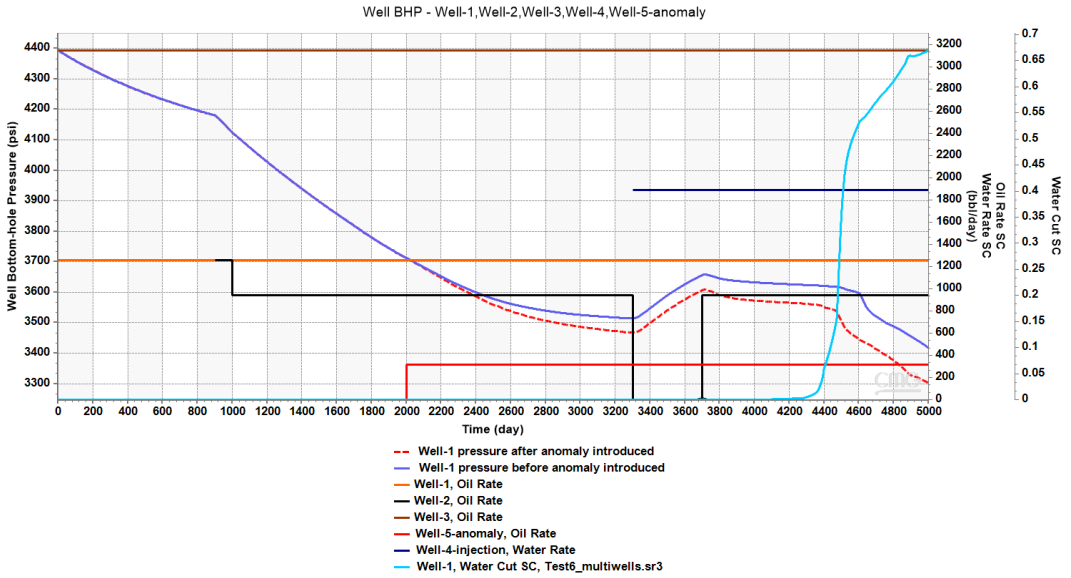

Figure 5.3. Scenario 3: pressure and production rate vs time. Well-1 pressure before (purple) and after (dashed red) anomaly introduced. Oil rate for Well-1 (Orange), Well-2 (black), Well-3 (brown), Well-5(red), and water injection rate for Well-4 (Dark blue). Water cut in Well-1 (sky blue).

5.2 ML and Anomaly detection performance

5.2.1 Algorithm stage 1

Scenario 1 complexity is no match to the complexity of the model. Hence, the model had no issue training and forecasting (Figure 5.5), given the fact that initially the model is trained on 1600 data points (Figure 5.4). When anomaly was introduced, the model showed an anomaly region and quickly adjusted itself to represent actual performance (Figure 5.6). Although anomaly introduced on day 900, model took long time to find it, and the reason is that the error limit that determines an anomaly is high. So, to fix this problem a changing error and adapting error boundary based on new forecast is applied in the final scenario. A second problem can be seen in (Figure 5.7) is when not correctly having correctly fitted scaler and stop training the model regularly. In here input data are tested first with no noise (Figure 5.8) then with noise (Figure 5.9) added to see model performance.

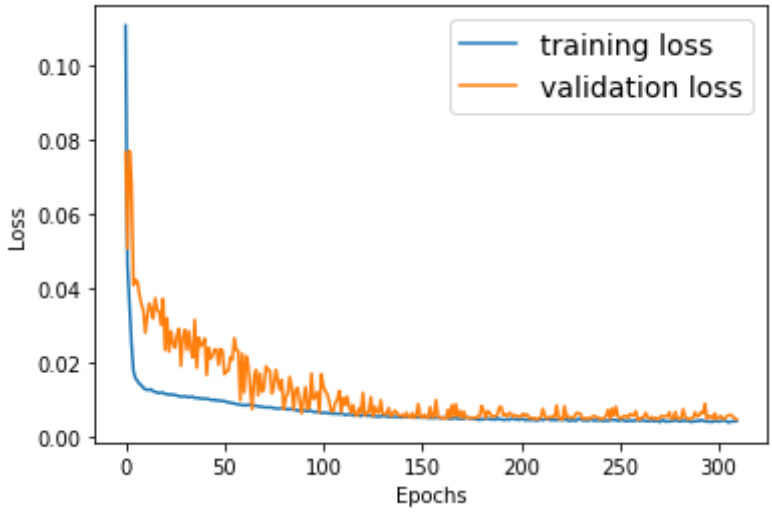

Figure 5.4. Loss decreases with time during model training.

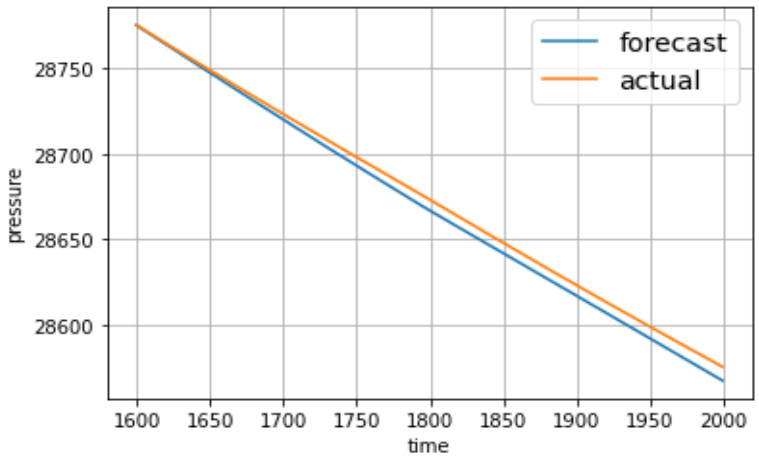

Figure 5.5. ML model forecasting pressure performance between day 1600 to day 2000.

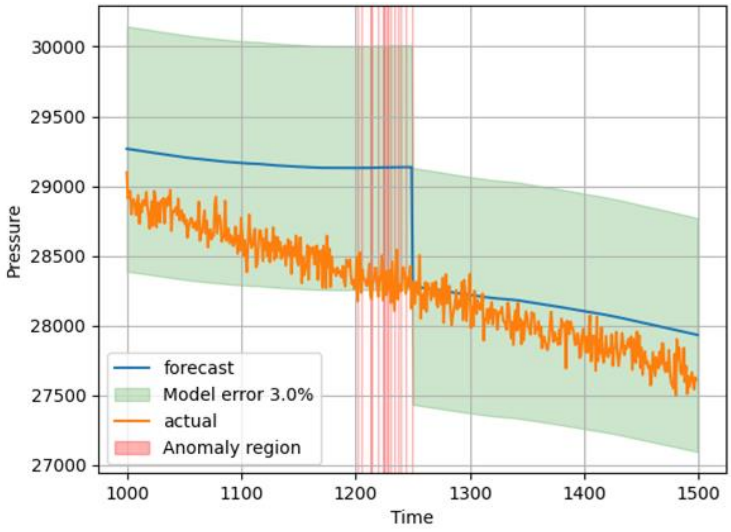

Figure 5.6. Scenario 1 identification of anomaly, and training for adjustment.

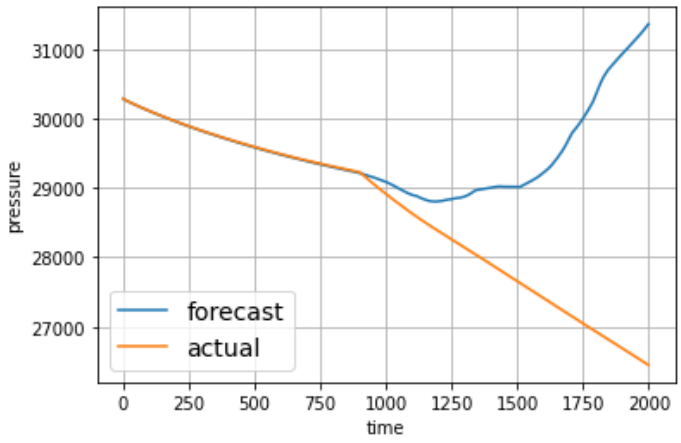

Figure 5.7. One-time trained model struggling to forecast abnormal pressures.

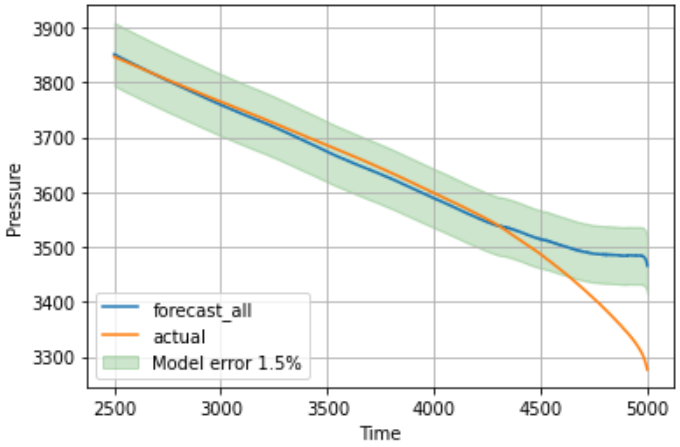

Figure 5.8. No noise inputs and outputs data and the model performance.

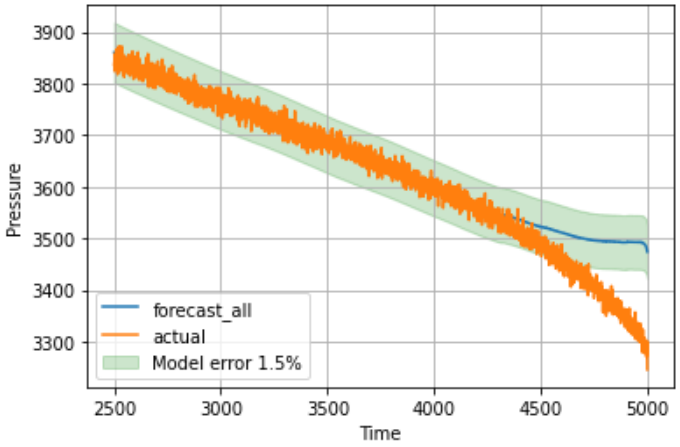

Figure 5.9. Noise (1%) applied in inputs and outputs data and the model performance.

5.2.2 Algorithm stage 2

This scenario, although it introduces two well producers with fixed production, the model appears to be sharply declining which again imposes no problem from training (Figure 5.10) and forecasting. However, it is shown in (Figure 5.11) that high LR will allow the model to adjust faster but can't find the local minima; hence, it oscillates back and forth between actual pressure. In contrast, (Figure 5.12) had low LR it adjusted slowly without oscillation. Consequently, applying a dynamic LR has been developed in the algorithm and result can be seen in (Figure 5.13) where the model learns faster with no oscillation applied.

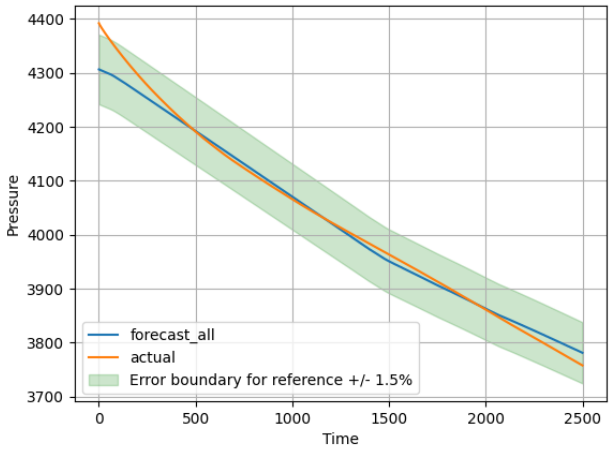

Figure 5.10. Scenario 2 initial model fitting.

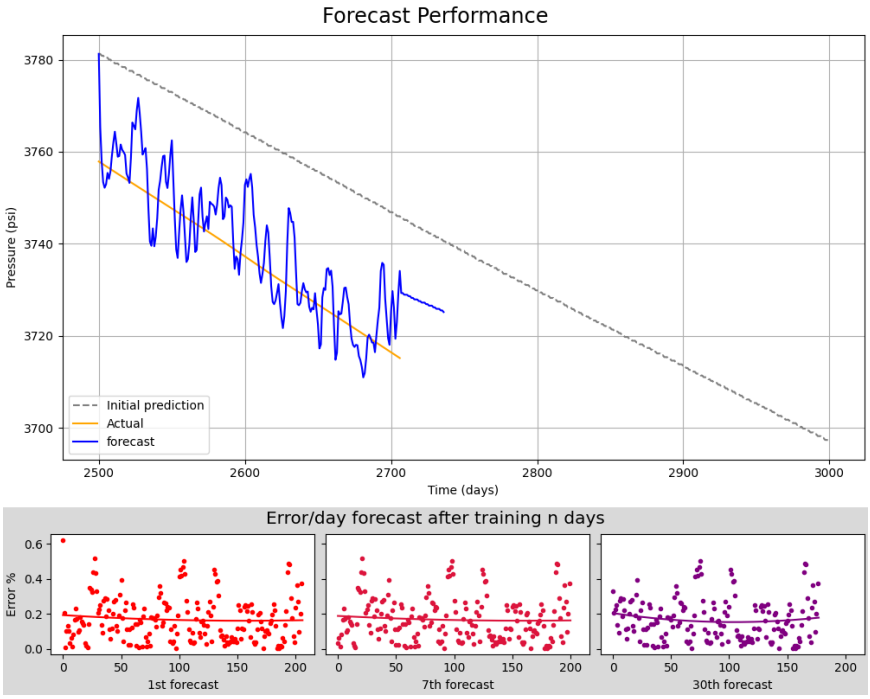

Figure 5.11. High LR (1e-5) and the model adjustment to pressure change

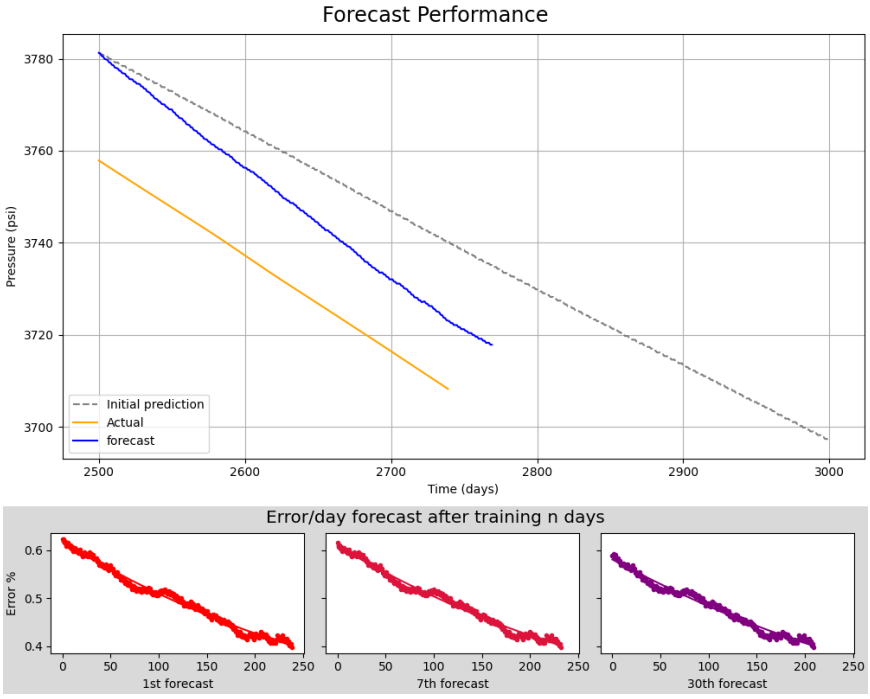

Figure 5.12. Low LR (1e-7) and the model adjustment to pressure change

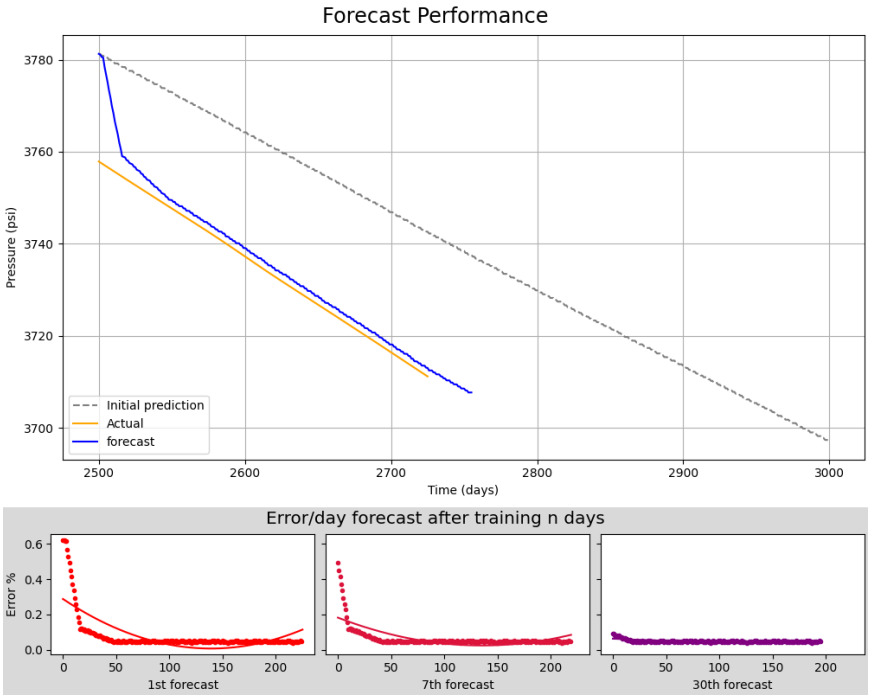

Figure 5.13. Dynamic LR, a step wise change in LR to allow model to learn faster with high error, and slowdown in learning when error is relatively smaller.

5.2.3 Algorithm stage 3

Focusing on well-1 forecasting and detection. Here, initiated training the model on only a small portion of data. For example, well-1 (Figure 5.14) trained only on 50 days initially. The standard was set for the acceptable model error, from initial model, to be less than 0.3%. Then the model is saved and used to continue the training and forecasting cycle.

Looking at Scenario 3 (Figure 5.3), it is seen at least 5 major changes in pressure due to external influence of other wells on the observed well-1. The goal is to identify those changes from well-1 perspective; therefore, scenario 3 is divided into 5 cases (Figure 5.15).

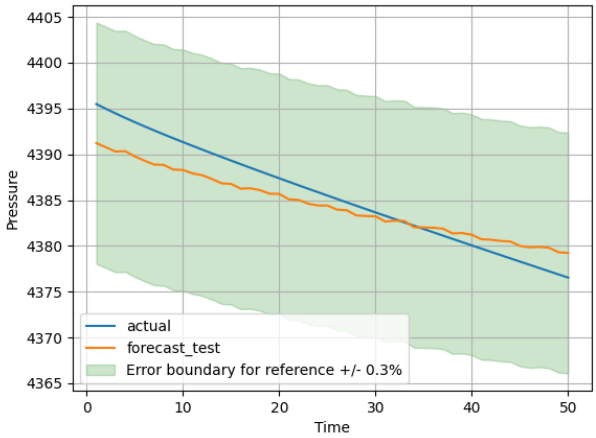

Figure 5.14. Scenario3: model trained initially on 50 days and its forecast error is less than 0.3% (13 psi) for well-1.

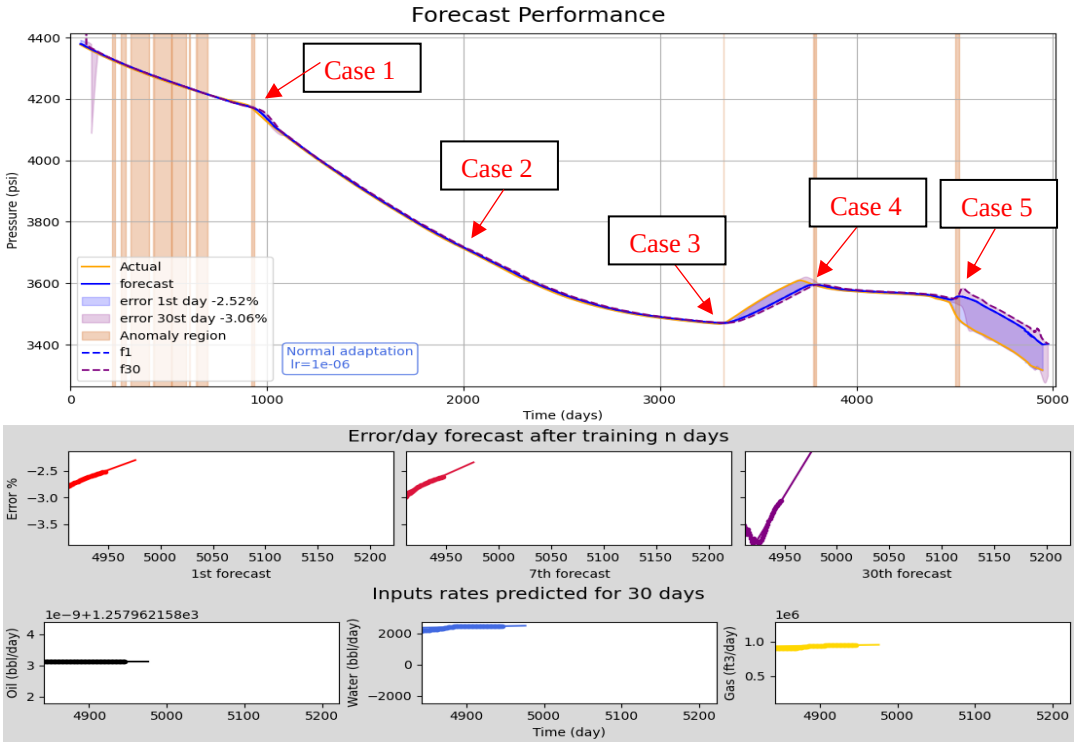

Figure 5.15. Well-1 algorithm detected error during the whole simulation run. Red arrows point towards error catched.

5.2.3.1.1 Case 1

Well-2 started production at 900 days. The algorithm predicted the there is an error at 921 days, 21 days after well-2 appeared (Figure 5.16). Error ranges are calculated by firstly fitting data to the last couple of errors it seen, then extrapolate (Figure 5.17). A linear regression is used here because after testing polynomial regression, it gives more noise to the error. Due to errors not fitting perfectly to the regression, an error range is added to the extrapolated error.

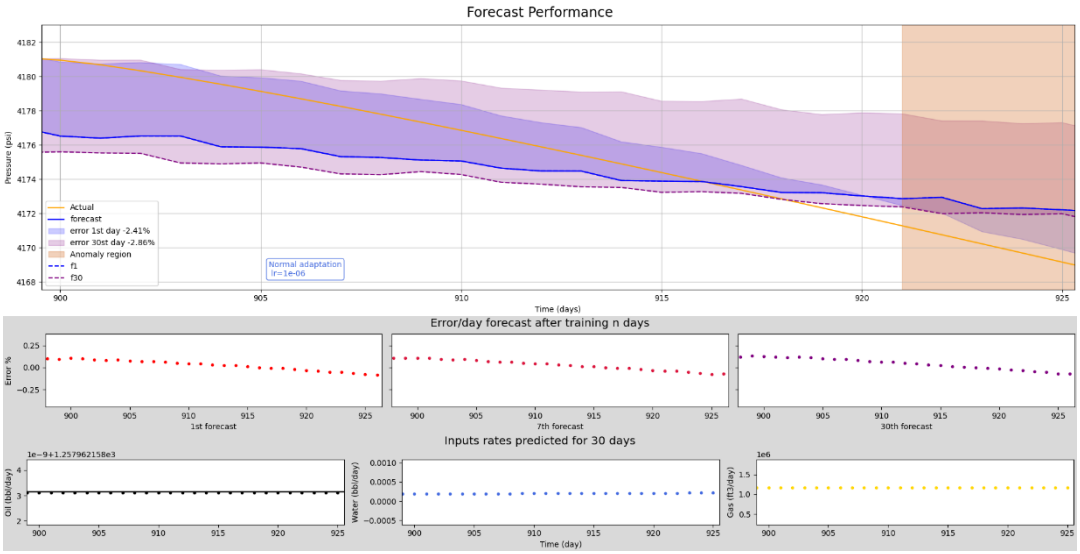

Figure 5.16. Case 1, Well-1 algorithm detected error on day 921.

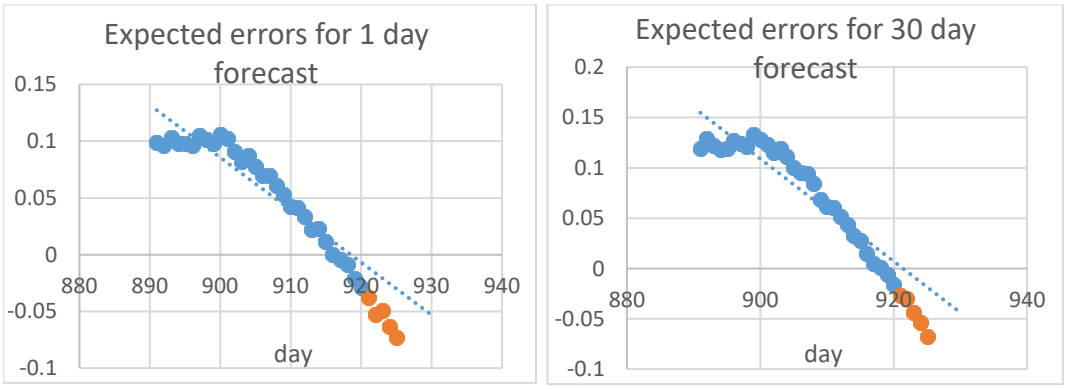

Figure 5.17. Algorithm regression on well-1 errors and extrapolating to find anomalies.

5.2.3.1.2 Case 2

Well-5, the anomaly, has started producing at a very low rate (315 bbl/d). This well is introduced as an anomaly that could belong to another company and out of this reservoir management plan. Unfortunately, it has not been detected by well-1 due to the change in pressure is very small as well as the model keeping adjusting to new pressure values as being the representative value of the current field.

However, another well-2 in can be used to fit another model with the same algorithm and observe well-2 performance. It got detected by it (Figure 5.18).

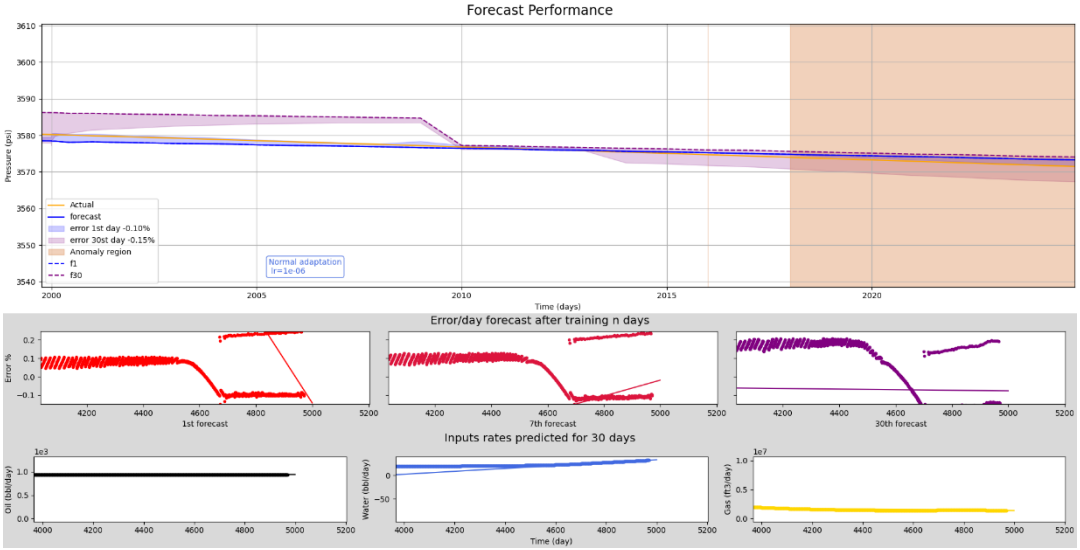

Figure 5.18. Case 2, Well-2 prediction found anomaly on day 2016.

5.2.3.1.3 Case 3

Here, Well-2 has stopped production, and Well-4 started injection as part of field development after 3300 days. A change in pressure was detected (Figure 5.19) on day 3326, 26 days after.

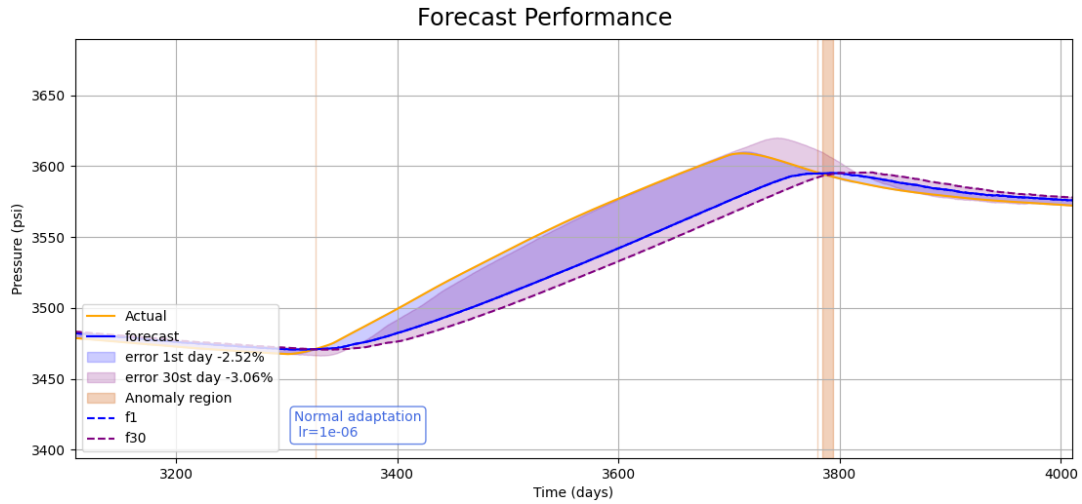

Figure 5.19. Case 3 and Case 4 anomaly detected by well-1 forecast performance graph on day 3326 and 3780 respectively.

5.2.3.1.4 Case 4

On day 3700, Well-2 return to production, and the pressure change was seen by well-1 on day 3780, 80 days later (Figure 5.19).

5.2.3.1.5 Case 5

Lastly, observing a sharp decline in pressure performance due to pressure support not being enough to hold pressure at the current development plan. It is also observed that a water flux increases in the well production (Figure 5.3). This change was also detected by the algorithm in well-1 on day 4501 (Figure 5.20).

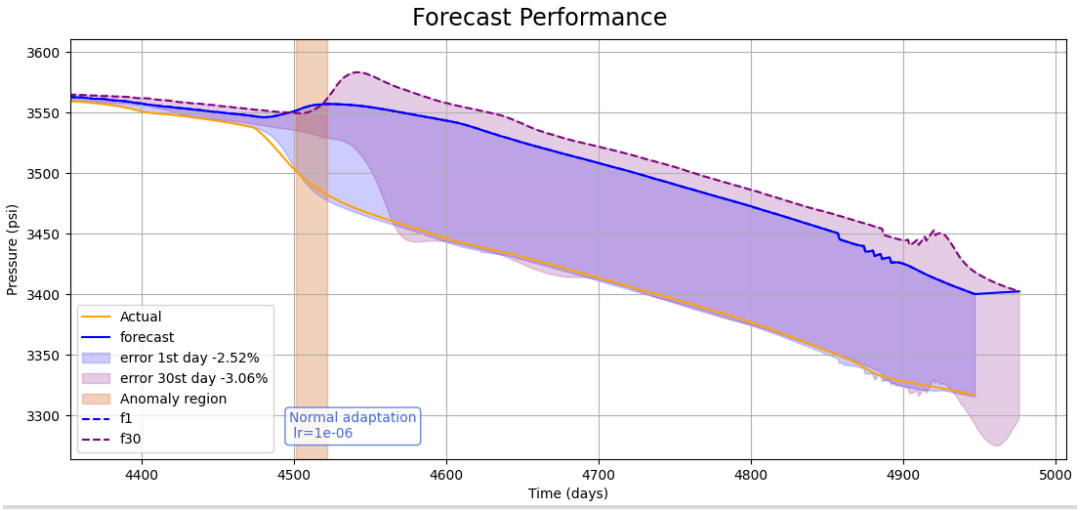

Figure 5.20. Case 5 anomaly detected on day 4501.

From (Table 5.1), it is seen that the model catches all sudden changes within a delayed time. Although, the model didn't show graphically the gradual change caused by well-5 production, it was found by **condition B** in the table above. Because (Table 5.1) is also exported data from the model, it is known that the ML algorithm successfully caught the anomaly, yet some work on the detection algorithm needs to be re-adjusted to be graphically shown. This can be further improved with a usable interface for the ease of use of engineers.

Table 5.1. Model predictions, error calculated, and anomalies found and shown graphically.

days	Actual before anomaly	Actual	day1 pred	1st error %	day 1	day30 pred	30th error %	day 30	Anomaly	
899	4181.1	4181.1	4177.0	0.1	4181.1	4175.5	0.1	4180.8	0	
900	4180.9	4180.9	4176.5	0.1	4180.9	4175.6	0.1	4180.8	0	Condition B (1.1 psi)
919	4172.3	4172.3	4173.2	0.0	4172.3	4172.6	0.1	4177.7	0	
920	4171.8	4171.8	4173.0	0.0	4171.8	4172.5	0.1	4177.4	0	
921	4171.3	4171.3	4172.9	0.0	4171.3	4172.4	0.1	4177.8	1	
922	4170.7	4170.7	4172.9	-0.1	4170.7	4172.0	0.1	4177.1	1	
1999	3713.2	3713.2	3717.0	-0.1	3713.2	3719.7	-0.2	3712.6	0	
2000	3713.2	3712.9	3716.7	-0.1	3712.9	3719.7	-0.2	3712.6	0	Condition B (0.7 psi)
2025	3705.4	3704.7	3708.6	-0.1	3704.7	3713.4	-0.2	3706.8	0	
2026	3705.1	3704.4	3708.0	-0.1	3704.4	3713.1	-0.2	3706.5	0	
2027	3704.8	3704.0	3707.8	-0.1	3704.0	3712.8	-0.2	3706.3	0	
3299	3514.0	3467.5	3470.9	-0.1	3467.5	3471.9	-0.1	3467.5	0	
3300	3514.0	3467.4	3471.0	-0.1	3467.4	3471.9	-0.1	3467.5	0	Condition B (1.7 psi)
3324	3517.5	3470.6	3470.9	0.0	3470.6	3470.8	-0.1	3466.3	0	
3325	3517.8	3470.8	3470.9	0.0	3470.8	3470.8	-0.1	3466.3	0	
3326	3518.0	3471.0	3470.9	0.0	3471.0	3470.7	-0.1	3466.3	1	
3699	3656.4	3607.5	3575.3	0.9	3607.2	3565.8	1.2	3608.4	0	
3700	3656.4	3607.8	3575.7	0.9	3607.5	3565.8	1.2	3608.4	0	Condition A (3.5 psi)
3701	3656.9	3608.0	3576.0	0.9	3607.7	3566.5	1.2	3609.1	0	
3780	3648.9	3595.47290	3594.7	0.0	3595.47274	3591.9	0.5	3611.3	1	
4485	3617.0	3524.7	3546.8	-0.6	3524.6	3551.2	-0.4	3538.1	0	
4490	3616.4	3517.6	3548.0	-0.9	3517.4	3550.6	-0.4	3537.2	0	Condtion B (47.7 psi)
4495	3615.8	3510.3	3549.7	-1.1	3509.8	3550.3	-0.4	3536.2	0	
4500	3615.2	3503.4	3551.2	-1.4	3502.8	3549.8	-0.4	3535.2	0	
4501	3615.0	3502.1	3551.6	-1.4	3501.4	3549.8	-0.4	3535.2	1	

5.3 Other ML models performance

5.3.1 LSTM

LSTM is a type of RNN but performs better on long forecasting. For training and forecasting it only depends on one parameter changing per time. Hence, compared to the DNN model, LSTM would recognize any changes in production rates or operational conditions to be an anomaly. Although trained on the same data, it requires different preprocessing, and can only old pressure as input and it forecasts the new pressure. Unfortunately, it was not successful in predicting (Figure 5.16) shows a constant pressure all the time.

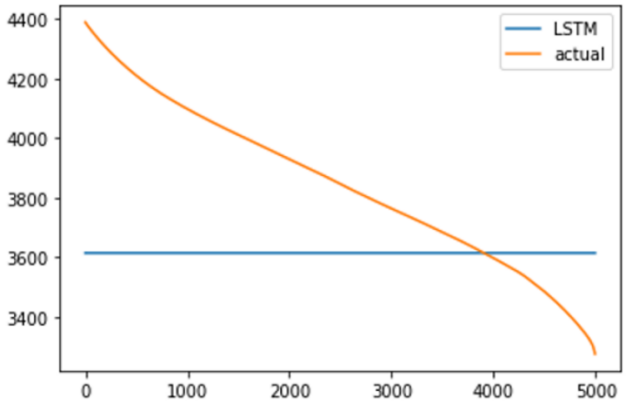

Figure 5.21. LSTM predicts pressure as constant and place it where it minimizes the error as much as possible during training.

5.3.2 Autoencoders – Unsupervised anomaly detection.

Unsupervised anomaly detection is a type of ML that can detect errors in abnormal changes in data giving the fact that it needs training on normal data patterns for as much as it needs. When this ML applied in this study however, it presented error only if it is outside the range of “normal” data. (Figure 5.17) shows both anomaly situations, however it recognized only parts that fall outside the range of (4400 to

3550 psi) as anomaly. The reason for that is normal data that the model trained on was also within the same range.

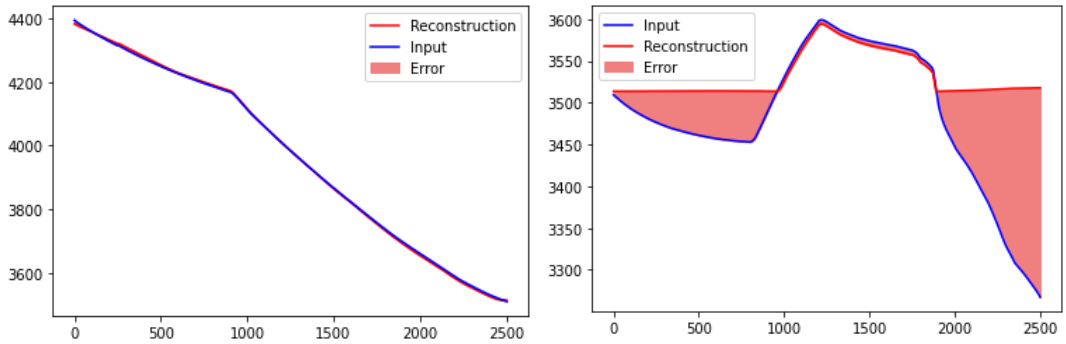

Figure 5.22. Autoencoder recognizes no anomaly on the left example and some parts of anomaly on the right example.

CHAPTER 6

CONCLUSION

In this study, a numerical simulation model has been developed using a commercial reservoir simulator to produce multiple problems that can be studied, and developed on the algorithm based on it. The aim was to develop and test the algorithm for forecasting reservoir pressure performance and anomaly detection under different scenarios. The model was trained on historical data before any anomalies and validated on new data with anomalies and noise.

The main findings of the thesis are:

1. The machine learning model was able to accurately forecast the pressure performance in scenario 1, where the complexity of the reservoir was low, and the pressure decline was linear. The model was able to detect and adjust to the anomaly introduced in the data, as well as to handle the noise added to the inputs and outputs. Performance was improved by using a fitted scaler.
2. The machine learning model was also able to forecast the pressure performance in scenario 2, where the complexity of the reservoir was slightly higher, and the pressure decline was sharper. The model was able to adapt to the changes in the production rates of the two wells, but the learning rate was a critical factor for the model adjustment. A high learning rate caused the model to oscillate around the actual pressure, while a low learning rate allowed the model to converge to the local minima, so a dynamic learning rate was introduced to the algorithm.
3. The machine learning model showed promising results for forecasting reservoir pressure performance and detecting anomalies in scenario 3, where the complexity of the reservoir was the highest and the pressure decline was the most irregular. The model was trained on a small portion of data initially

and then updated and validated on new daily incoming data. The model was able to catch all sudden in the pressure performance of well-1, with some delay and error. Although well-1 failed to find the gradual change given in case 2, when the algorithm was applied to well-2, it detected the anomaly from Well-2 forecasting performance.

4. The algorithm demonstrated its robustness and flexibility to learn beyond its original knowledge when it is applied in a single well or as a group.

The thesis contributes to the field of reservoir engineering by providing a novel and effective method for forecasting reservoir pressure performance and detecting anomalies using machine learning. It also demonstrates the potential of machine learning for solving complex and nonlinear problems in the oil and gas industry.

CHAPTER 7

RECOMMENDATIONS FOR FUTURE WORK

This study can be further improved with the following items:

1. Applying the machine learning model to other reservoirs and scenarios and comparing its performance with conventional methods.
2. Further improving dynamic learning rate for a faster and more accurate detection of anomalies.
3. Developing a user-friendly interface for the machine learning model that can facilitate ease of use for engineers.
4. Creating a method to analyze all anomalies found from other model-well output and generate a heatmap for the possible source location of the problem.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. & others (2016). TensorFlow: A System for Large-Scale Machine Learning. OSDI (p./pp. 265-283).

Ahmadi, M. A., Zendehboudi, S., & James, L. A. (2018). Developing a robust proxy model of CO2 injection: Coupling Box–Behnken design and a connectionist method. *Fuel*, *215*, 904–914. https://doi.org/10.1016/J.FUEL.2017.11.030

Akin, S., Kok, M. v., & Uraz, I. (2010). Optimization of well placement geothermal reservoirs using artificial intelligence. *Computers & Geosciences*, *36*(6), 776–785. https://doi.org/10.1016/J.CAGEO.2009.11.006

Alharbi, B., Liang, Z., Aljindan, J. M., Agnia, A. K., & Zhang, X. (2022). Explainable and Interpretable Anomaly Detection Models for Production Data. *SPE Journal*, *27*(01), 349–363. https://doi.org/10.2118/208586-PA

Arıtürk, M. S. (2019). *Optimizing the Production and Injection Wells Flow Rates in Geothermal Field Using Artificial Intelligence*. [Master's dissertation, West Virginia University]. West Virginia University Research Repository. https://researchrepository.wvu.edu/etd/3772

Artun, E. (2022). Machine Learning Assisted Forecasting of Reservoir Performance. *Machine Learning Applications in Subsurface Energy Resource Management*, 185–206. https://doi.org/10.1201/9781003207009-14

Ashri, R. (2020). What Is AI. *The AI-Powered Workplace*, 15–29. https://doi.org/10.1007/978-1-4842-5476-9_2

CMG (2021). CMG STARS Reservoir Simulation Software, Computer Modeling Group Ltd. Calgary, Alberta, Canada.

Coursera. (2023). *What Is Machine Learning? Definition, Types, and Examples*. https://www.coursera.org/articles/what-is-machine-learning

Covarrubias-Moreno, O. M. (2022). *Artificial Intelligence and Systems Thinking in the Public Sector* (pp. 35–54). https://doi.org/10.4018/978-1-6684-5624-8.ch002

Ertekin, T., & Sun, Q. (2019). Artificial Intelligence Applications in Reservoir Engineering: A Status Check. *Energies 2019, Vol. 12, Page 2897*, *12*(15), 2897. https://doi.org/10.3390/EN12152897

Feder, J. (2020). Machine-Learning Approach Improves Deepwater Facility Uptime. Journal of Petroleum Technology, 72(05), 54–55. https://doi.org/10.2118/0520-0054-JPT

Freitas, M. A. de. (2018). Inteligência artificial na medicina. *International Journal of Social Science and Human Research*, *2*(11), 937–941. https://doi.org/10.47191/IJSSHR/V3-I11-05

Hadavimoghaddam, F., Harandi, V. S., Mostajeran, M., & Zabihi, R. (2023). Application of data mining in gas injection methods. *Gas Injection Methods: A Volume in Enhanced Oil Recovery Series*, 359–380. https://doi.org/10.1016/B978-0-12-822302-4.00012-0

IBM. (2023). *What is Machine Learning?* Retrieved December 1, 2023, from https://www.ibm.com/topics/machine-learning

Khaleel, M., Ahmed, A. A., & Alsharif, A. (2023). Artificial Intelligence in Engineering. *Brilliance: Research of Artificial Intelligence*, *3*(1), 32–42. https://doi.org/10.47709/BRILLIANCE.V3I1.2170

Kuchuk, J. (2009). Radius of Investigation for Reserve Estimation from Pressure Transient Well Tests. SPE Middle East Oil and Gas Show and Conference. doi: https://doi.org/10.2118/120515-MS

Lashari, S.-e.-Z. (2018). Application of Artificial Intelligence (AI) in Petroleum Engineering Problems. West Virginia University, Master's Thesis.

Lobut, B., & Artun, E. (2023). Machine-Learning Based Selection of Candidate Wells for Extended Shut-In Due to Fluctuating Oil Prices. *Society of Petroleum Engineers - SPE EuropEC - Europe Energy Conference Featured at the 84th EAGE Annual Conference and Exhibition, EURO 2023*. https://doi.org/10.2118/214353-MS

Martí, L., Sanchez-Pi, N., Molina, J. M., & Garcia, A. C. B. (2015). Anomaly detection based on sensor data in petroleum industry applications. *Sensors (Switzerland)*, *15*(2), 2774–2797. https://doi.org/10.3390/S150202774

Mohaghegh, S. D. (n.d.). Data-driven reservoir modeling: top-down modeling (TDM): a paradigm shift in reservoir modeling, the art and science of building reservoir models based on field measurements. Society of Petroleum Engineers.

Noble, R., & Noble, D. (2023). Artificial Intelligence. In *Understanding Living Systems* (Understanding Life, pp. 99-112). Cambridge: Cambridge University Press. https://doi.org/10.1017/9781009277396.009

Pennsylvania State University. (2019, April 22). *Theory meets application: Machine learning techniques for geothermal exploration*. PHYS. https://phys.org/news/2019-04-theory-application-machine-techniques-geothermal.html

Qinghua, Huang. (2022). Artificial Intelligence and Systems Thinking in the Public Sector. https://doi.org/10.4018/978-1-6684-5624-8.ch002

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323, (533-536).

Yadav, S., Yadav, S., Srivastava, S. P., Gupta, S. K., & Mishra, S. (2022). Machine Learning. *Deep Learning for Targeted Treatments*, 407–430. https://doi.org/10.1002/9781119857983.CH13

APPENDICES

A. Code Algorithm

```
# Importing libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GaussianNoise
from tensorboard.plugins.hparams import api as hp
import warnings
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation, FFMpegWriter
import pandas as pd
import numpy as np
from statistics import mean
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
r2_score, mean_absolute_error, max_error
from sklearn import preprocessing
import joblib
from time import sleep
from itertools import zip_longest
import multiprocessing as mp
import time as systime
import datetime
import gc
gc.enable()
warnings.simplefilter('ignore', np.RankWarning)

# Confirm hardware integration
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("Num CPUs Available: ", len(tf.config.list_physical_devices('CPU')))

# Importing data
data = pd.read_excel("test_7_withanomaly.xlsx", sheet_name=0, skiprows=4)
time = pd.DataFrame(data.iloc[:,0])
pressure = pd.DataFrame(data.iloc[:,5])
water = pd.DataFrame(data.iloc[:,4])
gas = pd.DataFrame(data.iloc[:,2])
oil = pd.DataFrame(data.iloc[:,3])
```

```
# A function to calculate the difference/change in data
def delta_fun(data, initial):
    datay = data.copy().reset_index(drop=True)
    datay.loc[:] = datay.loc[:] - initial.values
    return datay


# A function for plotting
def plot_series(time, series, format="-", start=0,
                label=None):  # needs editing so i can add other graphs within same
function
    plt.plot(time, series, format, label=label)
    plt.xlabel("Time")
    plt.legend(fontsize=14)
    plt.grid(True)
    plt.show()

def pressure_plot(time, series, label):
    forecast = series[0].values[:,0]
    fig, ax = plt.subplots()
    for i in reversed(range(len(series))):
        ax.plot(time, series[i].values, label=label[i])
    error = 0.003
    P_err = forecast*error
    ax.fill_between(time[:,0], forecast+P_err, forecast-P_err, alpha=0.2,
color='green', label="Model error {}%".format(error*100))
    ax.set_xlabel("Time")
    ax.set_ylabel("Pressure")
    ax.legend(fontsize=10,loc='lower left')
    ax.grid(True)
    plt.show()

# A function to reverse the difference in pressure
def full_fun(data, initial):
    datax = data.copy().reset_index(drop=True)
    datax.loc[:] = datax.loc[:] + initial.values
    return datax

# Add Noise to data
def noisy(data,time,error):
    datax = data.copy()
    noise = np.random.normal(datax, (datax*error/3), datax.shape)
```

```
    noise = pd.DataFrame(noise, columns=datax.columns)
    for i in range(datax.shape[1]):
        y=datax.iloc[:, i]
        plt.scatter(time, noise.iloc[:, i], label= 'Data with Noise')
        plt.plot(time, y, label= 'True Data', color='r')
        P_err = y*error
        plt.fill_between(time.iloc[:,0], y+P_err, y-P_err, alpha=0.2, color='green',
label="Error range {}%".format(error*100))
        plt.xlabel("Time")
        plt.ylabel(datax.columns[i])
        plt.legend(fontsize=12)
        plt.grid(True)
        plt.show()
    return noise

# Preprocessing and Splitting data into train, validation, and test
X = data.drop(data.columns[[1, 5]], axis=1)
X = noisy(data= X.iloc[:, 1:], time=time, error=0.0003)
X.insert(0, time.columns[0], time.values)
Y = pressure
Y = noisy(Y, time=time, error=0.0003)
Yi = Y.take([0])
Xi = pd.DataFrame([[0,0,0,0]],columns=X.columns)

x_train0, X_test1, y_train0, Y_test1 = train_test_split(X, Y, test_size=0.99,
shuffle=False)
X_test0 , Y_test0 = X_test1 , Y_test1
X_test1, X_test5, Y_test1, Y_test5 = train_test_split(X_test1, Y_test1,
test_size=500, shuffle=False)
X_test1, X_test4, Y_test1, Y_test4 = train_test_split(X_test1, Y_test1,
test_size=500, shuffle=False)
X_test1, X_test3, Y_test1, Y_test3 = train_test_split(X_test1, Y_test1,
test_size=500, shuffle=False)
X_test1, X_test2, Y_test1, Y_test2 = train_test_split(X_test1, Y_test1,
test_size=500, shuffle=False)
x_train, x_val, y_train, y_val = train_test_split(x_train0, y_train0, test_size=0.2,
shuffle=True)
xlastinput = ylastinput = xlastinput_ab = ylastinput_ab = pd.DataFrame() #reset
last inputs

x_train = delta_fun(x_train, Xi)
y_train = delta_fun(y_train, Yi)
x_test1  = delta_fun(X_test1, Xi)
y_test1  = delta_fun(Y_test1, Yi)
```

```
x_test2  = delta_fun(X_test2, Xi)
y_test2  = delta_fun(Y_test2, Yi)
x_test3  = delta_fun(X_test3, Xi)
y_test3  = delta_fun(Y_test3, Yi)
x_test4  = delta_fun(X_test4, Xi)
y_test4  = delta_fun(Y_test4, Yi)
x_test5  = delta_fun(X_test5, Xi)
y_test5  = delta_fun(Y_test5, Yi)
x_test0  = delta_fun(X_test0, Xi)
y_test0  = delta_fun(Y_test0, Yi)
x_diff = delta_fun(X, Xi)
y_diff = delta_fun(Y, Yi)


# testing short
x_t = delta_fun(X.head(1400), Xi)
y_t = Y.head(1400)
x_t , x_tt , y_t, Y_tt = train_test_split(x_t, y_t, test_size=0.5, shuffle=False)
y_t = delta_fun(y_t, Yi)

#%% ----------------------3--------------------------------
#This step done only once, Scaler fitted for preprocessing
XScaler = preprocessing.MinMaxScaler().fit(x_diff) # (X-X.min())/(X.min()-
X.max())
YScaler = preprocessing.MinMaxScaler().fit(y_diff)
joblib.dump(XScaler, 'xtest6_multiwell.gz')
joblib.dump(YScaler, 'ytest6_multiwell.gz')

# Normalizing data
XScaler = joblib.load('xtest6_multiwell.gz')
YScaler = joblib.load('ytest6_multiwell.gz')
x_train0 = XScaler.transform(x_train0)
y_train0 = YScaler.transform(y_train0)
x_train = XScaler.transform(x_train)
y_train = YScaler.transform(y_train)
x_val = XScaler.transform(x_val)
y_val = YScaler.transform(y_val)
x_test1 = XScaler.transform(x_test1)
y_test1 = YScaler.transform(y_test1)
x_test2 = XScaler.transform(x_test2)
y_test2 = YScaler.transform(y_test2)
x_test3 = XScaler.transform(x_test3)
y_test3 = YScaler.transform(y_test3)
x_test4 = XScaler.transform(x_test4)
```

```
y_test4 = YScaler.transform(y_test4)
x_test5 = XScaler.transform(x_test5)
y_test5 = YScaler.transform(y_test5)
x_test0 = XScaler.transform(x_test0)
y_test0 = YScaler.transform(y_test0)
x_normalized = XScaler.transform(x_diff)
y_normalized = YScaler.transform(y_diff)

# Model structure build
tf.keras.backend.clear_session()
tf.random.set_seed(42) # Ensuring we get the same output with multiple runs
np.random.seed(42)

model = Sequential()
model.add(Dense(units=60, activation='relu', input_shape=(4,)))
model.add(GaussianNoise(0.01))
# model.add(Dropout(0.1))
model.add(Dense(units=60, activation='relu', kernel_regularizer='l2'))#,
kernel_regularizer='l2'
model.add(Dense(units=1))
model.summary()

#%% Model Compile with increasing Learning rate to test suitable lr to use

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-7)

lr_schedule = tf.keras.callbacks.LearningRateScheduler(

   lambda epoch: 1e-7 * 10 ** (epoch / 100))

model.compile(optimizer=optimizer,  # default='rmsprop', an algorithm to be used
in backpropagation

            loss='mse',

            # Loss function to be optimized. A string (name of loss function), or a
tf.keras.losses.Loss instance.

            metrics=['mse'],

            # List of metrics to be evaluated by the model during training and testing.
Each of this can be a

            # string (name of a built-in function), function or a tf.keras.metrics.Metric
instance.

            )

# History fit
```

```
history = model.fit(x_train, y_train, batch_size=10, epochs=500,
validation_data=(x_val,y_val),callbacks=[lr_schedule])


# Plot loss vs learning rate
plt.semilogx(history.history["lr"], history.history["loss"])
plt.semilogx(history.history["lr"], history.history['val_loss'], label='validation loss')
plt.xlabel('Learning rate')
plt.ylabel('loss')
plt.show()


# Fitting model with adam optimizer
batch_size = 5
epochs=50
validation_split=0.2
initial_learning_rate=1e-3
decay_steps=10000
decay_rate=0.96
decayed_lr = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate,
                                                            decay_steps,
                                                            decay_rate,
                                                            staircase=True)


optimizer = tf.keras.optimizers.Adam(decayed_lr)
model.compile(optimizer,
              loss='mse'
              )
#es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',
min_delta=1e-4, verbose=1, patience=30)
#tqdm_callback = tfa.callbacks.TQDMProgressBar()
```

```
# log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-
%H%M%S")
# tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)


# hparams_dir = os.path.join(log_dir, 'validation')
# with tf.summary.create_file_writer(hparams_dir).as_default():
#     hp.hparams_config(
#         hparams=HPARAMS,
#         metrics=[hp.Metric('epoch_accuracy')]  # metric saved by tensorboard_cb
#     )


# hparams_cb = hp.KerasCallback(
#     writer=hparams_dir,
#     hparams=HPARAMS
# )
xx1=tf.convert_to_tensor(x_train)
yy1=tf.convert_to_tensor(y_train)
history = model.fit(xx1, yy1,
                    batch_size,
                    epochs,
                    verbose=1,
                    # callbacks= [tensorboard_callback], # ,hp.KerasCallback(log_dir,
hparams)
                    validation_data=(x_val,y_val)
                    )


plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
```

```
plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend(fontsize=14)

plt.show()


#Saving the model

model.save('trainedmodel_multi_v13-9-well2_lowproduction_short.h5')

# Automated forecasting every day
# Daily data input prediction
# @profile # this code-line is called "Decorator" and used for debugging
def daily_forecast(production_expected, field_pressure, saved_model):
    # User defined variables
    iterate = 30 # forecast for 30 days forward only.
    error = 0.003 # is a changing variable based on forecast accuracy degradation
    err_factor = 1 # polynomyal degree for regression
    learning_rate=3e-7 # initial lr for daily re-training on new data LR(3e-7)

    # Import ---------------------------------------------------------------
    Range = len(field_pressure)
    actual = field_pressure.values[:,0]
    new_model = saved_model
    time = production_expected.values[:,0]
    x_all = XScaler.transform(delta_fun(production_expected, Xi))
    y_all = YScaler.transform(delta_fun(field_pressure, Yi))
    truedata_df = production_expected.copy()
    truedata_df['field_pressure']= actual
    truedata_df.to_csv('animation/csv_export/truedata_df.csv')
    # initialize empty variables
    forecast_all = np.empty(shape=(0,1))
    P_err = P_err1 = P_err7 = P_err30 = np.empty(shape=(0,1))
    day1, day7, day30 = np.empty(shape=(3,0))
    f1, f7, f30 = np.empty(shape=(3,0))
    anomaly = np.empty(shape=(0,1))
    counter = 0
    # Initial predictions based on model trained ----------------------------
    # forecast_old_all = new_model.predict(x_all)
    # forecast_old_all =
pd.DataFrame(YScaler.inverse_transform(forecast_old_all),columns=Yi.columns)
    # forecast_old_all = full_fun(forecast_old_all, Yi)
```

```
    # looping over daily data ----------------------------------------------------
    for i in range(Range-iterate):
        t_start = systime.time()
        # predicting the input rates
        if i<iterate:
            g = np.poly1d(np.polyfit(time[0:i+1], production_expected.values[0:i+1,1],
1))
            o = np.poly1d(np.polyfit(time[0:i+1], production_expected.values[0:i+1,2],
1))
            w = np.poly1d(np.polyfit(time[0:i+1], production_expected.values[0:i+1,3],
1))
        else:
            g = np.poly1d(np.polyfit(time[i-iterate:i], production_expected.values[i-
iterate:i,1], 1))
            o = np.poly1d(np.polyfit(time[i-iterate:i], production_expected.values[i-
iterate:i,2], 1))
            w = np.poly1d(np.polyfit(time[i-iterate:i], production_expected.values[i-
iterate:i,3], 1))
        oo, ww, gg = o(time[0:i+iterate]), w(time[0:i+iterate]), g(time[0:i+iterate])
        x_all_mod_df = pd.DataFrame(np.column_stack((time[0:i+iterate], gg, oo,
ww)), columns= production_expected.columns)
        x_all_mod = XScaler.transform(delta_fun(x_all_mod_df, Xi))
        forecast = new_model(x_all_mod[i:i+iterate,:])
        forecast =
pd.DataFrame(YScaler.inverse_transform(forecast),columns=Yi.columns)
        forecast = full_fun(forecast, Yi)
        forecast = forecast.values[:,0]
        forecast_all = np.append(forecast_all[:i], forecast)
        f1 = np.append(f1, forecast[0])
        f7 = np.append(f7, forecast[6])
        f30 = np.append(f30, forecast[-1])

        err1 = err(f1[i], actual[i])
        day1 = np.append(day1, err1)
        if i>=6:
            err7 = err(f7[i-6], actual[i])
            day7 = np.append(day7, err7)
        if i>=iterate-1:
            err30 = err(f30[i-iterate+1], actual[i])
            day30 = np.append(day30, err30)
        if i<iterate-1:
            if sum(day7)==0.0: day7 = np.zeros(shape=(i+1,))
            if sum(day30)==0.0: day30 = np.zeros(shape=(i+1,))
```

```
        if i <iterate-1:
            P_err = np.append(P_err[:i], error*forecast)
            P_err1 = P_err7 = P_err30 = P_err

        zp = np.linspace(time[0], time[i+iterate-1], i+iterate)

        if i >= iterate-1:
            z1 = np.poly1d(np.polyfit(zp[i-29:i+1], day1[-30:], err_factor))
            z7 = np.poly1d(np.polyfit(zp[i-29:i+1], day7[-30:], err_factor))
            z30 = np.poly1d(np.polyfit(zp[i-29:i+1], day30[-30:], err_factor))
            Ac = max_error(day1[-30:], z1(zp[i-29:i+1]))
            Bc = max_error(day30[-30:], z30(zp[i-29:i+1]))

            P_err1 = np.append(P_err1[:i], (z1(zp)[-
iterate+1]+np.sign(z1[0])*Ac)/100*f1[-1]) #appending the next expected error of
next day forecast
            P_err7 = np.append(P_err7[:i], z7(zp)[-iterate+1]/100*f7[-1])
            P_err30 = np.append(P_err30[:i], (z30(zp)[-
iterate+1]+np.sign(z1[0])*Bc)/100*f30[-1])
            errors_all_df = pd.DataFrame(np.column_stack((zp,
                                    z1(zp),
                                    z7(zp),
                                    z30(zp)
                                    )), columns=['zp','z1','z7','z30'])
            P_errors = pd.DataFrame(np.column_stack((P_err1,P_err7,P_err30)),
columns=['P_err1','P_err7','P_err30'])
            errors_all_df.to_csv('animation/csv_export/errors_all_df.csv')
            P_errors.to_csv('animation/csv_export/P_errors.csv')

            #Finding anomaly at the exact location when happening

            anomaly = np.append(anomaly, (abs(err30) > abs(P_err30[-1]) or
                                    abs(err1) > abs(P_err1[-1]) or
                                    abs(P_err1[-1]) > abs(P_err30[-1])
                                    )) #  and A!=B and AA!=BB
        else: anomaly = np.append(anomaly, False)


        # saving data to excel ----------------------------------------------
        x_all_mod_df['forecast_all']=forecast_all
        x_all_mod_df.to_csv('animation/csv_export/x_all_mod_df.csv')

        alldays_df = pd.DataFrame(np.column_stack((actual[0:i+1],
                                f1, day1,
```

```
                                f7, day7,
                                f30, day30,
                                anomaly)),
                            index=time[0:i+1],
                            columns=['actual',
                                     'day1 pred','err1',
                                     'day7 pred','err7',
                                     'day30 pred','err30',
                                     'anomaly']
                            )
        alldays_df.to_csv('animation/csv_export/alldays_df.csv')

        # setting the Learning rate in case the error is too big/small
        if abs(day1[-1])>3.0 or sum(abs(day1[-7:])>0.1)==17:
            learning_rate=1e-5
            print("high lr {}".format(learning_rate))
        elif (abs(day1[-1])<3.0 and abs(day1[-1])>0.1):
            learning_rate=1e-6
        # Daily training
        if i<iterate-1 or abs(day1[-1])>0.1:
            modeltrain(x_all, y_all, i, learning_rate, new_model)
            # Housekeeping: Trying to optimize excution time by cleaning old data
stored in memory
            del forecast, zp, g,gg, o,oo, w,ww, x_all_mod_df,x_all_mod, alldays_df,
new_model # ,errors_all_df,P_errors,z1,z7,P_err
            tf.keras.backend.clear_session()
            # gc.collect()
            new_model =
tf.keras.models.load_model('DNNmodels/trainedmodel_multi_v13-
9_lowproduction_short_cont2.keras')

        counter += 1
        t_end = systime.time()
        print('{} iteration, {:.2f}s'.format(i,t_end-t_start))

# @tf.function(experimental_relax_shapes=True) only works if i made my own
manual epoch loop
# @profile # this code-line is for debugging
# @tf.function(reduce_retracing=True)
def modeltrain(x, y, i, lr, new_model):
    x = x[i,:].reshape(1,4)
    y = y[i,:].reshape(1,1)
    x=tf.convert_to_tensor(x)
    y=tf.convert_to_tensor(y)
```

```
    new_model.compile(optimizer=tf.keras.optimizers.Adam(lr), loss='mse')
    new_model.fit(x, y,
                  batch_size=1,
                  epochs=10,
                  verbose=0
                  )
    new_model.save('DNNmodels/trainedmodel_multi_v13-
9_lowproduction_short_cont2.keras')

def err(forecast, actual):
    Error_increase = (actual-forecast)/actual*100
    return Error_increase

def lr(lr):
    decayed_lr =
tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=lr,
                                                   decay_steps=10000,
                                                   decay_rate=0.95,
                                                   staircase=True)
    optimizer = tf.keras.optimizers.Adam(decayed_lr)
    return optimizer

#%%------------------------12-----------------------------
daily_forecast(X_test0, Y_test0, saved_model=
tf.keras.models.load_model('DNNmodels/trainedmodel_multi_v13-
9_lowproduction_short.h5'))
```

B. Libraries require installation.

Package	Version	Package	Version
absl-py	1.4.0	pathspec	0.10.3
alabaster	0.7.12	pcre	8.45
arrow	1.2.3	pexpect	4.8.0
astroid	2.14.2	pickleshare	0.7.5
asttokens	2.0.5	pillow	9.4.0
astunparse	1.6.3	pip	23.0.1
atomicwrites	1.4.0	platformdirs	2.5.2
attrs	22.1.0	pluggy	1.0.0
autopep8	1.6.0	ply	3.11
babel	2.11.0	poyo	0.5.0
backcall	0.2.0	prompt-toolkit	3.0.36
bcrypt	3.2.0	protobuf	3.19.6
beautifulsoup4	4.11.1	psutil	5.9.0
binaryornot	0.4.4	ptyprocess	0.7.0
black	22.6.0	pure_eval	0.2.2
bleach	4.1.0	pyasn1	0.4.8
brotlipy	0.7.0	pyasn1-modules	0.2.8
ca-certificates	2023.08.22	pycodestyle	2.10.0
cachetools	5.3.0	pycparser	2.21
certifi	2023.7.22	pydocstyle	6.3.0
cffi	1.15.1	pyflakes	3.0.1
chardet	4.0.0	pygments	2.11.2
charset-normalizer	3.1.0	pylint	2.16.2
click	8.0.4	pylint-venv	2.3.0
cloudpickle	2.0.0	pyls-spyder	0.4.0
colorama	0.4.6	pynacl	1.5.0
columnar	1.4.1	pyopenssl	23.0.0
comm	0.1.2	pyparsing	3.0.9
contourpy	1.0.7	pyqt	5.15.7
cookiecutter	1.7.3	pyqt5-sip	12.11.0
cryptography	39.0.1	pyqtwebengine	5.15.7
cudatoolkit	11.2.2	pyrsistent	0.18.0
cudnn	8.1.0.77	pysocks	1.7.1
cycler	0.11.0	python	3.9.16
debugpy	1.5.1	python-dateutil	2.8.2

Package	Version	Package	Version
decorator	5.1.1	python-fastjsonschema	2.16.2
defusedxml	0.7.1	python-lsp-black	1.2.1
diff-match-patch	20200713	python-lsp-jsonrpc	1.0.0
dill	0.3.6	python-lsp-server	1.7.1
docstring-to-markdown	0.11	python-slugify	5.0.2
docutils	0.18.1	pytoolconfig	1.2.5
entrypoints	0.4	pytz	2022.7
et-xmlfile	1.1.0	pywin32	305
executing	0.8.3	pywin32-ctypes	0.2.0
ffmpeg	4.2.2	pyyaml	6
flake8	6.0.0	pyzmq	23.2.0
flatbuffers	23.3.3	qdarkstyle	3.0.2
flit-core	3.6.0	qstylizer	0.2.2
fonttools	4.39.0	qt-main	5.15.2
gast	0.4.0	qt-webengine	5.15.9
giflib	5.2.1	qtawesome	1.2.2
glib	2.69.1	qtconsole	5.4.0
google-auth	2.16.2	qtpy	2.2.0
google-auth-oauthlib	0.4.6	qtwebkit	5.212
google-pasta	0.2.0	requests	2.28.2
grpcio	1.51.3	requests-oauthlib	1.3.1
gst-plugins-base	1.18.5	rope	1.7.0
gstreamer	1.18.5	rsa	4.9
h5py	3.8.0	rtree	1.0.1
icu	58.2	scikit-learn	1.2.1
idna	3.4	scipy	1.10.1
imagesize	1.4.1	setuptools	65.6.3
importlib-metadata	6.0.0	sip	6.6.2
importlib-resources	5.12.0	six	1.16.0
importlib_metadata	4.11.3	snowballstemmer	2.2.0
inflection	0.5.1	sortedcontainers	2.4.0
intervaltree	3.1.0	soupsieve	2.3.2.
ipykernel	6.19.2	sphinx	5.0.2
ipython	8.10.0	sphinxcontrib-applehelp	1.0.2
ipython_genutils	0.2.0	sphinxcontrib-devhelp	1.0.2
isort	5.9.3	sphinxcontrib-htmlhelp	2.0.0
jedi	0.18.1	sphinxcontrib-jsmath	1.0.1
jellyfish	0.9.0	sphinxcontrib-qthelp	1.0.3
jinja2	3.1.2	sphinxcontrib-serializinghtml	1.1.5

Package	Version	Package	Version
jinja2-time	0.2.0	spyder	5.4.1
joblib	1.2.0	spyder-kernels	2.4.2
jpeg	9e	sqlite	3.40.1
jsonschema	4.17.3	stack_data	0.2.0
jupyter_client	7.4.9	string-color	1.2.3
jupyter_core	5.2.0	tensorboard	2.10.1
jupyterlab_pygments	0.1.2	tensorboard-data-server	0.6.1
keras	2.10.0	tensorboard-plugin-wit	1.8.1
keras-preprocessing	1.1.2	tensorflow	2.10.1
keyring	23.4.0	tensorflow-addons	0.21.0
kiwisolver	1.4.4	tensorflow-estimator	2.10.0
lazy-object-proxy	1.6.0	tensorflow-intel	2.11.0
lerc	3	tensorflow-io-gcs-filesys	0.31.0
libclang	15.0.6.1	termcolor	2.2.0
libdeflate	1.17	text-unidecode	1.3
libffi	3.4.2	textdistance	4.2.1
libiconv	1.16	threadpoolctl	3.1.0
libogg	1.3.5	three-merge	0.1.1
libpng	1.6.39	time-profiler	0.0.2
libsodium	1.0.18	tinycss2	1.2.1
libspatialindex	1.9.3	toml	0.10.2
libtiff	4.5.0	tomli	2.0.1
libvorbis	1.3.7	tomlkit	0.11.1
libwebp	1.2.4	toolz	0.12.0
libwebp-base	1.2.4	tornado	6.2
libxml2	2.9.14	tqdm	4.65.0
libxslt	1.1.35	traitlets	5.7.1
line_profiler	4.1.1	typeguard	2.13.3
lxml	4.9.1	typing-extensions	4.5.0
lz4-c	1.9.4	typing_extensions	4.4.0
markdown	3.4.1	tzdata	2022g
markupsafe	2.1.2	ujson	5.4.0
matplotlib	3.7.1	unidecode	1.2.0
matplotlib-inline	0.1.6	urllib3	1.26.14
mccabe	0.7.0	vc	14.2
memory_profiler	0.58.0	vs2015_runtime	14.27.29016
mistune	0.8.4	watchdog	2.1.6
mypy_extensions	0.4.3	wcwidth	0.2.5
nbclient	0.5.13	webencodings	0.5.1

Package	Version	Package	Version
nbconvert	6.5.4	werkzeug	2.2.3
nbformat	5.7.0	whatthepatch	1.0.2
nest-asyncio	1.5.6	wheel	0.38.4
numpy	1.24.2	win_inet_pton	1.1.0
numpydoc	1.5.0	wincertstore	0.2
oauthlib	3.2.2	wrapt	1.15.0
openpyxl	3.1.1	xz	5.2.10
openssl	1.1.1w	yaml	0.2.5
opt-einsum	3.3.0	yapf	0.31.0
packaging	23	zeromq	4.3.4
pandas	1.5.3	zipp	3.15.0
pandocfilters	1.5.0	zlib	1.2.13
paramiko	2.8.1	zstd	1.5.2
parso	0.8.3		

C. CMG simulation run summary.

Field Total	Fluid		
	Oil	Gas	Water
	-------	-------	-------
	(MSM3)	(MMSM3)	(MSM3)
Cumulative Production	4210	1051.1	158.69
Cumulative Injection	NA	0	510.82
Cumulative Gas Lift	NA	0	NA
Cumulative Aquifer Influx	NA	NA	5390.4
Current Fluids In Place	7745	1055.7	12189
Production Rates	0.9	0.13866	0.41385
Injection Rates	NA	0	0.30047

Timesteps: 5016 Newton Cycles: 5024 Cuts: 0 Solver Iterations: 37340
Average Implicitness : 0.029
Fluid Component Model : BLACKOIL (SINGLE-P)
Material Balances (owg): 1.000 1.000 1.000
Average Active Blocks: 783 Average Non-BHP Active Wells: 4
Total Blocks : 1092 Total Wells : 5
Active Blocks: 783 Non-BHP Active Wells: 5
Time at end of simulation: 5000.00 (days)
Average reservoir pressure excluding water zone: 25098.56 (kPa)
Total Number of Solver Failures: 0 Stalls: 0 ITERMAX Reached: 0
Jacobian Domains 1
Total lstpro/lstpar calls: 2 (3)
Linear Solver: Aimsol
Preconditioner Ordering: REDBLACK
Preconditioner Degree 1
KMP_AFFINITY: Default
OMP_SCHEDULE: Default
Max Impl Blocks: 38 %Impl: 4.9% (TS,CUT,NCYC): (4929, 0, 1)
Max Solver Iterations (TS,CUT,NCYC): 11 (1, 0, 1)
Number of threads set: 1
Total number of cpus: 12
Memory Usage Peak: 72 MB on TS: 3729 TS 1 Peak: 42 MB Average: 68 MB VM Size: 79 MB
Memory Usage Final Size: 73 MB
Host computer: MYDUCK
CPU Time: 12.00 seconds
Elapsed Time: 51.73 seconds
End of Simulation: Normal Termination