Machine Learning Based Algorithms for Improving Forecasting in Subsurface Energy Resources

by
Omar S. Alolayan
B.S., Computer Engineering, King Fahd University of Petroleum & Minerals, 2013
M.S., Computational Science, Mathematics & Engineering, UC San Diego, 2018

Submitted to the Department of Civil and Environmental Engineering
and the Center for Computational Science and Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2023

Authored by: Omar S. Alolayan
Department of Civil and Environmental Engineering and the Center for Computational Science and Engineering
April 25, 2023

Certified by: John R. Williams
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by: Colette L. Heald
Chair, Graduate Program Committee

Accepted by: Nicolas Hadjiconstantinou
Co-Director, Center for Computational Science and Engineering

Machine Learning Based Algorithms for Improving Forecasting in Subsurface Energy Resources

by

Omar S. Alolayan

Submitted to the Department of Civil and Environmental Engineering and the
Center for Computational Science and Engineering
on April 25, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Energy is an essential human need that is necessary for maintaining and improving quality of life. It also plays a crucial role in sustaining and developing the world economy. Accurate production forecasting models are important for governments, organizations, and companies as they enable them to make informed decisions and develop returns on investments. However, forecasting models for subsurface energy resources still face several challenges, such as mathematically ill-posed problems and lack of reliable data. Machine learning offers new methods to develop better forecasting models due its capacity to learn desired behavior from interacting with an environment of interest, as well as its ability to create optimal non-linear mappings between input and output data.

In this research work, we reformulate the history matching problem from a least-square mathematical optimization problem into a Markov Decision Process to develop a method in which reinforcement learning can be utilized to solve the problem. This method provides a mechanism where an artificial deep neural network agent can interact with the reservoir simulator and find multiple different solutions to the problem. Such formulation allows for solving the problem in parallel by launching multiple concurrent environments enabling the agent to learn simultaneously from all the environments at once, achieving significant speed up.

Additionally, we use deep neural networks to generate more accurate shale gas production forecasts in counties with a limited number of sample wells by utilizing transfer learning. By using transfer learning, we provide a way of transferring the knowledge gained from other deep neural network models trained on adjacent counties into the county of interest. This research project uses data from more than 6000 shale gas wells across 17 counties from Texas Barnett and Pennsylvania Marcellus shale formations to test the capabilities of transfer learning. The results reduce the forecasting error between 11% and 47% compared to the widely used Arps decline curve model.

Thesis Supervisor: John R. Williams
Title: Professor of Civil and Environmental Engineering

Acknowledgments

بِسْمِ اللهِ الرَحمٰنِ الرَحيْمِ

(The Arabic texts in this acknowledgement are prayers and poems for thanking God and my family)

When I started my PhD journey at MIT, I did not think it would be this challenging. Those challenges were even more difficult due to the fact that I had to face them during a global pandemic. I am lucky that I had tremendous help and support from my advisor, family and friends to help me complete this journey.

(وَإِذا كانَتِ النُفوسُ كِباراً تَعِبَت في مُرادِها الأَجسامُ)

I would like to start by thanking God, the most gracious, the most merciful, for the blessings bestowed upon me and providing me with this the opportunity along with the tools necessary to achieve my dream.

(رَبِّ أَوْزِعْنِي أَنْ أَشْكُرَ نِعْمَتَكَ الَّتِي أَنْعَمْتَ عَلَيَّ وَعَلَى وَالِدَيَّ)

I owe a debt of gratitude to my parents Suliman Alolayan (سليمان العليّان) and Fatimah Alsaadi (فاطمة السعدي) for believing in me and providing my with all the help and support needed to succeed. My parents have always believed in the importance of education and invested time, effort and money in my education from a young age. By earning my PhD from MIT, I hope to bring them a sense of pride and accomplishment. They have inspired me to be a better parent for my kids and I hope to be as dedicated and supportive of my children as they were for me.

(رَّبِّ ارْحَمْهُمَا كَمَا رَبَّيَانِي صَغِيرًا)

I would also like to thank my wife Basmah Alhudhayf (باسمة الحضيف) who closely supported me during this journey and endured all the challenges and difficulties that came with it. She sacrificed a lot especially during the difficult times of the Covid-19 pandemic. I am grateful to her as it would have been extremely difficult to pursue my PhD if it were not for her support.

My advisor, Professor John R. Williams, deserves special recognition for his guidance, patience, and support throughout my PhD journey. He provided me with all the tools needed to succeed at MIT, trusted me, and gave me the confidence to pursue difficult research topics. During the difficult times, he was patient and did everything he could to ensure my academic success. I am grateful to him for being available at all times to provide me with both academic and non-academic support.

I am also grateful to my father in law Abdullah Alhudhayf (عبدالله الحضيف) and my mother in law Reem Almohsin (ريم المحسن) for their tremendous support for both me and Basmah during this journey. They helped us during one the most difficult times when our son Tammam (تمّام) was born as they traveled all the way from Albukariyah, Saudi Arabia to Boston to provide us with all the help we needed. I am grateful for them as I always knew that whenever things got difficult I can count for their support.

I would also like to thank my siblings Khowlah, Khalid, Hajar, Asma and Aminah along with my nieces (Leen, Rawh, Mohrah, Deemah, Dorrah and Elaf) and nephews (Rayyan, Tameem, Ebraheem, Suliman). Living away from home was difficult and I am grateful for their support during that time.

I would like to express my deep appreciation to Dr. Abdullah Almaatouq, Dr. Mohammed Alhassoun, Abdullah Alomar, Mohammad Alsobay, Dr. Fahad Alhasoun, Dr. Tariq Alshaalan, Dr. Ahmad Alyoubi, and Dr. Nabeel Alzamil for their invaluable guidance during my thesis. Their technical and academic expertise provided me with the necessary knowledge to develop the algorithms that were crucial to generating the results and advancing the findings of my research.

I would like to express my sincere appreciation to my doctoral thesis committee, particularly Professor Ruben Juanes for serving as my committee chair. Under the guidance of Professors Juanes and Professor Herbert Einstein, I was able to develop and refine my research ideas into an algorithm that can be used by the industry to solve the history matching problem in a timely manner.

I would also like to express my gratitude to the Saudi Arabian Oil Company (Aramco) for their invaluable support during my graduate studies. Thanks to their

generous graduate fellowship, I have been able to pursue my PhD and further enhance my academic development. Aramco's investment in my career has been instrumental in helping me achieve my goals, and I will always be deeply grateful for their generosity and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, energy is one of the most important basic needs that is necessary for social progress and improving quality of life (1, 2). It is a cornerstone for sustaining and developing the world economy (3). Oil and gas, which are subsurface energy resources, constitute a significant proportion of the global energy production. As shown in Figure 1-1, these resources account for over 50% of the world's energy supply (4, 5).

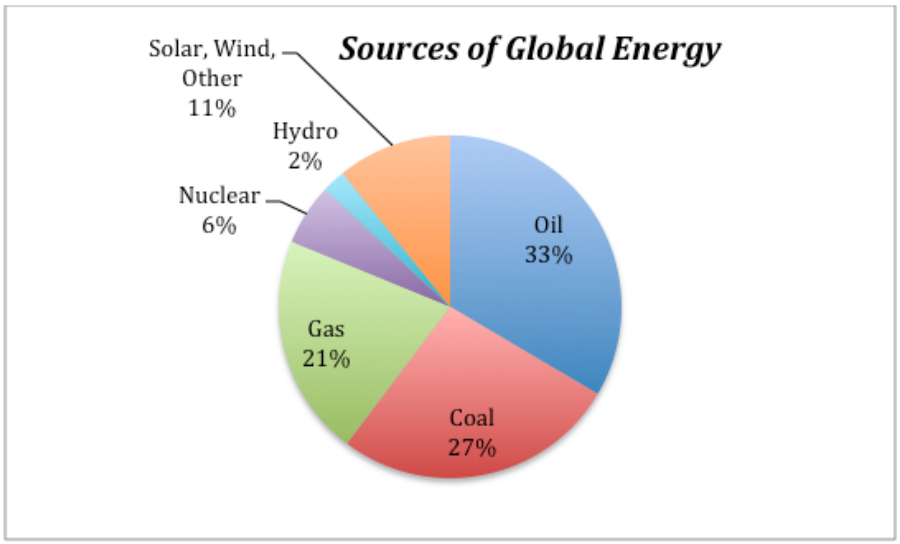

Figure 1-1: A chart showing the sources of global energy supply where oil and gas generate 54% of the world energy (4, 5).

Given the crucial role that these energy resources play in supporting societal

progress and economic development, it is essential to have accurate production forecasting models. Such models are necessary for government policy makers as they provide an insight that enables them to come up with informed decisions (6). It is also important for oil and gas companies to maintain and manage the wells as well as optimize returns on investments.

However, there are challenges and difficulties related to developing accurate forecasting models. For this reason, a lot of research has been done to overcome or mitigate such difficulties. Some of these challenges arise from the fact that the problem itself is mathematically ill-posed. In other cases, the models developed to forecast the production lack enough data to generate a reliable forecast.

In recent years, machine learning has gained lots of popularity as a tool to tackle research problems. Machine learning uses data to build a predictive model that can map input data to a useful output. In the literature, as shown in Figure 1-2 , machine learning is usually classified into three types: supervised learning, unsupervised learning and reinforcement learning.

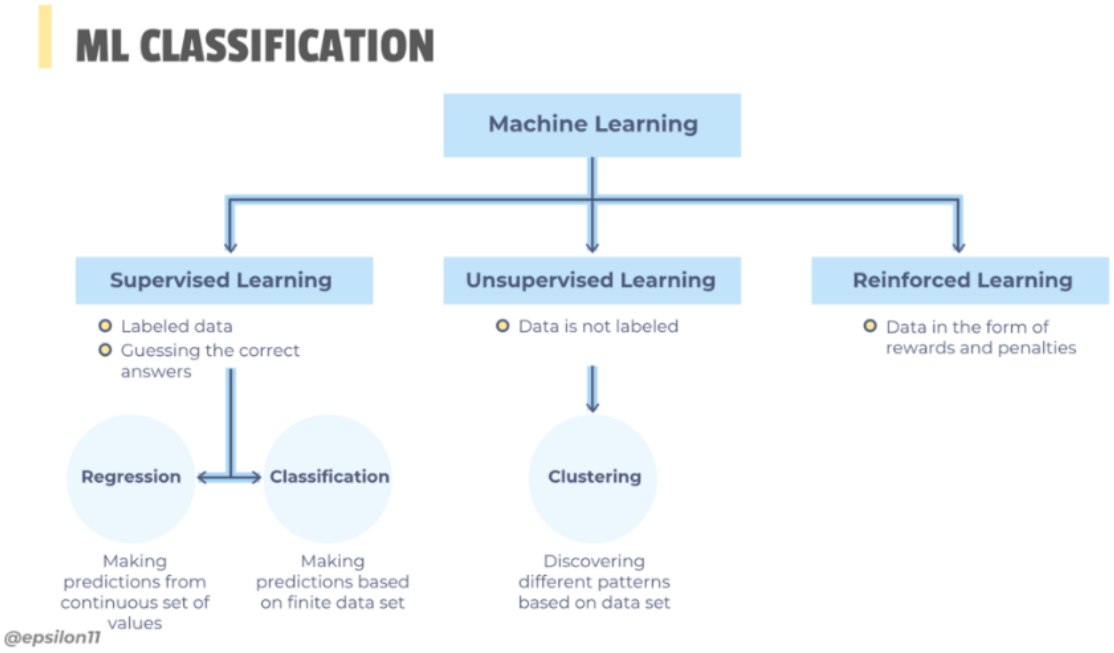

Figure 1-2: Machine learning types (7). In this research work, supervised learning and reinforcement learning will be utilized to overcome challenges in production forecasting models.

This research work will utilize reinforcement learning and supervised learning to address two current challenges in subsurface energy production forecasting models. In chapter 2, we will utilize reinforcement learning to develop an algorithm for solving the history matching problem in parallel. This algorithm can help reservoir engineers refine and enhance oil and gas reservoir models in a timely manner. The research findings in chapter 2 have been published in Energies Journal, in a paper titled: "Parallel Automatic History Matching Algorithm Using Reinforcement Learning" (8).

After that, chapter 3 employs supervised learning to develop more accurate shale gas production forecasting models. To overcome the challenges of data scarcity, transfer learning techniques are applied to the supervised learning models. The research findings in chapter 3 have been published in the Upstream Oil and Gas Technology Journal, in a paper titled: "Towards better shale gas production forecasting using transfer learning" (9).

In this research work, we will explore the following areas and try to answer the following questions:

- What scale of history matching complexity can reinforcement learning solve in a timely manner?
- How can human intuition and professional experience affect the speed of the algorithm and the quality of the solutions found?
- How does the initial guess in history matching affect the convergence of the suggested algorithm?
- How can machine learning utilize data from a similar domain to generate accurate forecasting models?
- How does transfer learning help to develop accurate production forecasts if the available data is very limited?

Chapter 2

Parallel automatic history matching algorithm using reinforcement learning

2.1 Background

Optimally developing an oil and gas field requires predicting future production using a reservoir model, whose key material properties are tuned in a process called history matching. This process of adjusting the key parameters is non-unique and computationally challenging. Typically, the reservoir model is divided into cells that match the geology of the field. The key properties of these cells, such as porosity and permeability, are assigned initially using core sample data, where available. For computational efficiency, the geological model is converted to a reservoir model using upscaling (10, 11, 12) to reduce the number of the cells in the model.

Due to the challenges of finding the key properties in each cell, history matching is used to adjust the values of these properties so the model reflects historical production data (13, 14, 15). History matching is typically done by matching the computed pressure and saturation data (oil, gas and water rates) from the simulation model and comparing it the actual historical data. The difference between the actual data and

data generated by the reservoir model is then computed using an objective function that quantifies the mismatch between the two quantities.

The problem of history matching can be expressed mathematically as a non-linear least-square optimization problem, where the optimization algorithm minimizes the objective function:

$$F(u) = \alpha[(q - \hat{q})^T C_q (q - \hat{q}) + \lambda (u - u^{prior})^T C_u (u - u^{prior})] \tag{2.1}$$

where α is a constant scaling factor used scale the quantity of the objective function, q is a vector containing the actual historical pressure or saturation quantity and $\hat{q}$ is a vector that represents the calculated quantity from the simulation model. C_q is a weight matrix used to assign weights to production data where it can be used if there is an order of magnitude difference in the production data. u is a vector containing the uncertain parameters in the reservoir model. The rates generated by the reservoir simulator $\hat{q}$ can be expressed as a function of the reservoir simulator with respect to u i.e., $\hat{q} = f(u)$. The second half of the equation is a regularization term where λ refers to the regularization parameter, u^{prior} refers to a priori estimate of the uncertain parameters u and C_u is another weight matrix that can be used to assign weights for each uncertain parameter.

The history matching problem is an ill-posed problem where multiple solutions can be found to match historical data (16, 17, 18). Due to such ill-posedness, multiple solutions are used to better assess the uncertainty in the forecasts(19, 20). The regularization term in the objective function in Equation 2.1 is used to mitigate the effect of ill-posedness of the history marching problem (21, 22).

History matching can be done manually where an experienced professional tunes the parameters of interest to find a matching model. However, manual history matching can be time consuming and prone to human bias and error (23, 24). History matching can also be done with the help of computers, which is known as assisted or automatic history matching. In automatic history matching, different methods have been used to search for parameters values that minimizes the objective function.

The methods used in the literature include gradient-based, stochastic or probabilistic algorithms.

Gradient-based algorithms, such as Levenberg-Marquardt, are exploitative by nature where the algorithm follow the gradient direction in order to search for a minima. Gradient-based algorithms can be powerful as they can convergence quadratically to a local minima under certain conditions (25). However, these algorithms are not feasible for large problems with a large number of parameters (26).

Stochastic optimization algorithms such as Genetic Algorithm which is inspired by the biological evolution of genes can better explore the parameters space. By allowing the parameters to evolve independently and then combining the results to choose the model with least error, Genetic Algorithm can handle large number of parameters (27). However, they require a large number of of function evaluations compared to gradient-based algorithms (28).

Probabilistic algorithms, such as Ensemble Kalman Filter (EnKF), uses an ensemble of realizations to represent the model uncertainty where these realizations are updated using a variance minimizing scheme (29). EnKF require fewer function evaluations but may not converge, and the parameters are assumed to have a Gaussian distribution (27, 28). Li and Misra (28) showed that the history matching problem can alternatively be solved by formulating it as a Markov Decision Process and then utilizing reinforcement learning methods. While Li and Misra (28) showed a successful proof of concept, their work was limited to a simple model with a small number of parameters.

The scalability and applicability of such approach to a more realistic complex model is yet to be established. The research detailed in this chapter introduces a novel algorithm where the use of a parallel stochastic reinforcement learning policy can efficiently find multiple solutions to the problem using a more complex 3D model and up to 27,000 uncertain parameters. Such parallelization allows for utilizing more computing resources to find multiple solutions to large models in a timely manner.

Solving the history matching problem in parallel using deterministic gradient descent algorithms can be complicated as such algorithms require to know the previous

objective function before taking a minimization step. Due to the fact that this kind of problems requires a great deal of function evaluations, a lot of research work have been done in order to parallelize the problem or some aspects of it. In the literature, ensemble Kalman filter was used to solve the history matching problem in parallel (30, 31, 32). Tanaka et al. (33) developed an optimization workflow in which the field development optimization can be solved in parallel. Sarma et al. (34) showed that the model based optimization and uncertainty quantification can be massively parallelized across thousands of commercial cloud computing nodes.

However, to the best of the authors' knowledge, at the time of publishing this research work no previous work has shown that the history matching problem can be solved in parallel using reinforcement learning. In addition, no work has been done previously to show that employing a stochastic policy in reinforcement learning can lead to finding multiple and different solutions to the history matching problem.

Section 2.2 of this chapter shows the details of the two reservoir models used as test cases and how their historical data is obtained. Then, Section 2.3 explains in details the methodology in which the suggested algorithm is developed and how it can be utilized to find multiple history matched models. Next, Section 2.4 shows the results of applying the algorithm on the two data sets and how the algorithm scales when increasing the number of computing resources. After that, Section 2.5 offers a thorough discussion and analysis of the algorithm and results along with future research work related to improving the algorithm.

2.2 Data

2.2.1 SPE9

To illustrate the algorithm capacity to find multiple solutions to the history matching problem, SPE9 reservoir model (35, 36) will be used. The reservoir model developed by Killough (37) is a three-dimensional 9000 cells models with 24 cells in the X direction, 25 cells in the Y direction and 15 cells in the Z direction. The reservoir

model contains 25 producer wells and one injector well as shown in Figure 2-1 and 2-2.

The well controls for this model are set as described by Killough (37) and Open Porous Media data repository (36), where the water injector is set to a maximum rate of 5000 STBW/day with 4000 PSIA as the maximum bottom whole pressure at 9110 ft reference depth. For the producer wells, a 1500 STBO/day is set as the maximum rate at the beginning with a minimum flowing bottom whole pressure of 1000 PSIA at a reference depth of 9110 ft for all the wells. The model described will be considered as the truth model where the reservoir simulator will simulate this truth model to generate field oil production rate (FOPR) values. The data generated contains five years worth of FOPR values collected every 15 days.

After that, for each cell, the permeability values K_x, K_y and K_z are scrambled with a geo-spatially correlated random noise to generate a starting point for the algorithm. The starting point is generated by adding the random noise vector that contains both positive and negative entries to the truth model permeability values. The goal of using a geo-spatial random noise is to ensure that the starting model presents a realistic model where the permeability values are mostly geo-correlated. Figure 2-3 shows the noise added to the permeability values. Since it is difficult to plot all values, the figure only shows a sample of the added noise on the first and last layer of the reservoir model. However, it is important to note that all permeability values across all layers have been scrambled with noise in order to generate the starting point for the algorithm. The added noise have altered the statistical mean by 7.4% in K_x, 19.0% in K_y and by 8.0% in K_z compared to the truth model.

The truth model is then discarded and only its FOPR values are kept to be considered as the historical values q. The starting point after scrambling the truth values is shown in Figure 2-4. The goal of starting with a known model then scrambling its permeability values is to imitate a real life scenario where the actual uncertain values (in this case permeability values) are not known exactly and the historical pressure or saturation data (in this case FOPR) are used to tune the uncertain parameters.

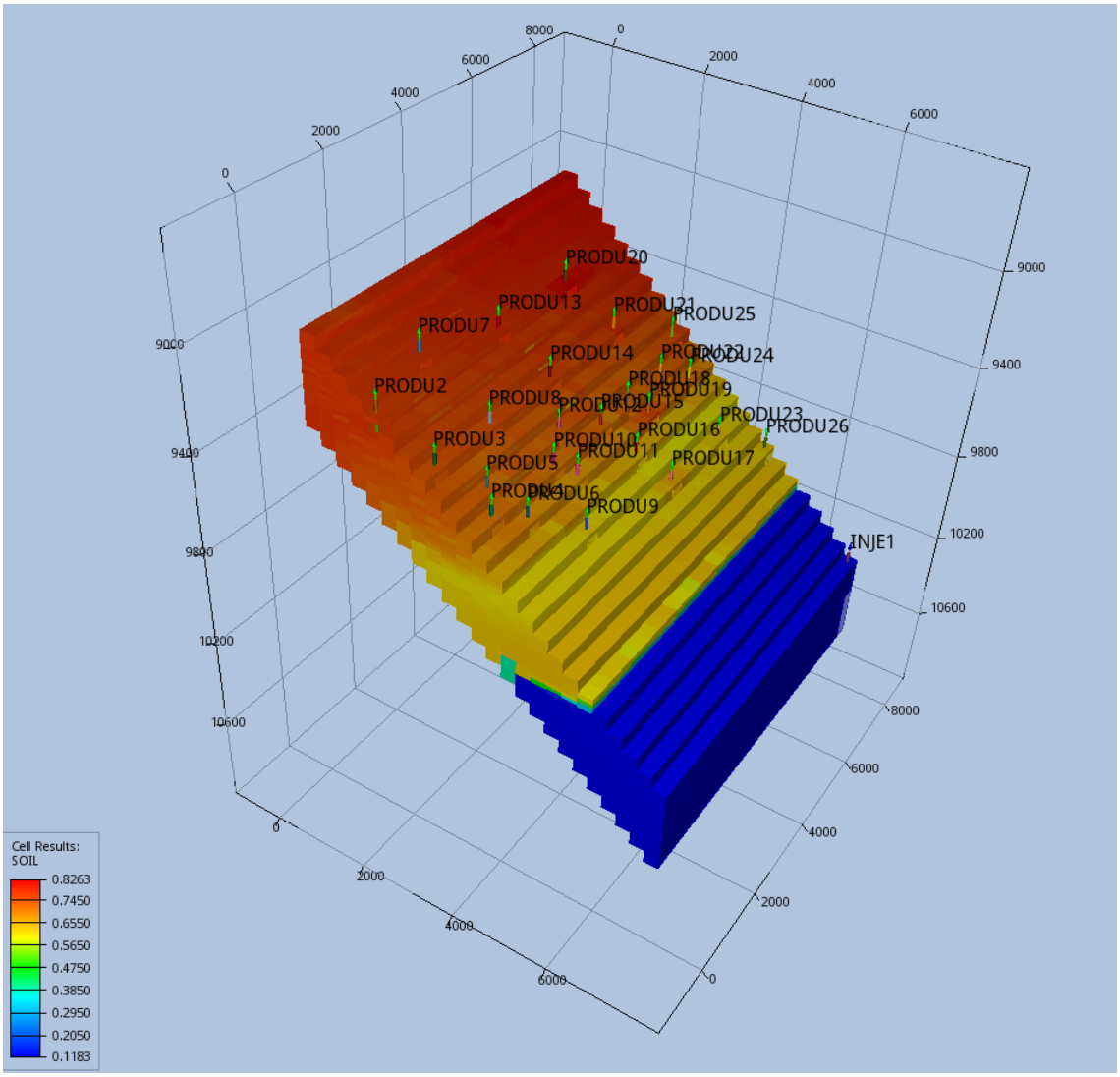

Figure 2-1: SPE9 reservoir model grid plot. SPE9 reservoir model is a three-dimensional 9000 cells models with 24 cells in the X direction, 25 cells in the Y direction and 15 cells in the Z direction. The reservoir model contains 25 producer wells and one injector well.

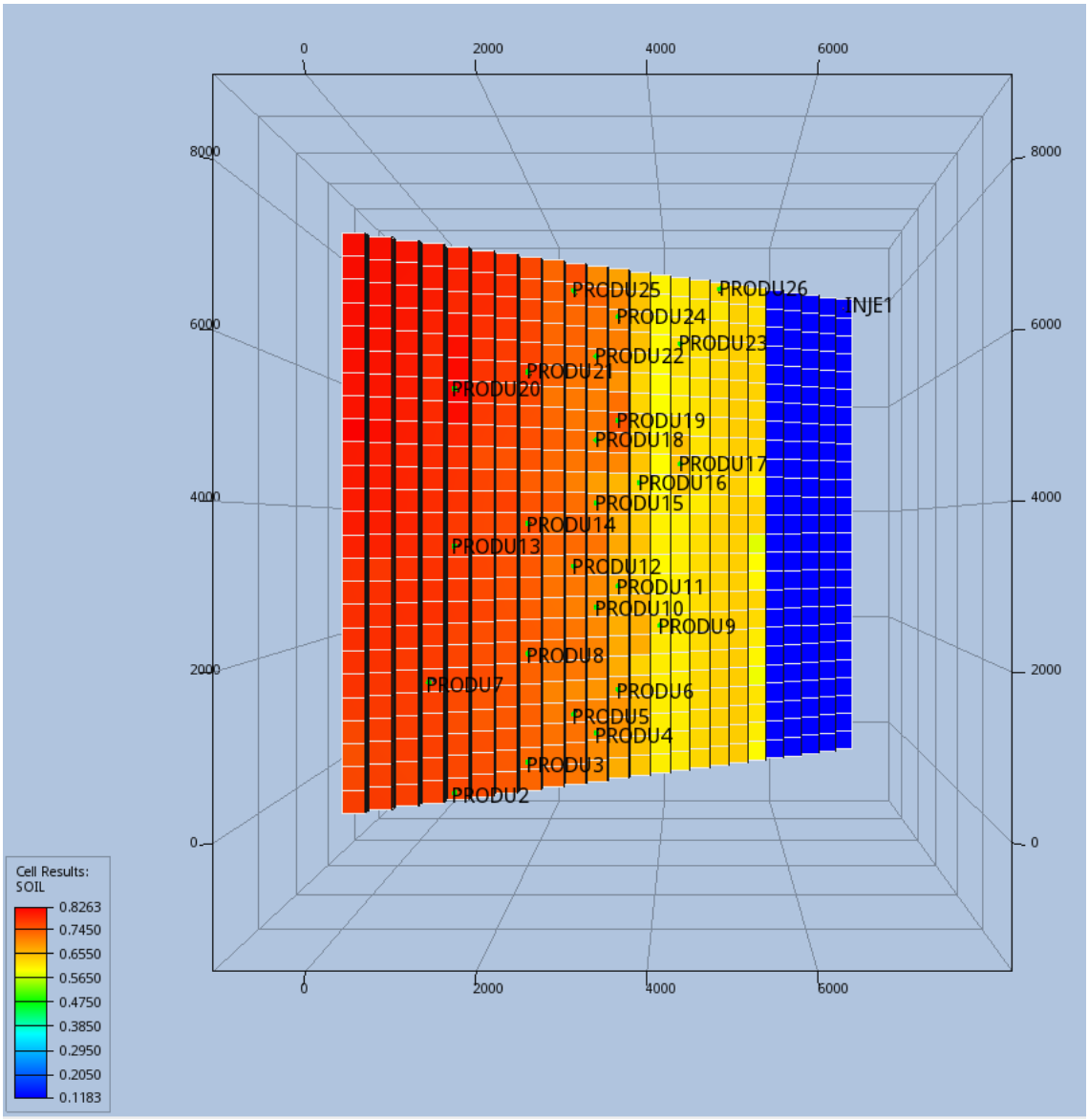

Figure 2-2: SPE9 reservoir model grid (top view).

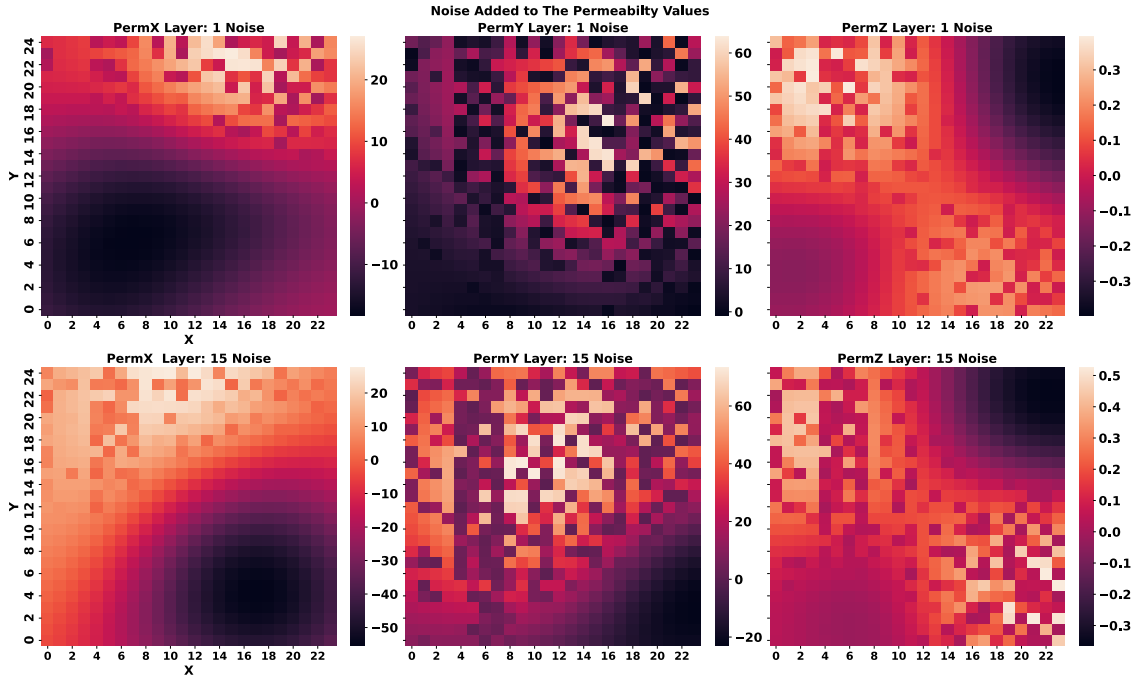

Figure 2-3: The geo-correlated random noise added to the permeability values in each cell in order to create a synthetic case for the algorithm. The plot shows the first and last layer of the noise added to the truth model permeability values in the X, Y and Z directions.

2.2.2 SPE1

To measure the scalability of the suggested algorithm, a smaller reservoir model is used. The main reason for choosing a smaller model is the fact that the scalability test requires repeating the experiment for a number of trials as it will be discussed in the Methodology Section.

The model used for testing the scalability in this chapter is a three-dimensional 300 cells model with 10 cells in the X direction, 10 cells in the Y direction and 3 cells in the Z direction as shown in in Figure 2-5. The reservoir model has one injector located in cell ($x = 1$, $y = 1$,$z = 1$) and one producer located in cell ($x = 10$, $y = 10$,$z = 1$) where the measurement of BHP is considered in the objective function.

The well controls for this model are set as described by Open Porous Media data repository (36), Odeh (38), where the gas injector (INJ) is set to a maximum rate of 100 MMscf/day with 9014 PSIA set as the maximum bottom whole pressure. For the producer well (PROD), a 20,000 STBO/day is set as the maximum rate with a

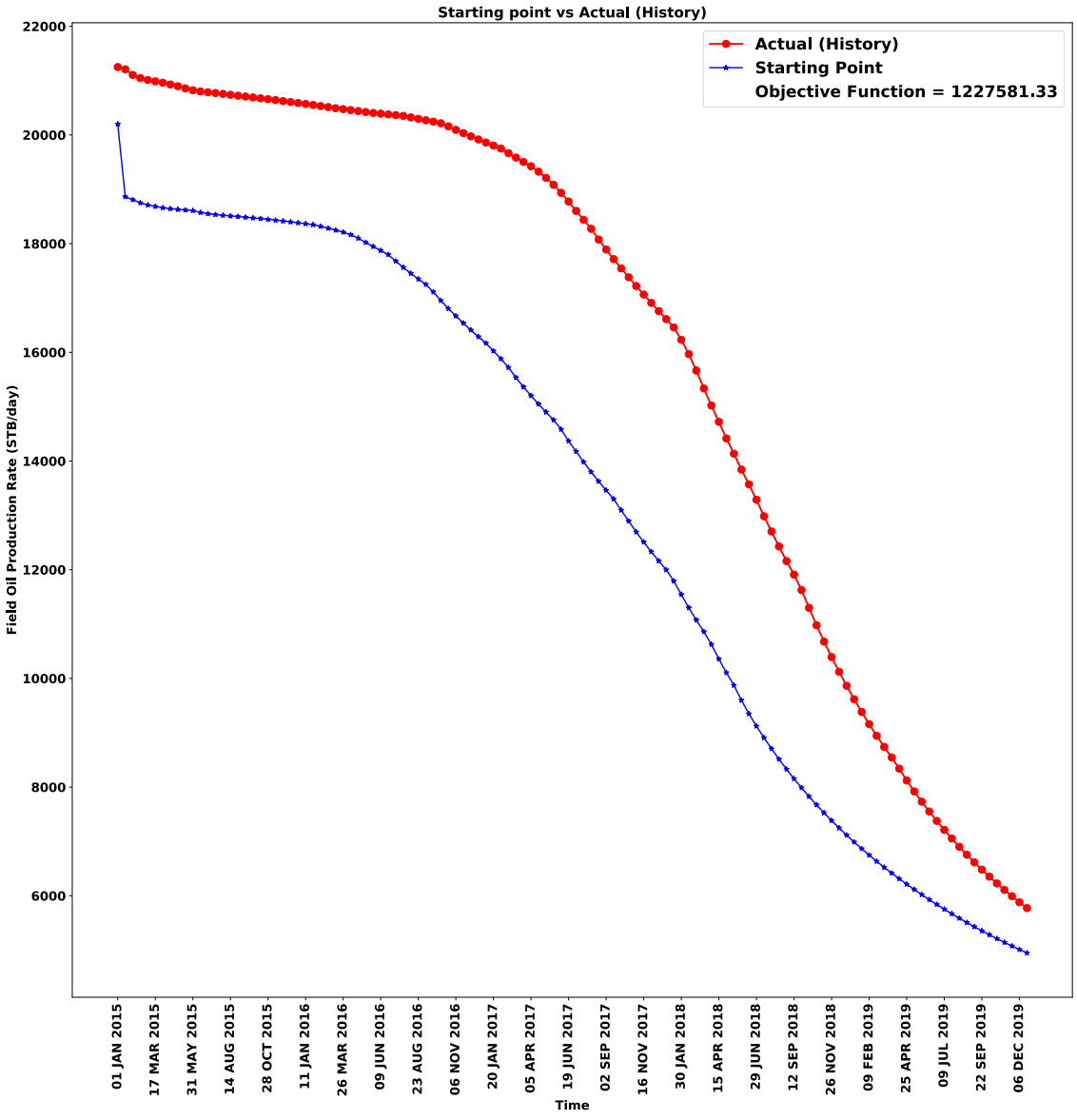

Figure 2-4: The difference between the actual historical FOPR (Field Oil Production Rate) values and the FOPR values generated from running the simulation on the starting point of the model. The objective function is calculated using Equation 2.1.

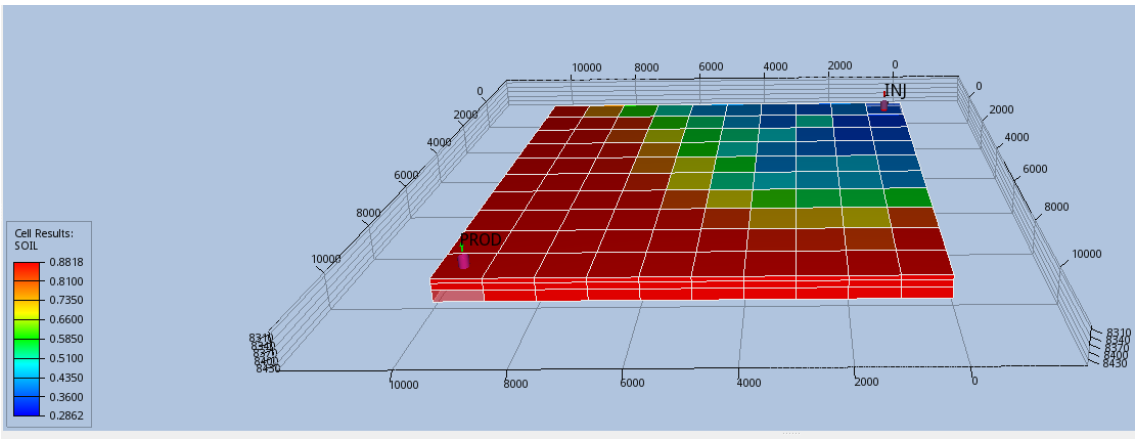

Figure 2-5: SPE1 reservoir model grid. SPE1 reservoir model is small 3D model with 10 cells in the X direction, 10 cells in the Y direction and 3 cells in the Z direction.The reservoir model has one injector located in cell (x = 1, y = 1,z = 1) and one producer located in cell (x = 10, y = 10,z = 1) where the measurement of BHP is recorded.

minimum flowing bottom whole pressure of 1000 PSIA.

The second data set, shown in Figure 2-6, employs a synthetic historical data obtained from running SPE1 (35, 36, 38) by following the same procedure used in the first data set, except that in this data set the Bottom Hole Pressure (BHP) values for the producer well is used to match the history instead of FOPR.

2.3 Methodology

2.3.1 Reinforcement Learning

In recent years, machine learning has been employed extensively to tackle research problems thanks to its capacity to map input data to a useful output by identifying certain features. In the literature, machine learning tasks usually falls into one of three categories: supervised learning, unsupervised learning and reinforcement learning (39). In supervised learning, a model is trained with labeled data i.e., examples in order to later predict new data with no label.

In unsupervised learning, the model is trained with unlabeled data points to identify certain features and subsequently classify them into clusters. In reinforcement learning, the model learns a policy that maximizes a certain reward in an environ-

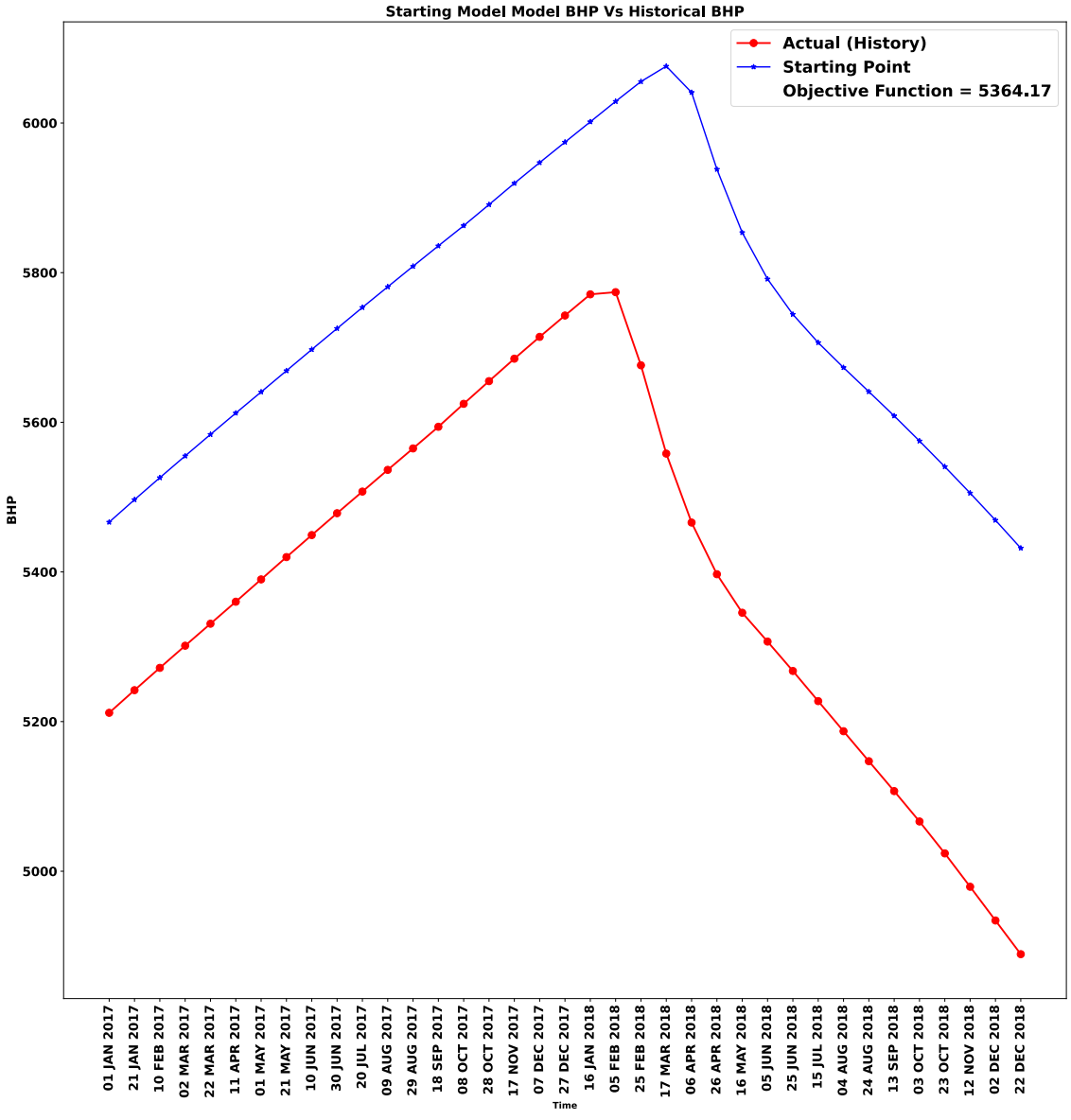

Figure 2-6: The difference between the actual historical BHP (Bottom Hole Pressure) values and the BHP values generated from running the simulation on the starting point of the model. The objective function is calculated using Equation 2.1.

ment of interest. The policy is often learned by directly interacting with the environment (40).

In the oil and gas industry, machine learning have been used extensively in a wide variety of problems. Montgomery et al. (41) used supervised learning to build data driven models to forecast shale gas production where Zhang et al. (42) developed a multi-component method for reservoir characterisation using unsupervised learning. Miftakhov et al. (43) used reinforcement learning to maximize the Net Present Value (NPV) of waterflooding by training a reinforcement learning agent to control the water injection rate.

Reinforcement learning is somewhat similar to the way humans learn by interacting with an environment and using a trial and error mechanism. The reinforcement learning agent, as shown in Figure 2-7, observes the state of the environment, takes an action and collects a reward for its action. The agent will learn from these interactions how to search the parameters space in order to maximize its rewards. Reinforcement learning gained more popularity after van Hasselt et al. (44) showed that a reinforcement learning agent was capable of achieving super human intelligence just by monitoring pixels on the screen. However, reinforcement learning is more generic, and is suitable for all problems that can be formulated a Markov Decision Process (MDP).

An MDP represents the sequential decision making paradigm in which the actions taken by the agent does not only affect the immediate reward but it also affects future long term rewards and future states (45). MDPs are often defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition kernel, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, and γ denotes the discount factor. For the transition kernel, we use $\mathcal{P}(s' \mid s, a)$ to denote the probability of transitioning from state s to state s' if action a is taken. We denote a stochastic policy by $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where we use $\pi(a \mid s)$ to denote the probability of choosing action a when in state s under policy π.

Under each policy, the value function $V_\pi(s)$ represents the reward-to-go when

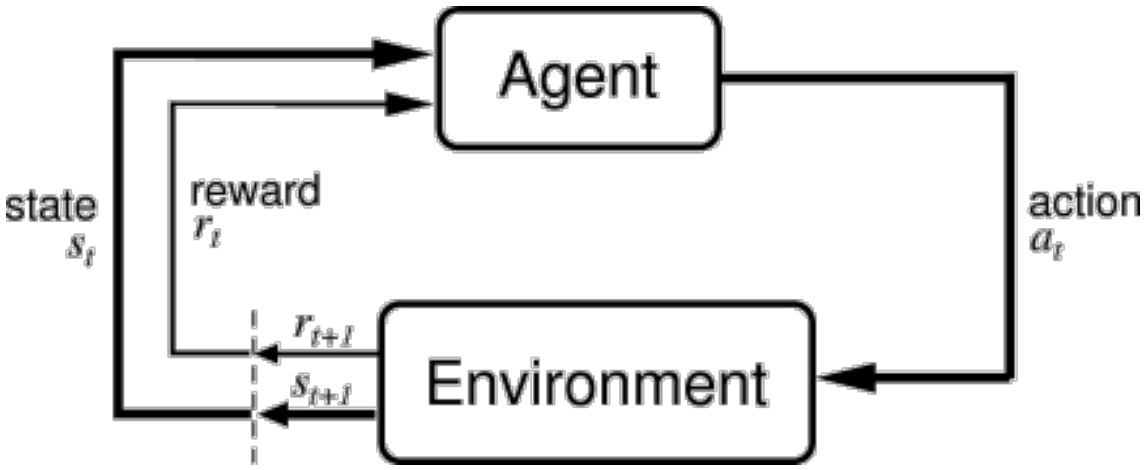

Figure 2-7: Sutton and Barto (45) illustration of how the artificial deep neural network agent learns by interacting with the environment. The agent reads the current state of the environment, takes an action and collects a reward for the action taken.

starting from state s and using policy π. Specifically, we have

$$V_\pi(s_t) = \mathbb{E}\left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+l}, a_{t+l})\right], \tag{2.2}$$

where $s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t)$, and $a_t \sim \pi(a_t \mid s_t)$, and the expectations are naturally taken with respect to the randomness in the policy and the transition kernel. In reinforcement learning, the goal is often to estimate the optimal policy $\pi^*(s) = \arg\max_\pi V_\pi(s)$ or the optimal value function $V^*(s) = \max_\pi V_\pi(s)$.

2.3.2 History Matching Using Reinforcement Learning

In this chapter, we utilize reinforcement learning to solve the history matching problem by formulating it into a Markov Decision Process as illustrated in Figure 2-8. By using the reservoir simulator as an environment where the reinforcement learning agent Ag can take action a_t and receive the new state of the environment s_t along with a reward r_t quantifying how good the action taken by the agent is. The current state returned by the environment is a vector containing all the uncertain parameters that need to be tuned in order to match the history and is equivalent to u in Equation 2.1.

For both cases used in this chapter, K_x, K_y and K_z permeability values of each cell are stacked into a vector as follows:

$$\mathbb{U} = \begin{bmatrix} K_x(x=0,y=0,z=0) \\ \vdots \\ K_x(x=X,y=Y,z=Z) \\ K_y(x=0,y=0,z=0) \\ \vdots \\ K_y(x=X,y=Y,z=Z) \\ K_z(x=0,y=0,z=0) \\ \vdots \\ K_z(x=X,y=Y,z=Z) \end{bmatrix} \tag{2.3}$$

where this vector is considered as the state of the environment s_t. The action a_t taken by the agent is also a vector with the same dimensions as the state, allowing the artificial deep neural network agent to act on the uncertain parameters and adjusting them to find a solution. At each reinforcement learning time-step t, the action taken by the agent is given a reward value r_t to quantify the quality of the action allows the agent to learn whether the action it took is good or bad. In this work, Open Porous Media Flow reservoir simulator (46) is used in the reinforcement learning environment.

Formulating the history matching problem in such manner allows for the use of reinforcement learning, where the agent will be able to learn from the reservoir simulator how to take actions that lead to minimizing the objective function. The agent is allowed to only select actions that are within a certain range i.e., the agent action space $\mathcal{A}$ at each entry of the action vector is = [-K_Δ,+K_Δ]. It is important to note that K_Δ is a design parameter that can be chosen by the engineers running the experiment based on their knowledge of the reservoir model at hand. The choice of K_Δ must not be too small nor too large. Choosing a small K_Δ value will change the objective function by a very small quantity, causing very slow convergence. While, choosing a large value can encourage the agent to take actions that push the permeability values

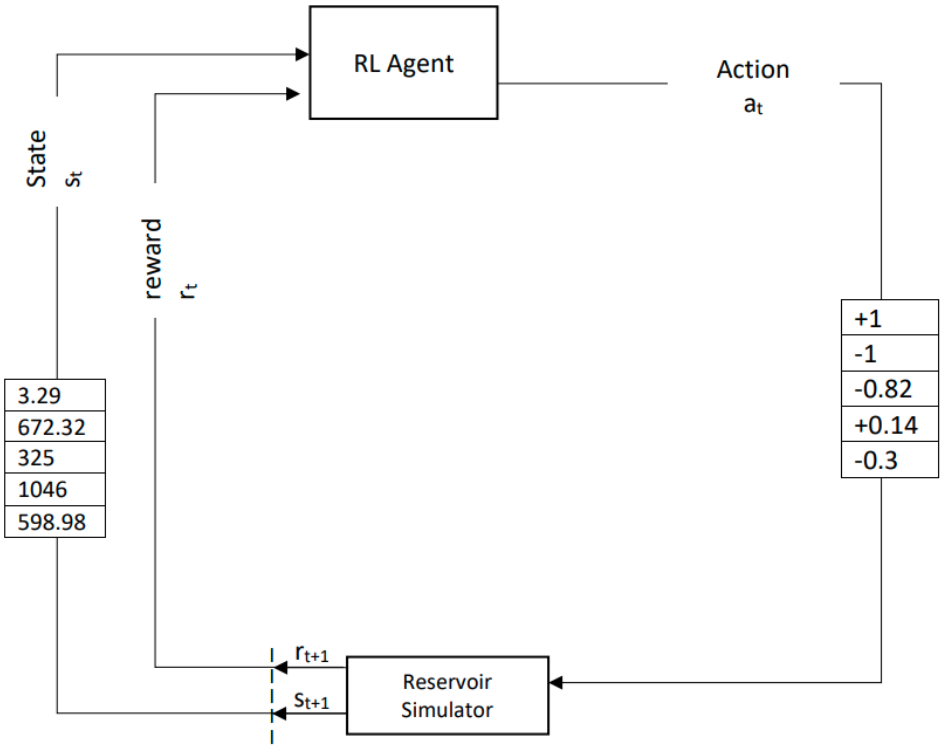

Figure 2-8: Reformulating the history matching problem from a least-square mathematical optimization problem into a Markov Decision Process by creating an environment that allows the agent to interact with the reservoir simulator. The agents observes the current state s_t of the uncertain parameters u, takes action a_t against these parameters and collects a reward r_t quantifying the quality of its action.

out of allowed bounds.

If the agent takes an action that causes the current state s_t to be out of bounds (outside of all possible state space $\mathcal{S}$), then the current state is clipped to satisfy physical restrictions. For example, if the agent takes an action that results in negative permeability values in some of the vector entries, then these values are set to zero in order to avoid feeding the reservoir simulator negative permeability values causing it to crash. The K_Δ parameter is similar to the step size parameter in gradient descent optimization algorithms where selecting a very small value will slow the convergence process and a big value may cause divergence. Another approach can be used to avoid reaching states with negative permeability values is to change the actions of the agent to multiply the current permeability value instead of addition or subtraction. By setting K_Δ to a small value, to avoid large updates, for example 0.1 the agent can sample an action between $1 + K_\Delta$ and $1 - K_\Delta$ in order to tune the permeability.

Reinforcement learning process consist of episodes where an episode is defined as a series of reinforcement learning time-steps starting from the initial state and ending when a termination criteria is met. This termination criteria is usually governed by a reward function. Typically, in reinforcement learning the reward function is designed in manner where the agent is given a positive reward for good desired actions and a negative reward for bad undesired actions.

However, choosing a reward function can be a challenging task as sometimes there are good actions and even better actions or bad actions and worse actions which can be difficult to quantify. Luckily, in the history matching problem, the quality of the action can be measured directly from the objective function, where the difference between the value of the previous objective function and the value of the new objective function after the agent took an action can quantify the quality of the match.

In the context of applying reinforcement learning to solve the history matching problem, the reward function can be computed as:

$$r_t = \mathbb{F}_{t-1} - \mathbb{F}_t \tag{2.4}$$

$\mathcal{F}_{t-1}$ is computed using Equation 2.1 and refers to the value of the objective function at time $= (t-1)$ before the agent takes an action, whereas $\mathcal{F}_t$ refers to the value of the objective function after the agent takes an action. This reward function will return a positive value if the agent took a good action that led to reducing the objective function and a negative value if the agent took a bad action that caused the objective function to become larger.

The advantage of using this reward function is that it will assign larger rewards to actions that leads to a larger reduction in the objective function, thus helping to guide the agent to take actions that will speed-up the convergence process. Assigning a higher reward to an action increases the probability of choosing that action again given the same state. The value of the objective function is scaled down using a scaling factor α in order to avoid feeding the deep neural networks very large quantities that might destabilize the training process.

The choice of the error quantification function can affect the reward function and subsequently the performance of the algorithm. Equation 2.1 utilizes a weighted squared error formula for the objective function. The weighted squared error is used since it is the common choice for history matching problems and it would provide the agent with a good feedback that quantify the quality of its action. However, weighted root squared error might provide a better error quantification for the algorithm as it would provide a somewhat uniform reward function to the agent unlike squared error, which penalizes larger error more severely possibly leading to a non-uniform reward function.

The reward function defined in Equation 2.4 is not enough on its own and it needs to address termination criteria when the agent takes an action that reduces the objective function to the tolerance level accepted. In order to address that, the agent is assigned a big constant reward (i.e., 50,000) to incentivize it with a big reward upon good episode termination. In addition, in order to save computing power and incentivize the agent to take actions that speed up the convergence process, the environment sets a time limitation (47) on the maximum time-steps allowed per episode. Once the maximum time-steps is reached without meeting any termination

criteria, the episode terminates with a constant negative reward (i.e., -20,000) if the agent does not find a solution.

The maximum time-steps per episode parameter is also a design parameter and should be chosen based on the problem at hand but it must not be too small. Small time-steps per episode can hinder the agent from exploring the environment properly due to the fact that in reinforcement learning the agent sometimes might pick actions that are not good at the current time-step, but can lead to better rewards in the future time-steps.

Furthermore, if the agent diverges away from the solution by making a sequence of bad decisions rendering the objective function to grow large. In this case, the episode is terminated with a big negative constant rewards (i.e., -50,000) as soon as the objective function value becomes twice as large as the initial objective function. In addition to saving computing power, such a limitation on the value of the objective function will punish the agent so it can learn from that mistake and avoid following such a trajectory in the future. The rewards of termination criteria are also design parameters and the engineers running the experiment should choose proper values relevant to the problem at hand. The rewards should be chosen so they can incentives the agent towards meeting the tolerance criteria as well as providing a good feedback to the engineers so they can monitor the learning process.

This formulation of the history matching problem into a Markov Decision Process means that at each reinforcement learning time-step, a simulation run is required to compute the new objective function and assess the quality of the action taken by the agent.

2.3.3 Proximal Policy Optimization

In this work, we use Proximal Policy Optimization (PPO) (48) algorithm to find multiple solutions to the history matching problem. PPO algorithm is an actor-critic method in which the agent utilizes two deep neural networks parameterized by θ, one for the actor and one for the critic as shown in Figure 2-9. The value function and the policy are represented by these two deep neural networks. Specifically, the actor

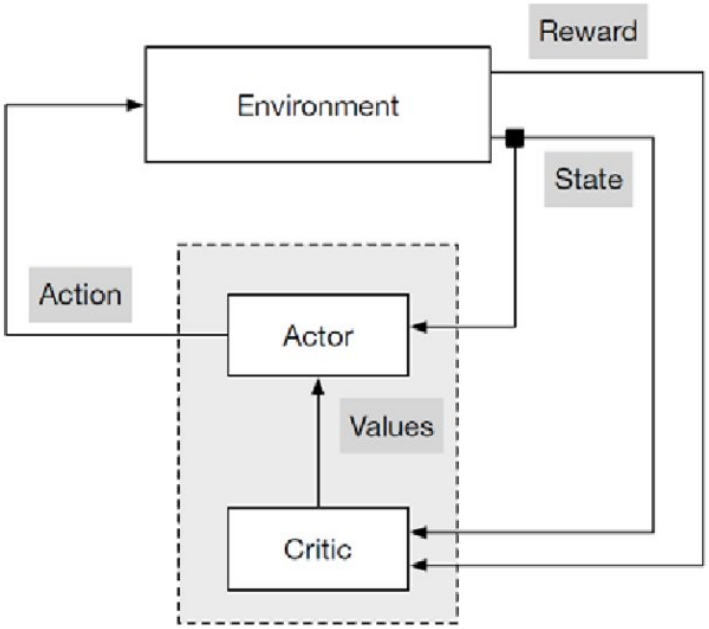

Figure 2-9: Diederichs (49) illustration of actor-critic architecture.

and critic networks represent the policy and the value function, respectively.

One of the prevalent issues in policy gradient methods is the destructively large policy updates. To overcome this issue, PPO suggests a simple surrogate objective that penalizes large changes in the policy update. Specifically, the objective function in PPO is as follows

$$L_t(\theta) = \mathbb{E}[L_t^{clip}(\theta) - c_1 L_t^{VF}(\theta)], \tag{2.5}$$

where $\mathbb{E}$ refers to the expectation operator, $L_t^{clip}(\theta)$ represents the clipped objective function for the policy network and L_t^{VF} represents the error in estimating the value function. Specifically, for the policy loss, the clipped objective function is defined as

$$L_t{}^{clip}(\theta) = \mathbb{E}_t[\min(\delta_t(\theta)\hat{A}_t, \text{clip}(\delta_t(\theta), 1-e, 1+e))], \tag{2.6}$$

where $\delta_t(\theta)$ is a probability ratio of the new policy to the old policy computed as $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, e is a clipping range variable and is set to 0.2 in the case study, and $\hat{A}_t$

represents the estimate of the advantage function calculated as

$$\hat{A}_t = \sum_{k=0}^{T} \gamma^k r_{t+k} - V_\theta(s_t), \tag{2.7}$$

where $V_\theta(s_t)$ is the value function estimate given by the value network.

Note that the advantage function at time t is defined as the discounted sum of rewards (starting from t) minus a baseline. The value function at s_t is often used as the baseline, as done in PPO. The goal of the clipping operator is to stop drastic policy updates based on the noisy estimation $\hat{A}_t$, and hence it avoids drastic policy updates. Further, $L_t^{VF}(\theta)$, which denotes the loss in estimating the value function is defined as

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t)^2 = \hat{A}_t^2, \tag{2.8}$$

where $V_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$ are samples estimates collected from the environment, and T is the maximum number of steps. Recall that the value function $V_\theta(s_t)$ represents an *estimate* of the expected rewards for a given state s_t at a given time $= t$.

This estimate is estimated using the critic network, which along with the policy network, is updated during training. Recall also that γ is a discount factor which controls the trade off between short-term and long-term rewards. Specifically, as γ decreases, more emphasis is given to short-term rewards.

While seemingly complicated, the objective function described above is quite intuitive. Minimizing L_t^{VF} ensures that the critic's estimate of the value function is close to what the observed samples suggest. On the other hand, maximising ${L_t}^{clip}$, if we ignore the clipping, encourages policies with a better advantage, i.e., policy that chooses relatively better actions. The clipping, as mentioned above, mitigate potentially destructive updates due to potentially huge values of δ_t.

Note that while PPO is used in this work, other actor-critic methods such as A2C (50) which supports sampling from parallel environments can also work with this problem formulation. However, PPO outperforms other alternatives in terms of speed and quality of solutions found. In addition, PPO is sample efficient which is

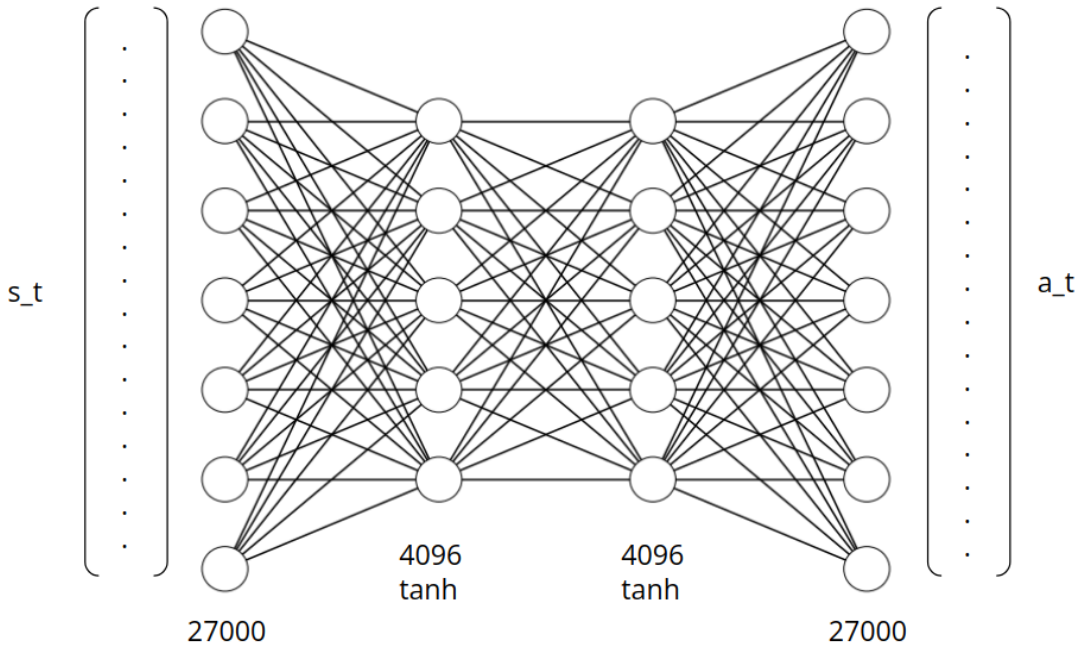

Figure 2-10: The deep neural network design for the actor network π_θ is composed of an input layer containing 27,000 neurons allowing it to observe the current state of each uncertain parameter, two hidden layers where each layer contains 4096 tanh activated neurons and an output layer that also equal to 27,000 allowing it to take action against 27,000 different uncertain values.

very important in the history matching problem due to the fact that the objective function computation is costly.

In order to find multiple different solutions, it is important that we use a stochastic policy when we interact with the environment. Specifically, a stochastic policy will lead to a better exploration of the state-action space and hence it will find multiple solutions in each run.

The deep neural network design for the actor network π_θ is shown in Figure 2-10. It is composed of an input layer containing 27,000 neurons (equal to the number of uncertain parameters in SPE9); two hidden layers where each layer contains 4096 tanh activated neurons; and an output layer that is also equal to 27,000 as it needs to take action against 27,000 different uncertain values. The critic network $V(\theta)$, as shown in Figure 2-11, has the same structure as the actor network except for the last layer where it has only one neuron since it is only trying to estimate a scalar value.

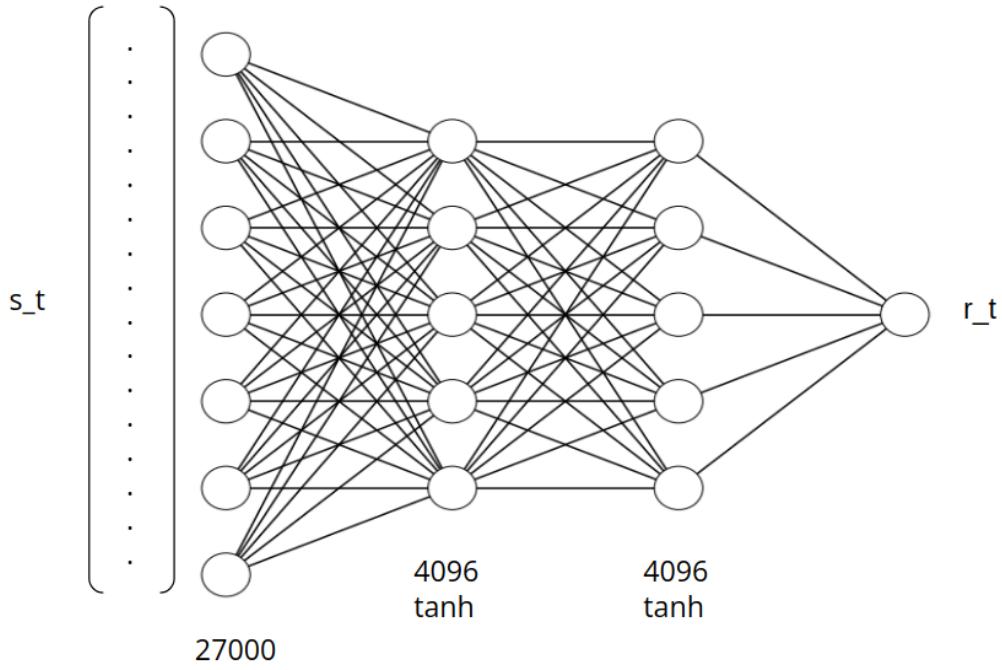

Figure 2-11: The deep neural network design for the critic network $V(\theta)$ is composed of an input layer containing 27,000 to observe the current state of the environment, two hidden layers where each layer contains 4096 tanh activated neurons and an output layer that also equal to one neuron as it only needs to estimate one value.

The second data set uses the same structure. However, since the problem has much fewer uncertain parameters (900 compared to 27,000) resulting in 900 neurons in the input and output layers. In addition, the number of the hidden neurons in the hidden layer is equal to 128 instead of 4096 used in the first data set.

2.3.4 Parallel Reinforcement Learning History Matching

History matching in general is a serial problem where the use of optimization methods to find the minimum of the objective function via gradient descent makes it necessary to know the previous objective function before taking the next step. However, in reinforcement learning the agent learns how to find the minimum by interacting with the environment one step at a time and collecting these experiences then training the deep neural network agent. So in essence, the deep neural network is trying to map states into actions so the agent needs only the current state, the action taken and the reward assigned to this action in order tune its deep neural network weights and choose the decisions that will maximize the rewards. PPO allows for the experiences to be collected into a batch every few time-steps and then sent to the agent for training even if the episode is not completed.

The way in which the actor-critic reinforcement learning train and optimize its agent gives us the chance to speed up the history matching process by training the agent in parallel and allowing it to collect experiences from running multiple scenarios of the history matching process at once. This can be done by allowing the agent to simultaneously interact with N number of environments, observe different N states, take N number of different actions and collect N different rewards as shown in Figure 2-12. For example, in the case of SPE9, the batch size is set to 192 where the algorithm will run 192 experiments by interacting with the environment then collects these experiments in a batch then sends it to the agent to train its deep neural networks and update their weights. Instead of waiting for a single environment to collect 192 experiments, the algorithm will launch eight environments in parallel where each environment collects 24 experiments which allows the agent to collect the 192 experiments in a much shorter time period.

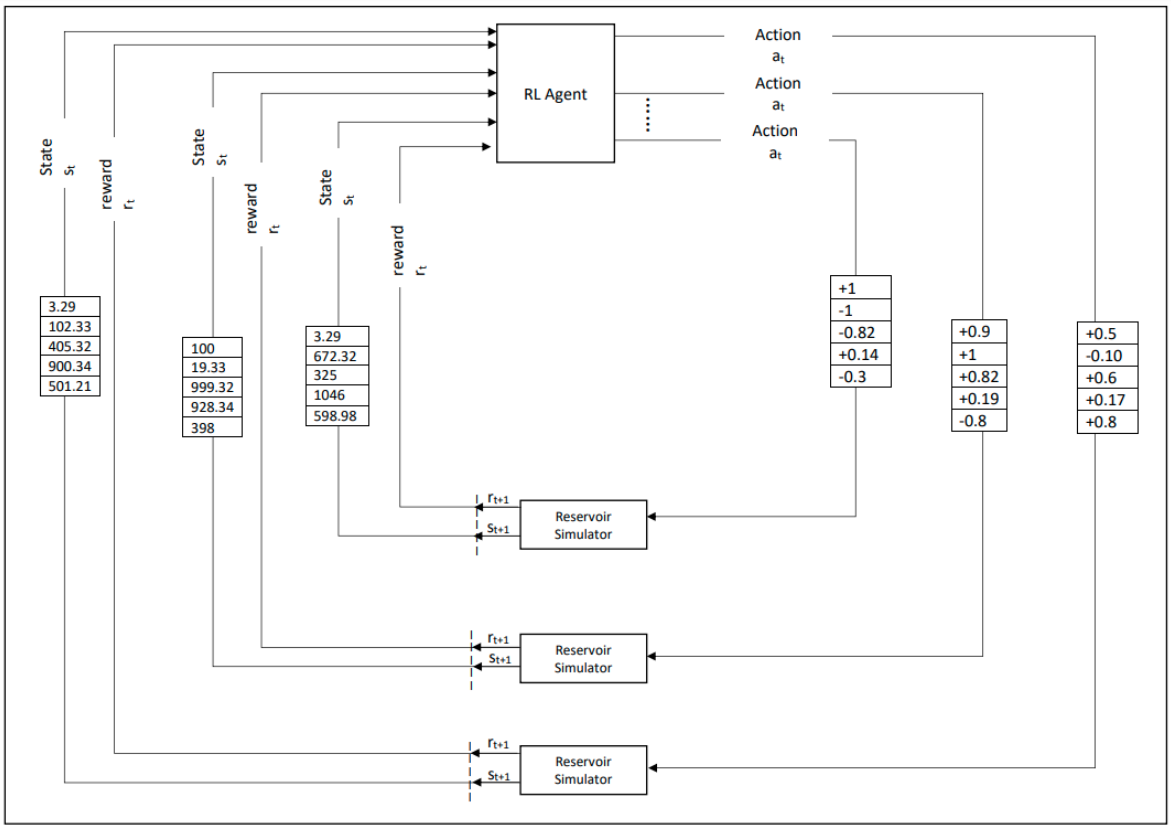

Figure 2-12: Redesigning the environment from Figure 2-8 in order allow the agent to learn from multiple simultaneous environments. The agent can observe N states, take N actions and collect N rewards in parallel using the new parallel architecture.

Stable Baselines 3 (51) implementation provides a wrapper that allow for the launch N CPU threads, each thread works on an environment to collect the experiences then these experiences are sent to the GPU to train the agent's deep neural networks in a faster manner. In addition to running multiple environments in parallel, inside each thread running an environment to collect the experiences, the environment itself can run the reservoir simulator in parallel to speed up the process of computing the objective function which is very common in reservoir simulation. For example, when running the experiment on SPE9, 4 MPI processes were used per environment.

However, it is very important to note that only the process of collecting the experiences and running the reservoir simulator can be done in parallel but each environment will run each episode in a serial manner and thus each environment must have its own desk directory with its own DATA, ECL and SMSPEC files. For

example, the state of environment 12 at time-step (t) will depend on the state and the action of environment 12 at time ($t-1$) as this process can not be parallelized. The details of the algorithm is shown in Algorithm 1 pseudocode.

Algorithm 1: Parallel Reinforcement Learning History Matching

```
Launch N parallel environments
Create active disk directory for each environment
Set B = Batch Size
Set t = 0
Previous Objective Function 𝔽_{t-1} = Initial Objective Function 𝔽_{t=0}
while t < max time-steps do
    do in parallel
        for i = 0, 1, ..., i < B/N do
            Observe state s_t
            Take Action a_t against uncertain parameters
            Run Reservoir Simulator with adjusted parameters
            Compute New Objective Function Value 𝔽_t
            Reward r_t = 𝔽_{t-1} − 𝔽_t
            𝔽_{t-1} = 𝔽_t
            if 𝔽_t < ϵ then
                save history matched reservoir model to disk
    Update π_θ & V networks using B experiments
```

2.3.5 Reproducibility

One of the major challenges faced while dealing with deep neural networks is reproducibility (52, 53) in which repeating the experiment may lead to a different result with a different run-time taken to find the solution. The reproducibility problem is caused by the random network weights initialization prior to training in addition to using stochastic optimizers to optimize such weights. Another factor contributing to the reproducibility problem is the stochastic policy used by PPO where a small random noise is added to the action suggested by the agent or in some cases some the actions taken by the agents are sampled from a random noise matrix to ensure that the agent explores the environment efficiently to find multiple solutions to the history matching problem.

The reproducibility problem does not affect the outcome of the results in the first data set as all the solutions found meet the tolerance criteria. However, it becomes important when running the scalability test on the second data set due to the fact that some runs might find a solution quickly and other runs may take a while to find a solution. For this reason, an approach similar to Alolayan et al. (9) was used to reduce the effects of the reproducibility problem where each of the scalability test run-time measurement is repeated 10 times and the average of these 10 runs is considered the final result.

2.4 Results

Algorithm 1 was used to run 20,000 reinforcement learning time-steps in parallel on the first data set to search the parameters space for solutions to the history matching problem. As shown in Figure 2-13, in the beginning of training the algorithm behaved as expected from any reinforcement learning algorithm where it kept making wrong decisions and collecting negative rewards. As the agent spends more time interacting with the environment, it learns how to map states into actions that enables it to increase its rewards.

Per the definition of the reward function in Equation 2.4, as the agent tries to increase its rewards, it will tune the uncertain parameters and find models that reduce the objective function. As the agent spends more time in the environment trying to maximize its reward, it will reduce the objective function to meet the tolerance criteria and thus it will find a solution to the history matching problem. As shown in Figure 2-14, thanks to the stochastic policy the agent uses, it can find multiple different solutions as it tries to take different trajectories to better explore the environment.

The approach of using a stochastic policy may encourage the agent to take bad actions and waste computing power. However, it is necessary for the agent to do so in order to search for new different solutions as this is always the case with the exploration-exploitation dilemma. Such behaviour can be seen from the agent's learning progress it appears in Figure 2-13 around the 13,000 time-step. Although the

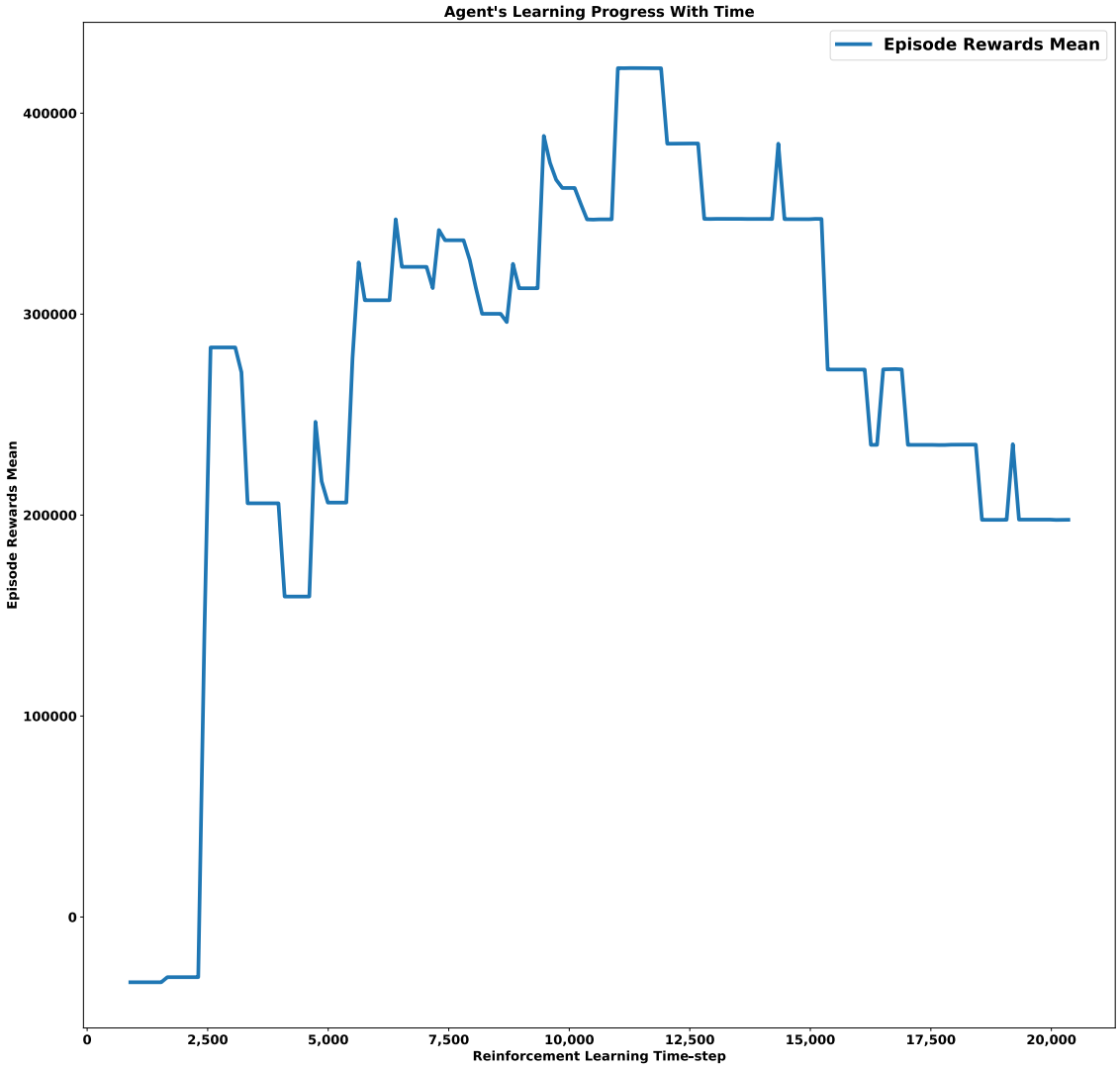

Figure 2-13: In the beginning of the training, the agent explore the environment often making bad actions and collecting negative rewards. Then, it starts to learn how to increase its rewards which enables it to find solutions that meet the tolerance criteria.

agent found few solutions and collected positive rewards by that time-step, it did not repeat its actions to maximize rewards. Instead, the agent explored the environment for new different solutions causing it to make decisions that resulted in less rewards than previously acquired.

In order to study the forecasting capabilities for the model realizations found by the artificial deep neural network agent, the quality of the models are then validated by running each model for an additional year that the agent did not train on. Figure 2-15 shows each realization forecast compared to the truth model and the starting point. As shown from the figure, the realizations found by the agent exhibited a good forecasting capabilities for the reservoir model and provided multiple production scenarios where some models over-predicted and some models under predicted.

This can help engineers run uncertainty quantification analysis with a higher degree of confidence. When using this algorithm, it is recommended to reserve a portion of the historical data for validation in order to assess the quality of the solutions found by the agent. Additionally, the validation period can also be used by the engineers to further filter out unwanted solutions based on their own criteria.

To examine and measure how Algorithm 1 scales with the number of available computing resources, a scalability test was conducted on the second data set. The scalability test was conducted using the second data set as it has a fewer number of uncertain parameters and a smaller initial objective function, making the problem easier to solve resulting in a shorter run-time. A reasonable run-time is necessary for the scalability test as it requires the experiments to be repeated tens of times for reproducibility purpose as discussed in the Methodology.

Figure 2-16 shows the capacity of Algorithm 1 to reduce the run-time when doubling the number of available computing resources. On average, every time the computing resources are doubled, a 41% reduction in run-time is achieved. When using 16 environments instead of one, Algorithm 1 reduced the total run-time from 18.6 minutes to 2.2 minutes, achieving an 88% reduction in the time needed to find one solution.

As shown in Figure 2-17, Algorithm 1 achieved a speed-up of 8.45 when using

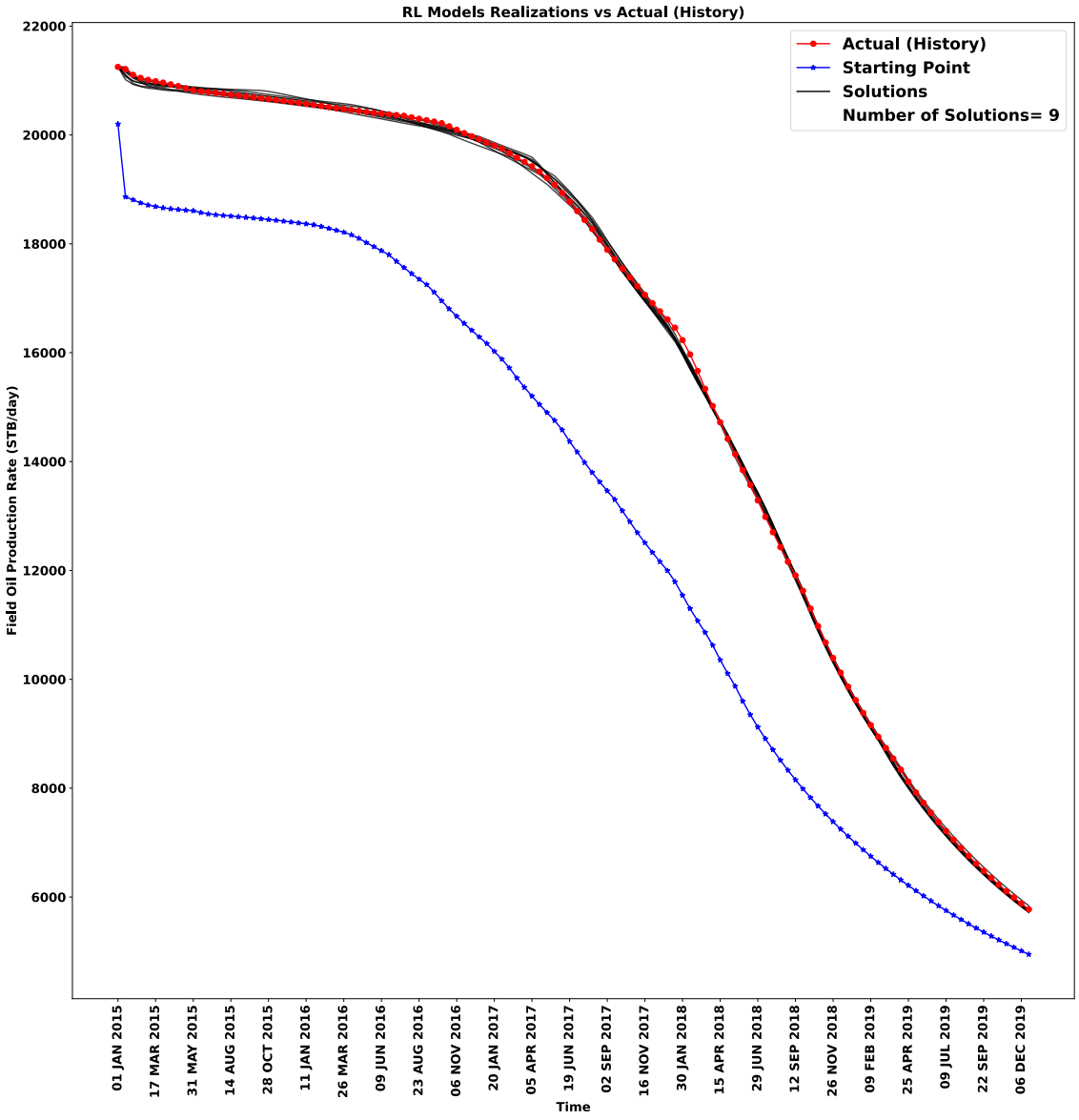

Figure 2-14: Algorithm 1 enabled the artificial deep neural network agent to learn from multiple environments simultaneously. Thanks to the stochastic policy used, 9 multiple and different solutions to the history matching problem are found to the history matching problem during the 20,000 time-step.

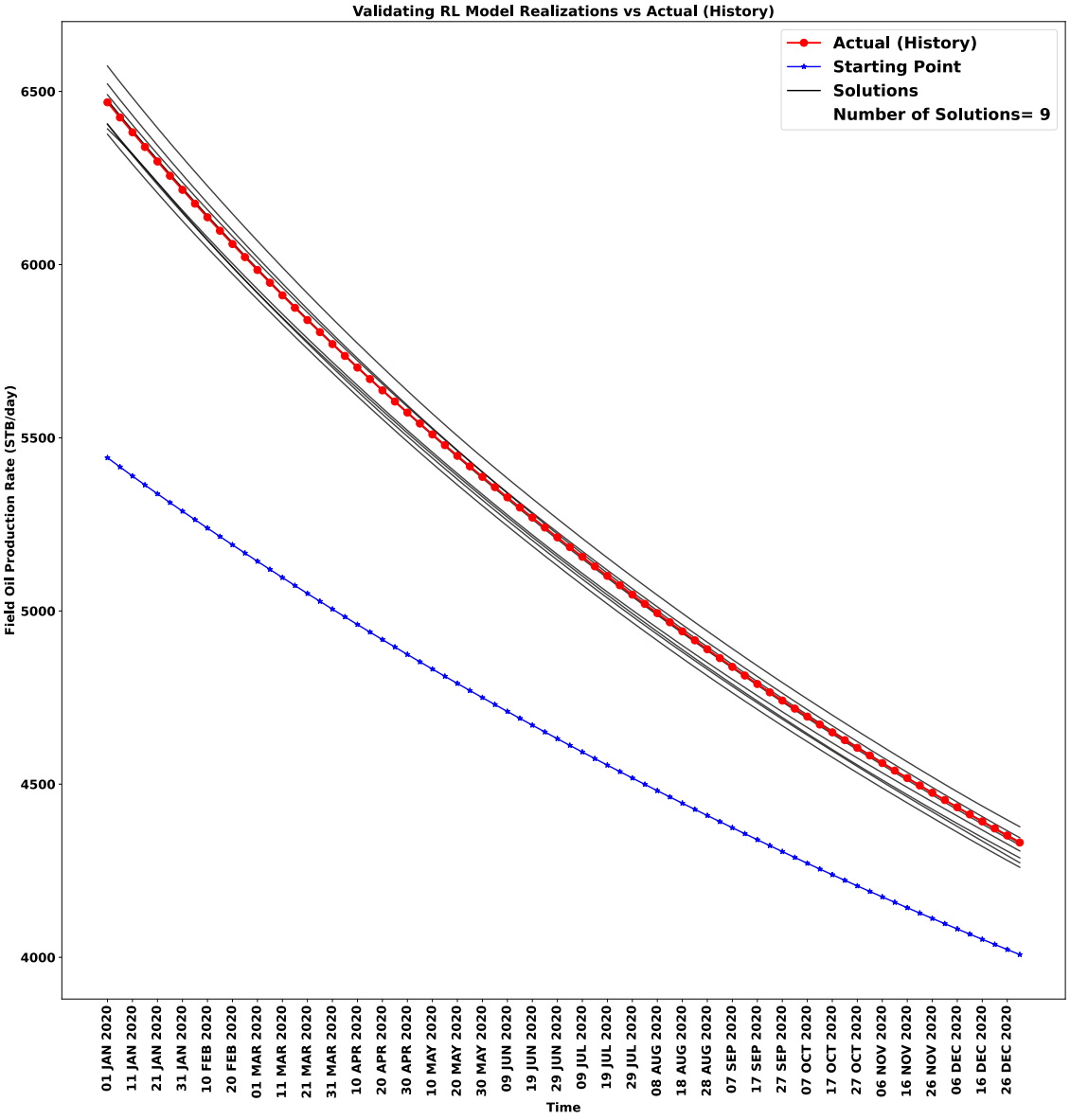

Figure 2-15: To test the forecasting capabilities of the realizations found by the artificial deep neural network agent, the realizations are validated by running them for one more year that the agent did not train on. The realizations found by the agent shows a good forecasting capabilities and provided multiple production scenarios that are within close range of the truth model.

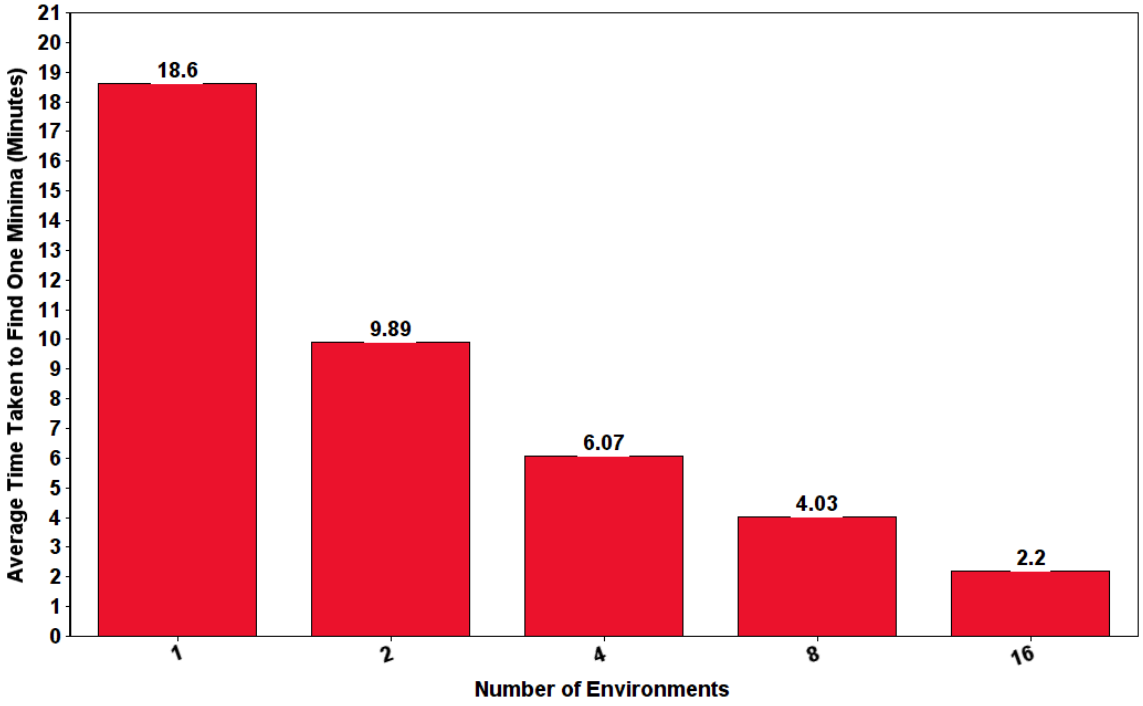

Figure 2-16: A significant reduction in run-time is achieved when computing resources are doubled. On average, a reduction of 41% in run-time is achieved when the computing resources are doubled resulting in total run-time reduction of 88% when using 16 environments instead of one. For reproducibility purpose, each column represents the average time taken across 10 runs to find one solution to the history matching problem.

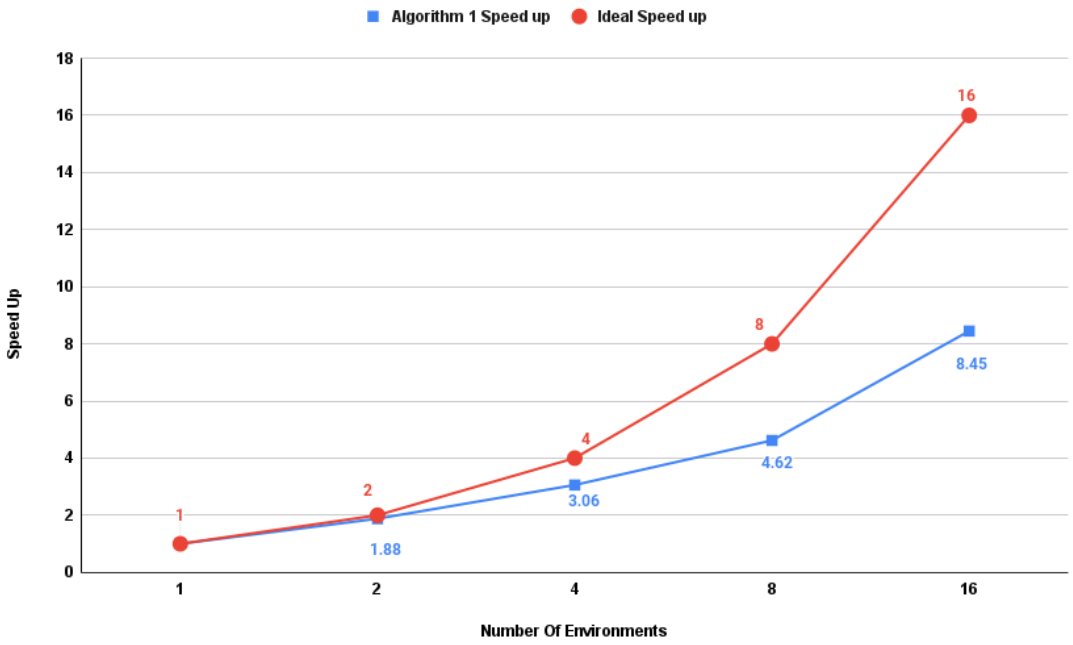

Figure 2-17: Based on the run-time values shown in Figure 2-16, Algorithm 1 scalability plot shows a significant speed-up when more computing resources are added. On average, a speed-up of 1.57 is achieved every time the computing resources are doubled.

16 environments compared to one environment. The algorithm scored an average speed-up of 1.57 when doubling the number of resources compared to a speed-up of 2 in the ideal scenario where the ideal scenario refers to the non-realistic case when an algorithm achieves a speed-up of 2 every time the resources are doubled. The algorithm achieved a maximum speed-up of 1.88 when running the algorithm on two environments instead of one, and a minimum speed-up of 1.51 when running the algorithm on eight environments instead of four.

A major advantage of the parallelization procedure employed by Algorithm 1 is the fact that the number of reinforcement learning time-steps (number of forward simulation runs) did not increase significantly while increasing the number of computing resources as shown in Figure 2-18. Such property indicates that the algorithm can scale efficiently when adding computing resources. As shown in the figure, it takes the algorithm around 2,000 time-steps to find a minima and instead of taking 18.6 minutes running on one environment, it only took 2.2 minutes when running on

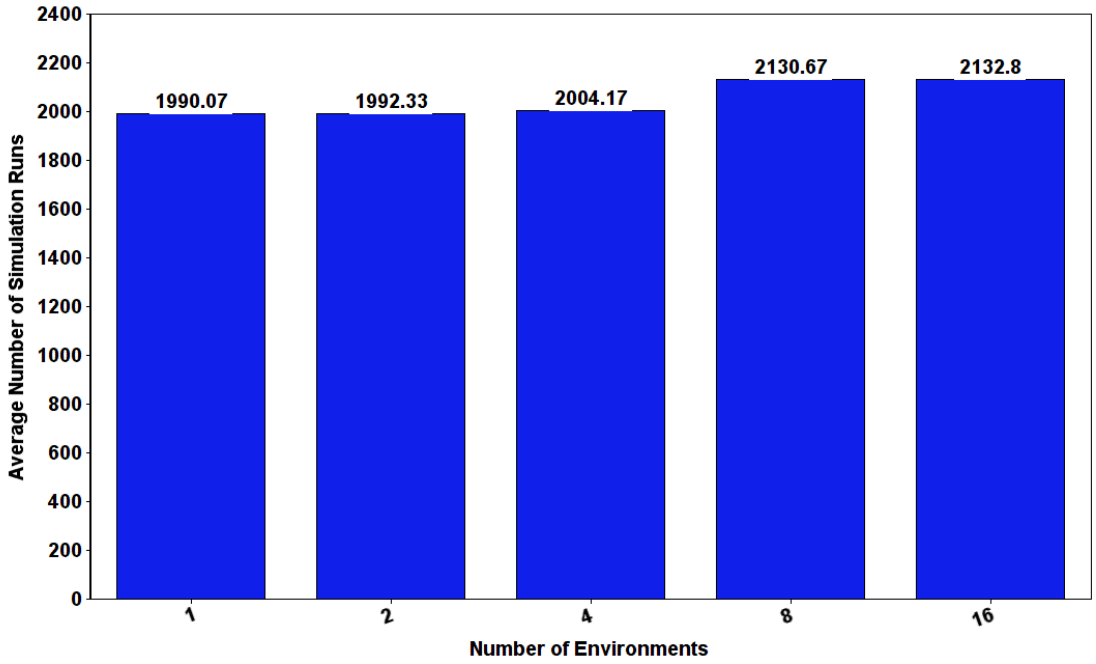

Figure 2-18: Based on the run-time values reported in Figure 2-16, Algorithm 1 had the capacity to efficiently scale when increasing the number of environments without significantly increasing the number of simulation runs.

16 environments while slightly increasing the number of simulation runs from around 1990 to around 2133 runs.

2.5 Analysis & Discussion

By reformulating the history matching problem from a mathematical least-square optimization problem into a Markov Decision Process problem, the suggested algorithm provided a way in which the history matching problem can be solved in parallel where adding more computing resources can speed up the convergence process. In addition, the suggested algorithm can be used to find multiple and different solutions to the history matching problem, allowing for better forecasting uncertainty analysis. By creating multiple environments and allowing the artificial neural network agent

to sample from these multiple environments simultaneously, the algorithm had the capacity to tune 27,000 uncertain parameters. Such capacity allowed the agent to find multiple and different historically matching models in a timely manner.

The multiple different solutions shown in Figure 2-14 does not have to necessarily be 9 solutions as more solutions can be found. As a matter of fact, there are infinite solutions to the problem (54) and if more solutions are desired to better assess the uncertainty, then the engineer running the experiment can run Algorithm 1 for a longer period of time. Giving the agent more time to learn from the environment will allow it to find more solutions. The process of training the agent for a longer time to find more solutions does not require the training process to be repeated from scratch. In order to avoid retraining, the agent's knowledge can be re-used by saving the deep neural network models to the disk and then re-loading them later to resume training if the number of solutions found is not sufficient.

The results also show that the agent learns how to interact with the environment in order to maximize its rewards, where in the beginning the algorithm will not be able to find any solutions until it learns how to map the current states of the environment into actions that reduce the objective function, eventually finding a solution. Naturally, the more complex the problem (i.e., the more uncertain parameter needs to be tuned) the longer the training needed. This is due to the fact that the agent will need more samples to explore and learn from the environment. In addition, the deep neural networks used to map the state into action will be larger as explained in the Methodology, thus needing more data to be able to successfully find multiple solutions to the history matching problem.

It is important to also note that as the number uncertain parameters increases, the training process becomes more difficult. Such difficulty is expected as the higher the number of uncertain parameters, the larger more complex the deep neural network needed to handle the problem. A larger network can cause instability in training and would require a smaller learning rate that slows down the algorithm. We found out that adding a regularization term to the objective function would not only help mitigate the ill-posedness effect on the problem, but it also can help stabilize the

learning process and allow for a larger learning rate to be used which can speed up the convergence of the algorithm.

Although the algorithm have successfully found solutions to the history matching problem by tuning 27,000 uncertain parameters on SPE9 reservoir model, it would be very challenging to tackle larger models such as SPE10 if the targeted uncertain parameters are permeability values in each cell. SPE10 reservoir model is a 3D model with 220 cells in the X direction, 60 in the Y direction and 85 in the Z direction resulting in a reservoir model with 1,122,000 cells. By choosing all three permeabilty values in each cell to be tuned, then the number of uncertain parameters would be 3,366,000.

This is mainly caused by the combinatorial impact of the solution space which grows exponentially with the number of parameters hindering the algorithm's ability to successfully maps states into good actions. For example, assuming at each time-step the algorithm will tune each uncertain parameter by the following actions [+1,0,-1] i.e., the algorithm would choose one of three options: either increase the permeability by one, decrease it by one or do nothing. This means that if we have 1,000 uncertain parameters, the algorithm would have 3^{1000} (approximately 10^{477}) different actions to take. To put it in perspective, this number is larger than the number of atoms in the observable universe which is estimated to be around 10^{80}.

For this reason, the algorithm will not explore the whole solution space as it is not mathematically feasible to do so. However, instead of exploring the whole solution space the algorithm will search for solutions that are within the vicinity of the starting point. Thus it is essential that the starting point of the algorithm present a good estimate of the truth model where it should be chosen based on the knowledge of the geologist reservoir structure.

It is important to note that the starting point also plays an important rule in terms the speed of convergence. Naturally, a model with a small initial objective function of 100,000 will converge faster than a model with an initial objective function of 500,000. This is an expected behavior that also occurs when using optimization algorithms to solve the problem where the closer the starting point to the solution, the faster the

convergence.

Faster convergence can also be achieved if the tolerance criteria is loosened as this is the case with other algorithms as well (55). Reducing the historical mismatch by more than 99% when tuning 27,000 uncertain parameters will always be difficult as it will require most algorithms to get more samples from the reservoir model parameters. As with other algorithms, a trade-off between accuracy and speed will also needs to be considered. Moreover, faster convergence may also be achieved if the maximum time-steps per episode parameter is set to a smaller value. However, this might hinder the algorithm's ability to explore the parameters space efficiently as sometimes the agent takes few bad actions on purpose in order to search for new solutions.

The run-time might be reduced by reducing the number of uncertain parameters using sensitivity analysis where the cells that have the least effect on the change of the objective function can be ignored and not included in the uncertain parameters vector just like when dealing with inactive cells. The number of uncertain parameters can also be reduced using permeability multipliers, where instead of tuning each single permeability value, a group of constant multipliers are tuned, where each multiplier is used to multiply the permeability values in certain cells. However, in both data sets used in this chapter, no multipliers are used in order to show the capacity of the algorithm to tune thousands of parameters. Another approach to reduce the run-time is the use of proxy models (15, 56, 57) where the reservoir simulator $f(u)$ is replaced by a fast approximate model $g(u)$ that can map the input of the reservoir simulator into an output within some acceptable accuracy.

A major advantage of using the suggested algorithm is its flexibilty in terms of the parameters space. Where unlike Ensemble Kalman Filter (EnKF), it does not require the parameters space to have a Gaussian distribution. Another advantage is its versatility, where some optimization algorithms might diverge away from the solution if the initial objective function is large, this algorithm is less prone to divergence with an initial guess that is reasonably far from any solution if given enough time. This is mainly caused by the nature of this algorithm where it would restart itself to the starting point upon divergence. Then, thanks to the shaping of the reward

function the agent will be punished severely for diverging. By punishing the agent for diverging away from the solution, it will reduce the probability of choosing these actions in the future to enable the agent to learn from such mistakes and not repeat the same actions again that lead to divergence.

It is important to note that when using a very large number of computing resources across hundreds of environments, the algorithm scalability will be lower compared to the case when utilizing tens of environments . This is due to the fact that the batch size (1024, 512, etc.) may not be big enough to divide the work efficiently amongst the environments. When this occurs, the batch size would have to be fixed when increasing the number of environments causing PPO to be less sample efficient as resources increase.

In that case, a good speed-up is still expected to be achieved when adding more computing resources but Algorithm 1 might not scale very well compared to its scaling performance when using tens of environments. When dealing with a very large model that requires hundreds of environments, engineers will have to spend some time adjusting the hyperparameters and ensuring efficient workload for each environment in order to achieve high scalability.

One of the major challenges arising from using Algorithm 1 is the number of hyperparameters related to each experiment. Where in addition to the usual deep neural networks hyperparameters (batch size, learning rate, number of neurons, etc.) and reinforcement learning hyperparameters (γ, maximum time-steps, etc.), the experiments itself has its own hyperparameters, such as K_Δ and maximum time-steps per episode as these parameters are problem-specific that depend on the reservoir model at hand. The engineer using the algorithm must rely on his knowledge of the reservoir model to find proper hyperparameters to help guide the algorithm into convergence. In addition, the engineer might have to spend some time trying to find optimal parameters that can make the algorithm converge faster.

The vector containing all uncertain parameters u does not have to only contain permeability values. Any uncertain parameters, for example the porosity ϕ, can also be included in u. However, the engineer running the experiment must also adjust the

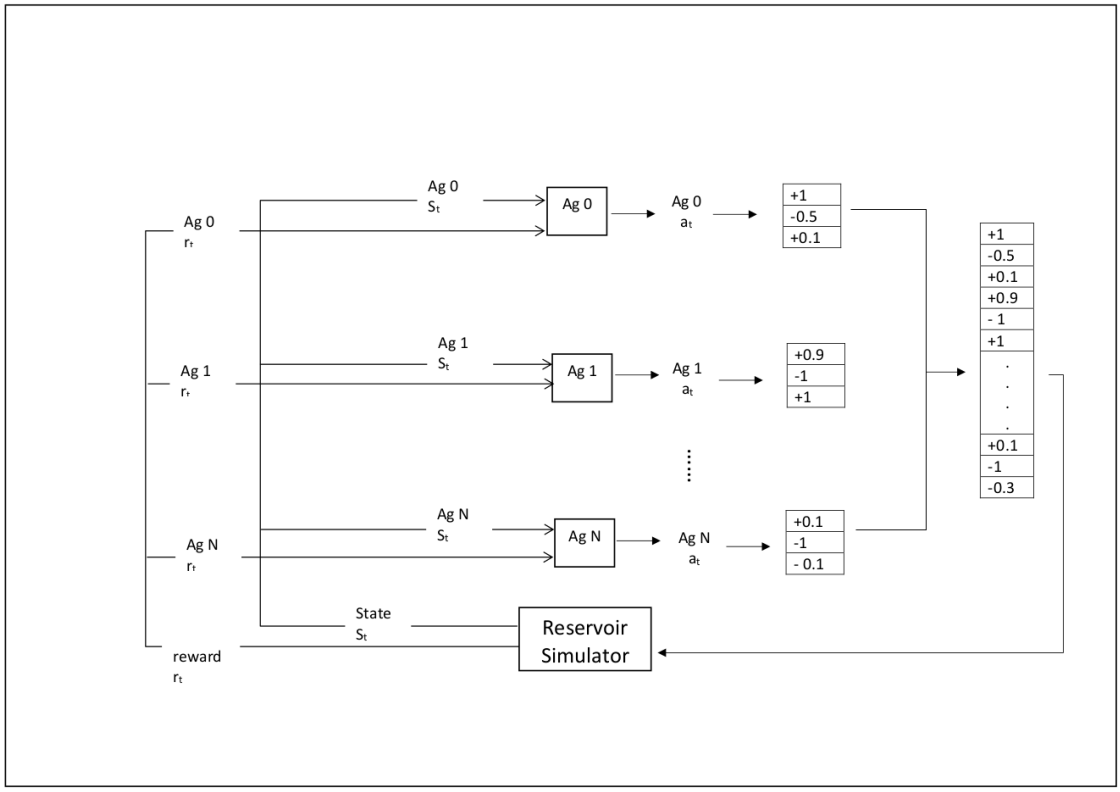

Figure 2-19: A multi-agent approach to solve the history matching problem. Using multiple collaborating artificially intelligent agents may provide a mechanism to handle more complex problems.

action space $\mathcal{A}$ to take into account the scale of change between different properties at each reinforcement learning time-step, where the maximum change allowed for the permeability value K_Δ might differ than the maximum change for porosity ϕ_Δ.

A future research opportunity can be explored by extending this framework using Multi-Agent Reinforcement Learning (MARL) (58, 59, 60) instead of using a single agent across multiple environments. As shown in Figure 2-19, the multiple agents can work collaboratively towards reducing the objective function where each agent is responsible for a section of the reservoir model. Each agent can observe the entire environment or just a part of it to take actions. Such approach can allow researchers to tackle very large models because each agent would have to map an easier function with a smaller number of actions. The small number of actions per agent would result in smaller deep neural networks, requiring less training time.

However, such an approach comes with its own difficulties as well, where it would

be difficult to come up with an appropriate reward function. The most difficult part would be figuring out a way to reward good actions and punish bad ones, where at each time-step it would be difficult to identify which agents took good actions and which agents took bad ones from a single reward function.

Applying the algorithm in multiple rounds can be used in order to achieve tolerance with a tighter tolerance criteria. The approach may be used to speed up the process when the tolerance criteria and help the algorithm to start in the later rounds from a point that is closer to the tolerance criteria. However, changing the starting point will require the algorithm to restart training process the for the agent due to the fact that the previous knowledge will not applicable to the new starting point i.e., the paths or trajectories that the agent learned to follow in order to minimize the objective function will not be the same when using a different starting point.

Restarting the training process at every round will slow the agent learning process especially at the beginning of the round. However, such slowness might be compensated by saving more time by starting from a closer point and might be worth the computing time spent in order to find better models with a tighter condition for tolerance. Additionally, the knowledge gained by the agent in the previous round might also be used to help the agents' learning in the next round by utilizing transfer learning. A flowchart of the approach is shown in Figure 2-20.

It is important to note that changing the starting point multiple times will limit the exploration process as the agent might only find solutions that are very similar. This is caused by the fact the agent will search for solutions that are within close approximate to the first round best model.

2.6 Conclusions

The results drawn from the research conducted in this chapter show that suggested parallel automatic history matching algorithm using reinforcement learning (Algorithm 1) can train an artificial deep neural network agent that is capable of finding multiple different solutions to the history matching problem, even when dealing with

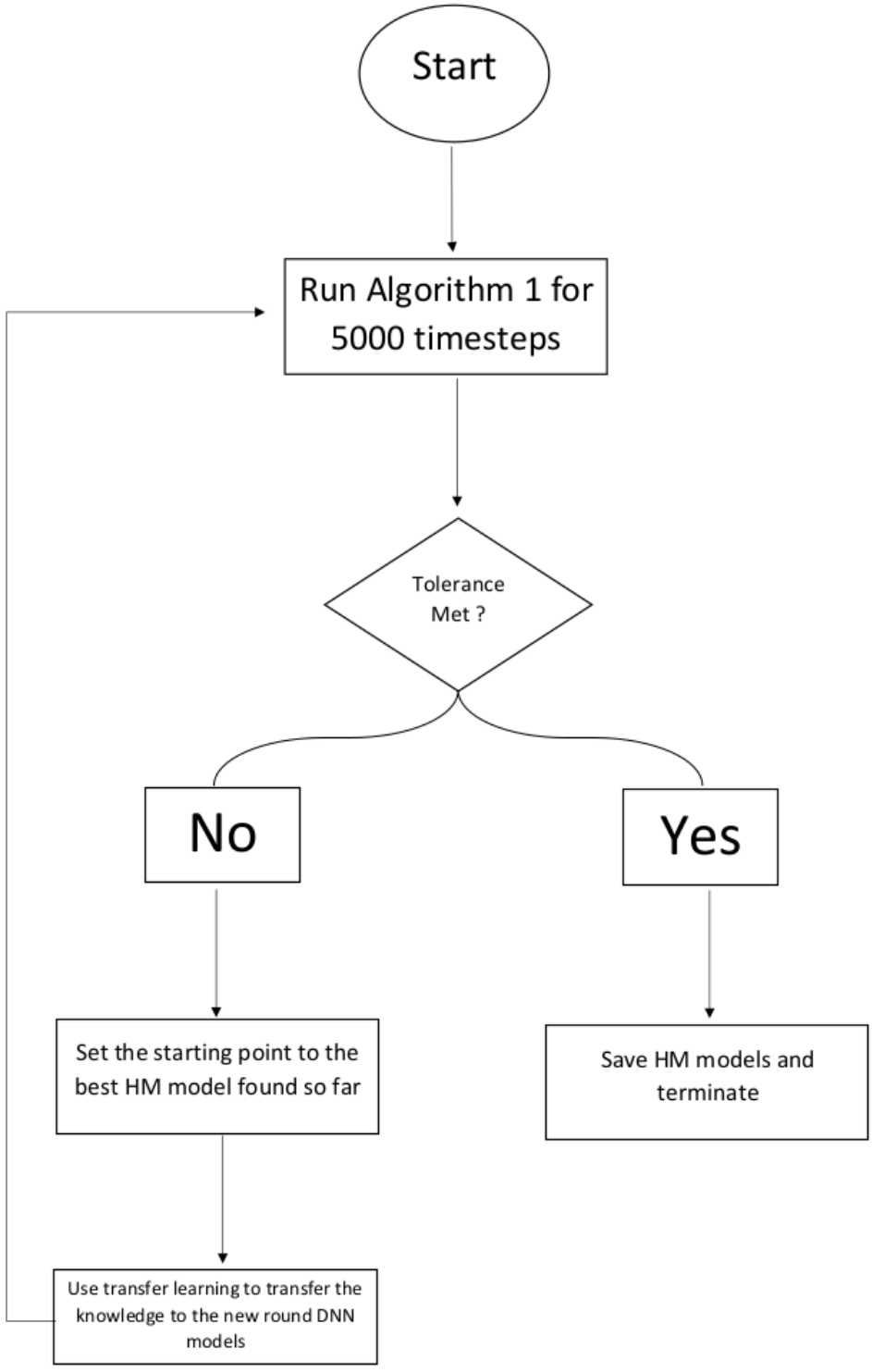

Figure 2-20: A flowchart illustrating the process of applying Algorithm 1 on multiple rounds in order to archive tolerance with tighter tolerance criteria. The use of transfer learning may improve the learning process for future rounds.

tens of thousands of uncertain parameters. By reformulating the problem from an optimization problem into a Markov Decision Process, the algorithm gave the chance to sample data from multiple trajectories across multiple environments, allowing the agent to learn faster as more computing resources are added.

As shown from the Results, the algorithm achieved an average speed-up of 1.57 when the computing resources are doubled and had the capacity to reduce the run-time needed to find one solution by 88% from 18.6 minutes to 2.2 minutes, when using 16 environments instead of one. Such parallelization gave the opportunity to tackle complex problems and find multiple solutions in a timely manner when tuning 27,000 uncertain parameters.

2.7 Nomenclature

a_t : Reinforcement learning action taken by an agent at time-step t.

$\mathcal{A}$: Set of all possible actions that can be taken by the agent.

Ag : artificial deep neural network agent.

B: Reinforcement learning batch size.

$f(u)$: Function representing the reservoir simulator.

$g(u)$: Function representing the proxy model.

K_x,K_y and K_z : Permeability in the x,y and z directions.

K_Δ: Permeability limits of each action taken.

m : Number of time-steps in the simulation model.

n : Number of wells in the reservoir model.

N: Number of environments.

q : Actual pressure or saturation from historical data.

$\hat{q}$: Simulated rate from the simulator or proxy model.

r_t : The reward obtained from the environment at time-step t.

s_t : Current state of the reinforcement learning environment at time-step t.

$\mathcal{S}$: Set of all possible states in the reinforcement learning environment.

t : Reinforcement learning time-step iterator.

u : Vector containing all uncertain parameters in the model.

X,Y and Z: Number of cells in the x,y and z direction in the reservoir model.

ϵ : Error tolerance level accepted.

γ: Reinforcement learning discount factor.

π_θ: PPO Actor Network.

Chapter 3

Towards better shale gas production forecasting using transfer learning

3.1 Background

Thanks to hydraulic fracturing technology, extracting oil and gas in an unconventional shale formation became economically viable making the United States the top oil and gas producer in the world in 2020 according to the US Energy Information Administration (EIA) (61). Accurate production forecasting is necessary for many reasons such as decision making, developing returns on investments, and maintaining and managing the wells. As natural gas consumption is expected to grow in the near future increasing the demand for such resource (62), developing accurate production forecasting models plays an important role in helping policy makers better assess the supply and demand situation and allocate resources based on better understanding of the matter.

The most accurate method for oil and gas production forecasting is reservoir simulation. However, current reservoir simulation techniques that are used in forecasting conventional oil and gas wells still face some challenges when it comes to forecasting unconventional resources due to challenges in modeling (63), costly data acquisition (64), complex fracture network (65) and variation of fracture permeability (66). Another approach used to forecast unconventional gas resources is decline curve analysis

(DCA) in which different models have been developed and used over the years (67, 68). Some of these decline curves were developed from physical interpretation of gas flow (69, 70) and some of them are developed empirically based on observations (71, 72).

One of the most common ones that is also used by EIA is the hyperbolic function known as the Arps model (73). Arps decline curves are cheap,fast and easy to implement, and the model can generate forecasts with a minimal amount of data. However, it is a heuristic approach based on empirical observation and does not generate accurate predictions for unconventional gas wells as it tends to overestimate the production after the early stages of a well's production (74). Montgomery et al. (75) show that DCA shale gas forecasting is an ill-posed problem due to non-unique model parameter estimates especially when the production history is short.

The availability of historical data along with the recent advancement in machine learning granted a new method for forecasting production by using a deep neural network (DNN) that helps generate a more accurate production forecast. Al-Fattah and Startzman (76) developed a DNN to forecast the US cumulative natural gas production using different physical and economic input parameters such as the gas annual depletion rate and growth rate gross domestic product (GDP). DNNs also have been used in order to improve decline curves where Li and Han (77) used a DNN to get an estimation of parameters in a logistic decline curve model. Sun et al. (78) used recurrent neural networks to build time series models that forecast the production using the well's production history and the tubing head pressure as input.

Montgomery et al. (75) developed a DNN forecasting model that used data aggregated from multiple counties in Texas Barnett shale to forecast production in which the first few months of actual production data is used to forecast the production for the next few years. This research detailed in this chapter uses a similar approach in which transfer learning is used to build county-specific DNN models by utilizing the knowledge acquired from other nearby counties. In general, the DNN can be expressed as a function $\mathbb{F}$ that is theoretically able to learn the best non-linear mapping from the input data, which are the first few months of production, to the output data which are the next few years of production (75). A DNN can be trained by dividing

the well production data into input data and label (output) data, the input data are the first 4,6,8 or 10 cumulative monthly production while the label data are the remaining cumulative monthly production data. The function $\mathbb{F}$ is then trained to map a specific well's first few months of cumulative production into a forecast of its future production.

This research conducted in this chapter presents a novel approach in which transfer learning DNN models can reduce the error in forecasting between 11% and 47% compared to Arps decline curve models. This chapter also demonstrates that by utilizing transfer learning to apply the knowledge acquired from production data in nearby counties into counties with limited data, this approach helps mitigate a major disadvantage that machine learning models have which is the need for a large amount of data to develop a good predictive model. In addition, the research conducted in this chapter also shows how transfer learning can still be useful in generating accurate production forecasts even when the data available for a certain county is not enough to properly train a transfer learning model.

This chapter starts first by going over the details data sets used for this study in Section 3.2. Then, in Section 3.3, it explains the methodology in which the Arps decline curve model is used as a benchmark for the suggested new approach and the details of the comparison procedure between the two models. Next, in Section 3.3 and 3.4, this chapter shows that DNNs can map input production data into output future production forecast and hence can be used effectively to develop shale gas production forecasts. After that, Section 3.4 shows how the technique of transfer learning is utilized to overcome data availability that limits DNNs from developing county-specific models. Lastly, Section 3.5 offers a thorough discussion and analysis of the results and its potential for further improvements in the future.

3.2 Data

Two data sets were used in this chapter. The first data set is composed of 4439 wells from Texas Barnett shale with 120 months worth of cumulative monthly production

Sample wells from Texas Barnett shale					
Well-API	County	State	Production (Mscf), Month:1	...	Production (Mscf), Month:120
42-367-34721	Parker	Texas	22660.3	...	2302.3
42-425-30160	Somervell	Texas	21295	...	1373.6

Table 3.1: Sample wells from Texas Barnett shale

Sample wells from Pennsylvania Marcellus shale					
Well-API	County	State	Production (Mscf), Month:1	...	Production (Mscf), Month:66
059-26033	Greene	Pennsylvania	87601	...	18529
115-21351	Susquehanna	Pennsylvania	96286.8	...	31708.6

Table 3.2: Sample wells from Pennsylvania Marcellus shale

data (79). The second data set is composed of 2172 gas wells from the Marcellus shale in Pennsylvania with 66 months worth of cumulative monthly production data (80). The cumulative production in both data sets is measured in thousands of standard cubic feet (Mscf). In addition, both data sets contain which county the well is located in. However, both data sets lack geological formation physical properties as well as well-specific physical properties (fracking volume, perforated length, etc). Table 3.1 and Table 3.2 provide sample wells from both data sets where the Well-API refers to well-specific unique identifier.

3.3 Methodology

3.3.1 Arps Forecasting Model

To measure how accurate the forecasts of the suggested DNN models are, this chapter compares the DNN models to the Arps decline curve model that is widely used by the industry and currently used by EIA (61, 81). Although the Arps model is an

empirical model that lacks physical basis compared to other forecasting models such as Rate Transient Analysis (RTA) (82), it is still widely used due to its simplicity and low data requirements. Due to the lack of physical data needed for RTA, this chapter uses the Arps model as a benchmark. The Arps model was first introduced in 1945 by Arps (73) and it empirically forecasts the decline in production of a specific well using historical data fit into a three-parameter model (83) as shown in Equation 3.1:

$$Qt = \frac{Qi}{(1 + b * Di * t)^{1/b}} \tag{3.1}$$

where Qt refers to the production at month t, Qi is the production rate at time 0, b is the line curvature degree, t is the months in production and Di is the initial decline rate. The Statistical and Analytical Agency in the US Department of Energy (81) provides county-specific parameters by averaging the production of all wells in the county and then fitting the average production to the decline curve using Equation 3.1.

However, the Arps model uses only Qi (first month of production) as input to generate predictions, while the DNN utilizes the first 4,6,8 or 10 months of production data. In order to provide a fair comparison between the two models, the Levenberg–Marquardt algorithm (84, 85, 86) (a non-linear least square fit) is used to find optimal b, Qi and Di parameters that give the best fit between the Arps model and the input data (first few months of production) as this approach is usually followed to utilize the first known months of actual production data. Simply put, for each well, the actual first 4,6,8 or 10 months of production data will be used to tune the Arps parameters so it would generate a better forecast. This procedure provides a fair comparison between the DNN and the Arps model as it allows both models to utilize exactly the same data given as input.

3.3.2 Deep Neural Network Forecasting

A traditional supervised learning technique was used to train and test a DNN denoted as a mathematical function $\mathbb{F}$ to generate a predictive relationship between the input and output (87), by mapping the input vector $X = \{x_1, x_2..., x_n\}$ to an output vector $Y = \{y_1, y_2, ..., y_m\}$ where n is the number of months used as an input to the DNN, x_1 is the first cumulative monthly production, y_1 denotes the first cumulative monthly production forecast, and y_m denotes the last month production forecast. Figure 3-1 illustrates how the DNN maps the input X, which are the first few months of production data, to Y which represents the production forecast.

The DNN model developed for this chapter is composed of a four-layer deep sequential neural network with an input dense layer of 30 neurons, two hidden dense layers with 35 and 50 neurons, and an output layer with a number of neurons equal to the desired number of output months as shown in Figure 3-1. All the neurons in the network used a rectified linear unit (ReLU) as an activation function. In addition, three dropout layers with a value of 0.1 were used. The dropout layers were added after each of the first three layers to reduce the effects of overfitting (88). The model was trained with the Adam stochastic optimizer (89) minimizing the mean absolute error (MAE) of the loss between the predictions and the labels.

This chapter employs MAE as a loss function instead of Mean Square Error (MSE) due to the fact that MSE penalizes large errors more than small errors. This property would put more weight on the data outliers making them more important to the loss function. Due to the fact that the data is volatile especially in the first few months of production, MAE was chosen instead of MSE in order to avoid assigning larger weights to the outliers and generate a better fit for the decline in production. This model configuration was chosen as it resulted in the least testing error after extensive testing for all of the DNN parameters including all available activation functions and optimizers. In addition, the input and output data points (number of months) of the DNN are flexible and can be chosen during training time in order to see how the DNN will behave given a certain number of input months.

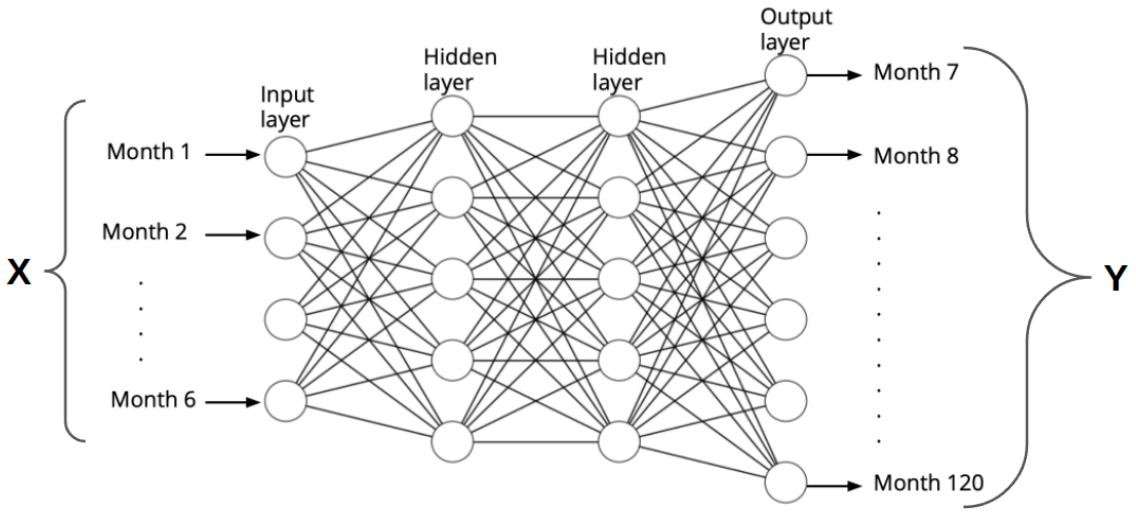

Figure 3-1: Layers used by the DNN model, which consists of an input layer with 30 neurons, two hidden layers with 35 and 50 neurons, and an output layer with a number of neurons equal to the desired number of output months. The first three layers are followed by a dropout layer with a value of 0.1 to reduce training data overfitting.

3.3.3 Transfer Learning Models

DNNs are powerful tools and have been utilized heavily to tackle research problems, but one of their major drawbacks is that DNNs need an abundance of data to generate accurate predictions and in shale gas forecasting that can be a problem especially for counties with limited data. Transfer learning, which is a machine learning technique that can effectively use the knowledge from a pre-trained model to make predictions on a new set of data from a related problem (90), can be utilized to overcome the limitation of the available data. Transfer learning can be very useful in the case where the data set from the new problem is not large enough to properly train a DNN.

The formal definition of transfer learning in the context of machine learning as defined by Yang et al. (91) is: "Given a source domain D_S and learning task T_S , a target domain D_T and learning task T_T , transfer learning aims to help improve the learning of the target predictive function $\mathbb{F}_T$ in D_T using the knowledge in D_S and T_S , where $D_S \neq D_T$, or $T_S \neq T_T$". The goal is to use the knowledge stored

in $\mathbb{F}_S$ acquired from the pre-trained model that had enough data for proper training and testing, to assist the new models' predictive function $\mathbb{F}_T$ in making accurate predictions despite data scarcity (92) .

For example, a marine biology research team wants to develop a DNN that can classify images of sharks and dolphins but does not have enough data (images) to train the DNN to generate accurate predictions. The team can use transfer learning by utilizing the VGG16 (93) model as a source model $\mathbb{F}_S$ to help increase the accuracy of the predictions in their own target model $\mathbb{F}_T$. The VGG16 model is a convolutional DNN model that achieves more than 90% accuracy in ImageNet (94) which contains more than 10 million labeled images. Although the VGG16 may not necessarily have been developed to classify images of sharks and dolphins, utilizing its ability of feature extraction (95) can help generate a more accurate classifier for the sharks and dolphins pictures.

In the research work detailed in this chapter, transfer learning is implemented by taking the pre-trained model $\mathbb{F}_S$ and removing its output layer that makes the predictions and only keeping prior layers (knowledge transfer layers) as shown in Figure 3-2 then designing the new model $\mathbb{F}_T$ by taking the knowledge transfer layers and adding a new untrained output layer to be trained only on the new data. However, it is very important that the knowledge transfer layers are not trained while training $\mathbb{F}_T$ to preserve the knowledge that they have from training the source model $\mathbb{F}_S$. In this chapter, transfer learning will be used to develop county-specific DNN models in order to compare the results to the Arps model benchmark since the Arps models are county-specific models as described by EIA (81). Transfer learning will be used to overcome the need for a large data set to properly train a DNN since most of these counties do not have enough samples (wells).

These county-specific DNN models are developed in the following manner, for each specific county in the state's data set a source model $\mathbb{F}_S$ was developed by training it on all of the available data except for that specific county. Then, using transfer learning, a target model $\mathbb{F}_T$ was trained and tested only on that county's data. After that, in order to compare the forecasting quality, the MAE of both forecasting models

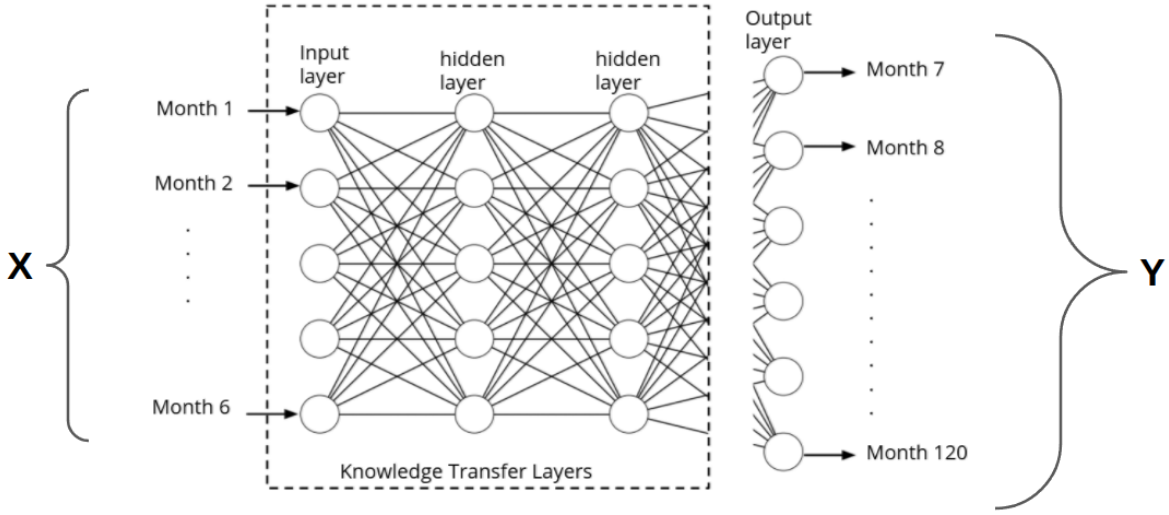

Figure 3-2: In this chapter, transfer learning is implemented by extracting the knowledge transfer layer from the source model $\mathbb{F}_S$ and using it along with a new untrained output layer to develop a target model $\mathbb{F}_T$. While training the transfer learning target model $\mathbb{F}_T$, the knowledge transfer layers must not be trained to preserve the knowledge that they have from training the source model $\mathbb{F}_S$.

are computed against the actual production for each individual well. Then, the mean error across all the sample wells in the testing set is reported as the forecasting error.

3.3.4 Training and Testing Procedure

For the DNN forecasting models, 75% of the data is used for training while the remaining 25% is used for testing. Due to the fact that the testing set is only used when the training of the model is complete, a small portion (10%) of the training data is reserved for validation to monitor the error of the model during training where it helps the training algorithm perform better by providing it feedback to tune its hyperparameters before the learning process is finished (96). The DNN models are initially set to be trained for 200 epochs where in each single epoch the DNN model goes over the entire training data set and updates its weight accordingly using the back propagation algorithm (97).

However, more training epochs than needed can cause overfitting because the optimization algorithm will over-optimize the weights of the model based only on the

training data. This results in a model that does not generalize well and will produce a high error during testing (98). For this reason, an auto-stopping mechanism is used to monitor the validation score and if the validation score has not improved in the last 10 epochs then the training will automatically be stopped and the model with least error on the validation set will be considered as the forecasting model. The auto-stopping mechanism would prevent the model from overfitting or underfitting the data, save computing power and reduce unnecessary training time (99).

3.3.5 Reproducibility and Verification

In DNN and in machine learning in general, one of the main challenges is reproducibility (100) which is difficult to achieve due to the random initialization of the weights prior training in addition to many other factors such as using a stochastic optimization algorithm. The problem complicates the efforts of quantifying the improvement of the suggested enhanced forecasting models due to the fact that each time a model is trained and tested it generates slightly different results. To make sure that the improvement in the forecasts was not caused by such randomness, the process of initializing, compiling, training, and testing of each model were repeated 100 times and the average of those 100 runs was reported as the final result.

Furthermore, to take into account the possibility that the DNN models outperform the Arps model only on a specific testing set (i.e, Arps may perform similar or better if the testing set is changed), at each run before the data is split into training and testing sets the data is randomly shuffled. However, it is important to mention that although the data is shuffled at the beginning of each run, the measurement of the forecasting error in terms of MAE across all models is done on exactly the same testing data set for that run. Figure 3-3 explains the flowchart of the comparison process between the models of interest.

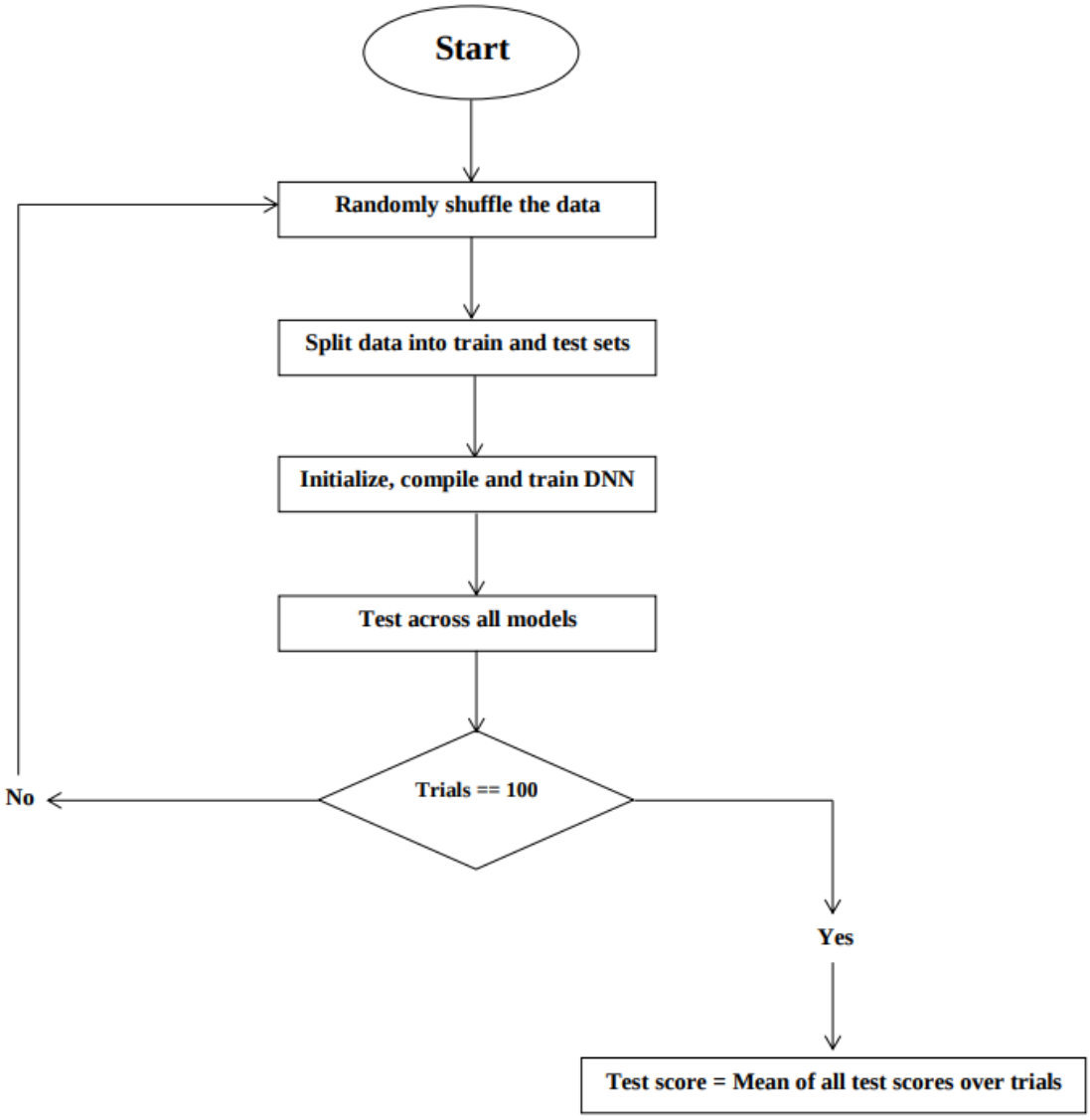

Figure 3-3: A flowchart showing the process of comparing the two forecasting models over the course of 100 trials to address both the reproducibility concerns caused by the randomness in the DNN and to ensure a fair comparison between the models of interest across multiple test sets.

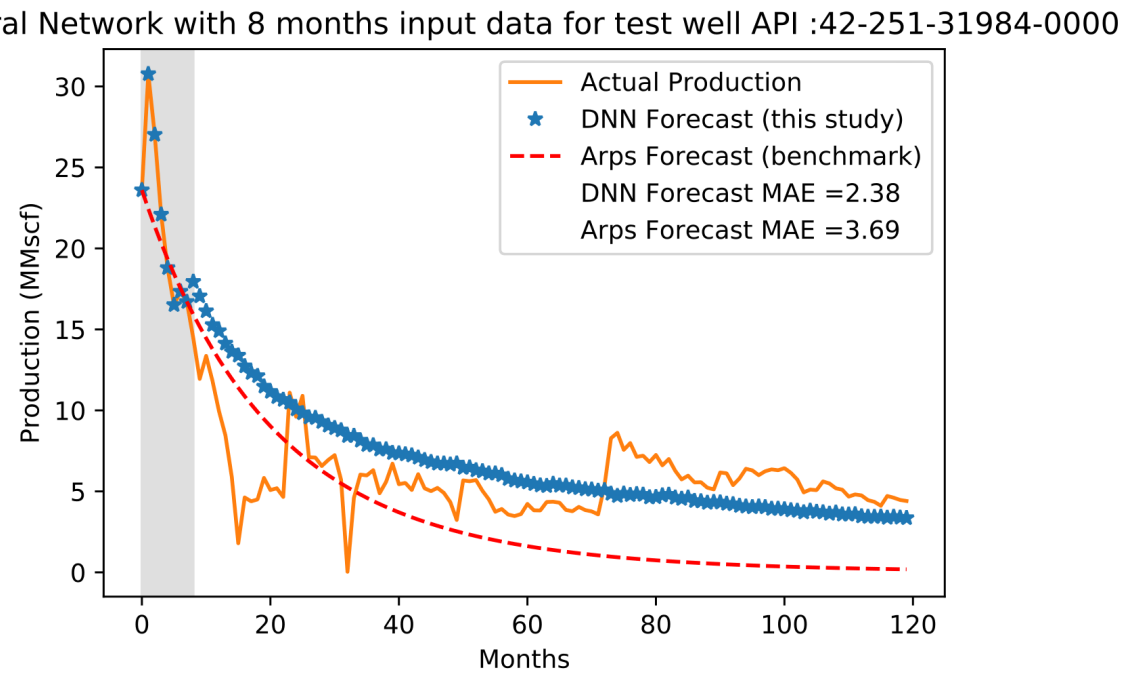

Figure 3-4: A sample test well from the Barnett shale shows the DNN model and the Arps model forecasting compared to the actual production data where 8 months of input data is used to forecast the production. The plot shows that with enough data, the DNN model can generate a more accurate production forecast than the Arps model. The gray area on the left of the plot indicates the data points used as input to the forecasting models while the rest of the plot shows the production forecast.

3.4 Results

3.4.1 Deep Neural Networks

As described in Section 3.3.2, two DNN models were trained and tested where the first DNN model was developed on the Barnett data set and the other one on the Marcellus data set. A sample well production forecast is shown in Figure 3-4 and Figure 3-5 where the grey area on the left of the plot indicates the data points (number of input months) used as an input to generate the forecast while the rest of the plot shows the models' predictions. To measure the accuracy of the forecasting models, each model is compared to the actual production in terms of MAE.

Both figures show that DNN can generate more accurate production forecasts than the Arps model. The DNN models were able to achieve more than 35% error reduction

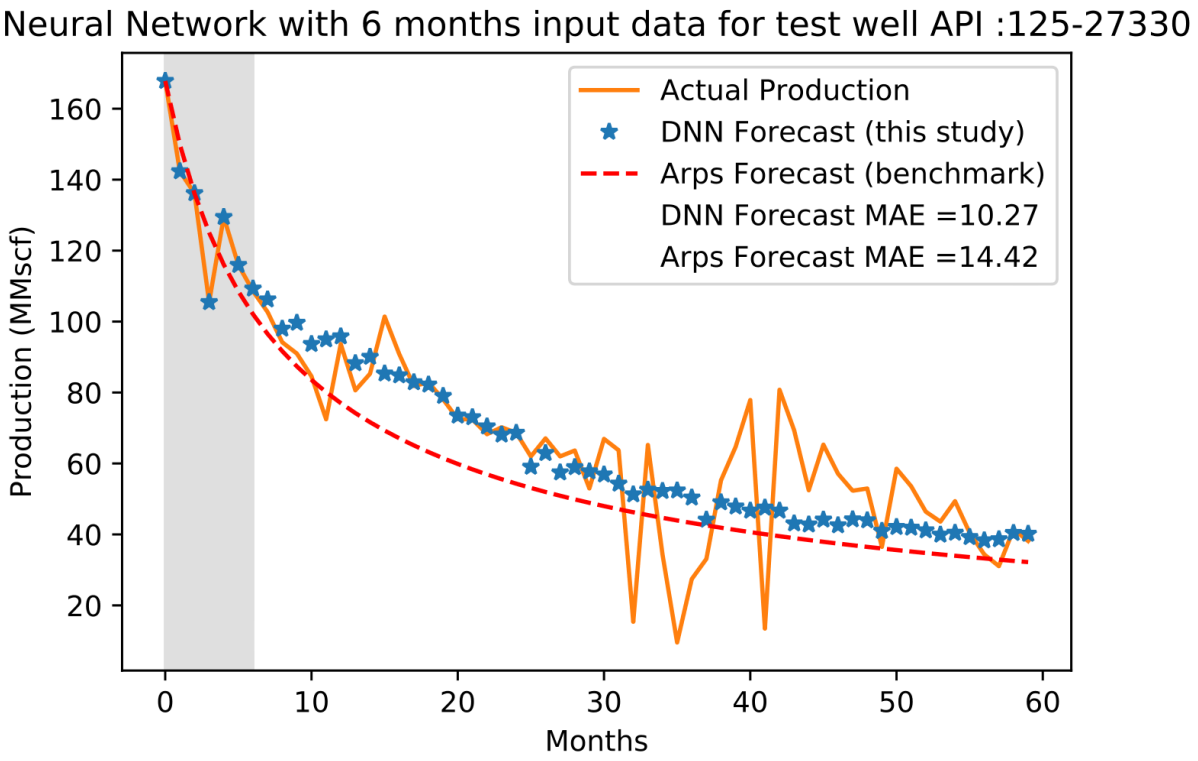

Figure 3-5: A sample test well from the Marcellus shale shows the DNN model and the Arps model forecasts compared to the actual production data where 6 months of input data is used to forecast the production. The plot shows that with enough data, the DNN model can generate a more accurate production forecast than the Arps model. The gray area on the left of the plot indicates the data points used as input to the forecasting models while the rest of the plot shows the production forecast.

Number of wells per county in the Barnett data set											
Johnson	Tarrant	Denton	Parker	Wise	Hood	Hill	Erath	Jack	Palo Pinto	Ellis	Somervell
1372	1050	620	469	425	272	131	36	24	16	12	12

Table 3.3: Number of wells per county in the Barnett data set

Number of wells per county in the Marcellus data set				
Susquehanna	Greene	Wyoming	Washington	Westmoreland
658	535	94	703	182

Table 3.4: Number of wells per county in the Marcellus data set

on the Barnett sample well and more than 28% error reduction on the Marcellus sample well. These results show that DNN models have the ability to map input data into useful output and generate accurate production forecasts. However, the two DNN models are statewide-level models and the Arps models are county-specific models so in order to fairly compare the results across all the wells in the testing set we need to compare DNN county-specific models to the current Arps county-specific model benchmark.

3.4.2 Transfer Learning

As described in Section 3.3.3, this chapter employed the technique of transfer learning to develop a better shale gas production forecast than the Arps benchmark model. Tables 3.3 and 3.4 show that in both data sets most counties do not have enough samples to properly train a DNN.

After following the procedure described in the Methodology Section 3.3.3 and applying it on both the Barnett and the Marcellus data sets, the DNN transfer learning models were able to reduce the error significantly. Figure 3-6 shows that when this method is used with 6 months of input data on the Barnett shale data set, DNN transfer learning models were able to reduce the error between 47% in Hill County and 30% in Wise County when compared to the Arps benchmark model. Similar results as shown in Figure 3-7 is obtained from applying the same method on the Marcellus shale data set where an error reduction between 34% in Susquehanna County and 11% in Westmoreland County is achieved compared to Arps.

However, in the Barnett data set the counties of Somervell, Ellis, Palo Pinto, Jack

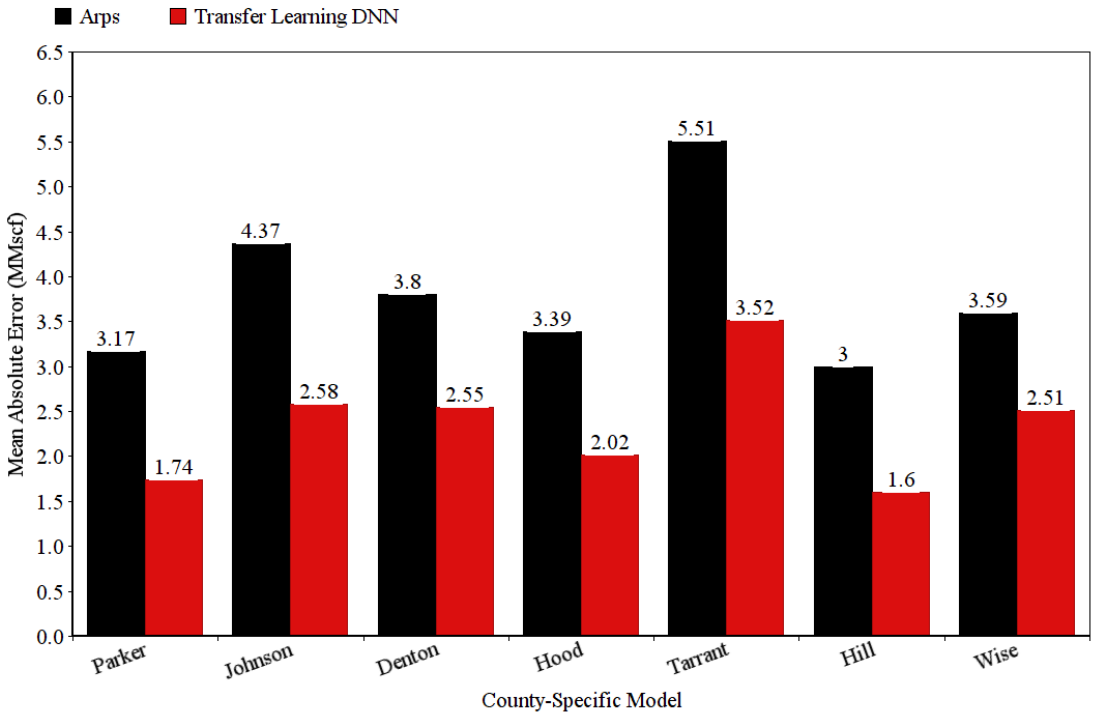

Figure 3-6: The new suggested county-specific DNN models achieved an error reduction between 47% in Hill County and 30% in Wise County compared to the Arps model benchmark when 6 months of input data were used to generate both forecasts.

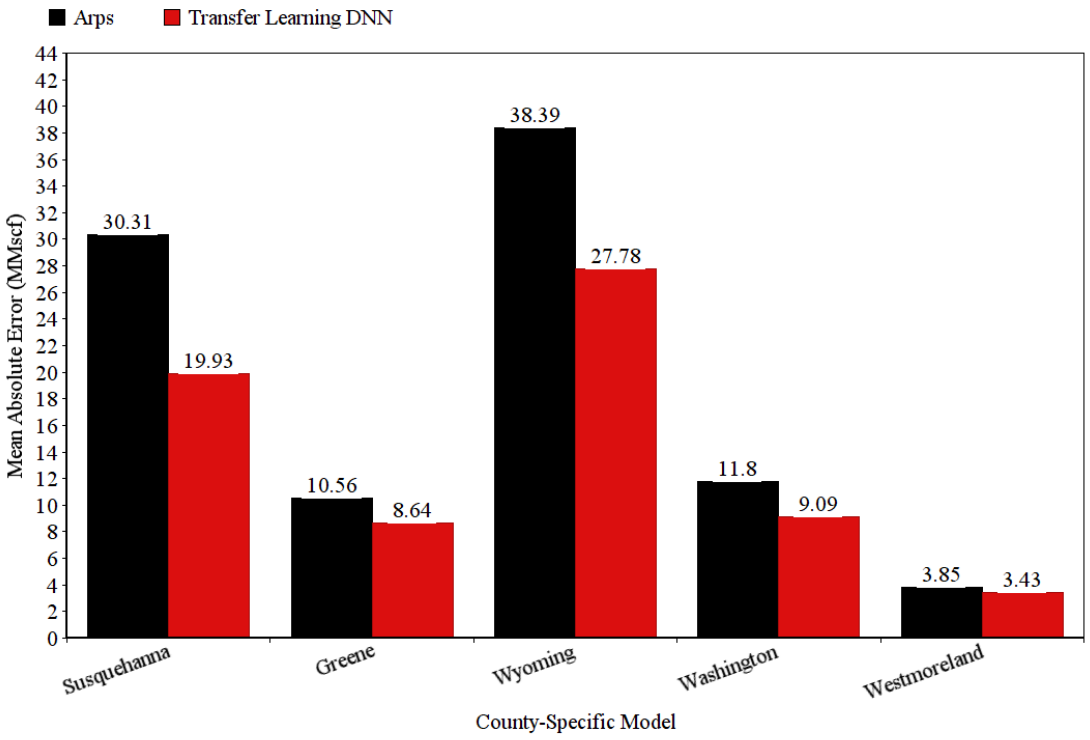

Figure 3-7: The new suggested county-specific DNN models achieved an error reduction between 34% in Susquehanna County and 11% in Westmoreland County compared to the Arps model benchmark when 6 months of input data were used to generate both forecasts.

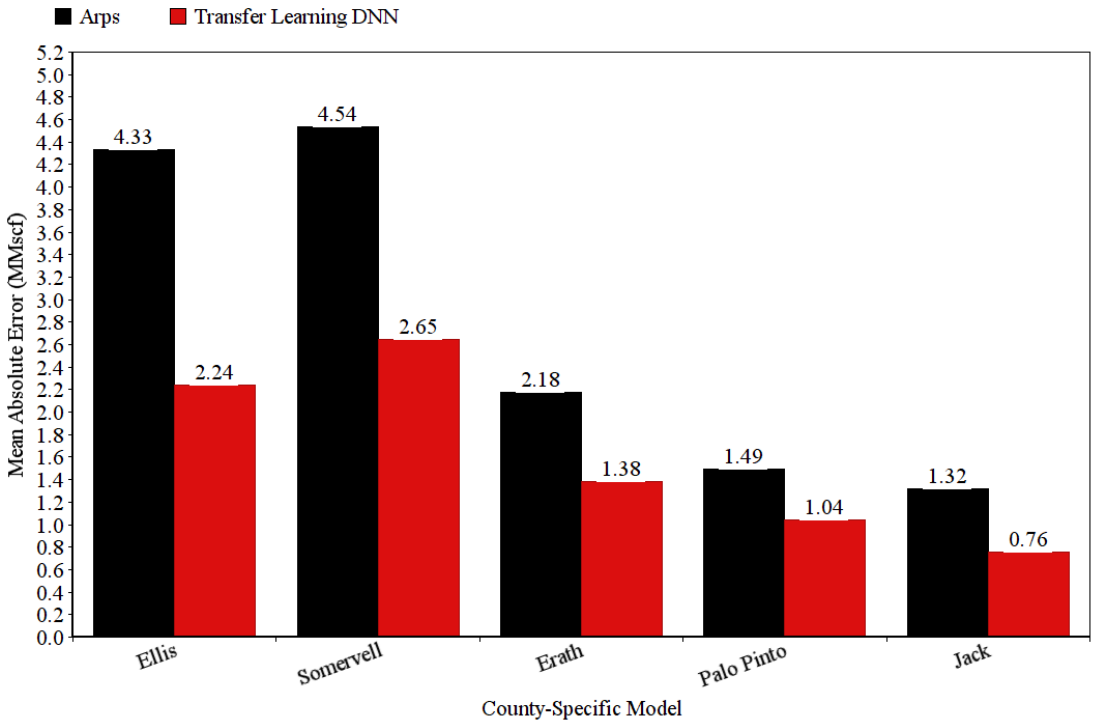

Figure 3-8: County-specific DNN models were able to achieve a significant error reduction compared to the Arps model ranging between 30% in Palo Pinto County and 48% in Ellis County despite the fact that the DNN model was not trained on that specific county data and the whole county's data was used for testing. The use of transfer learning enabled the DNN models to generate accurate forecast despite data scarcity in these counties. The source model $\mathbb{F}_S$ was trained on all other counties in the state, except for the county of interest, then all of this county's data was used for testing.

and Erath do not even have enough data to train a transfer learning model where the number of samples is very small to even train the last layer and develop a good predictive model. For this reason, the source model $\mathbb{F}_S$ which was trained on all other counties in that data set, except for the county of interest, was imported and used as is without any training where all that county's data will be used for testing. The model is then used to forecast the production in the county of interest. Figure 3-8 shows that when 6 months of input data is used, an error reduction between 48% in Ellis County and 30% in Palo Pinto County is achieved compared to the Arps model. Figures 3-9 and 3-10 show the error reduction compared to Arps when using

county-specific DNN models across all counties for each state. The percent reduction in error shown in the figures represents the weighted average error reduction on the all of the testing sets across all counties as shown in Equation 3.2:

$$\text{Overall Error Reduction} = \frac{\sum_i \text{Error Reduction in County(i)} \times \text{Number of test wells in County(i)}}{\text{Number of test wells in all counties}} \tag{3.2}$$

The figures show that the DNN approach consistently outperformed the Arps model across all counties and input months used as it averaged an error reduction on the Barnett shale data set between 41% when 4 months of input data is used to 33% when 10 months of input data is used and between 24% and 15% on the Marcellus shale data set.

The county-specific DNN models performed better on the Barnett data set than they did on the Marcellus data set due to the fact that production history available is longer allowing the DNN to learn better with more data. This shows that the county-specific DNN models can be improved further as more production history becomes available. The web pages of Railroad Commission of Texas (79) and Pennsylvania Department of Environmental Protection (80) publish new production data frequently giving the new approach an advantage over the Arps models which tend to overestimate the production in the middle and late life of a well.

3.5 Analysis & Discussion

As the results show in Section 3.4, DNN models can significantly reduce the error in forecasting compared to the Arps models. The use of transfer learning to develop county-specific DNN models generated a more accurate production forecast and provided the chance to develop DNN models for counties with very limited numbers of samples. One of the main reasons behind the improvement introduced by the DNN models is they have the ability to fit the data on models with up to 800 million parameters (101) giving them the power to generate models that can map very complex

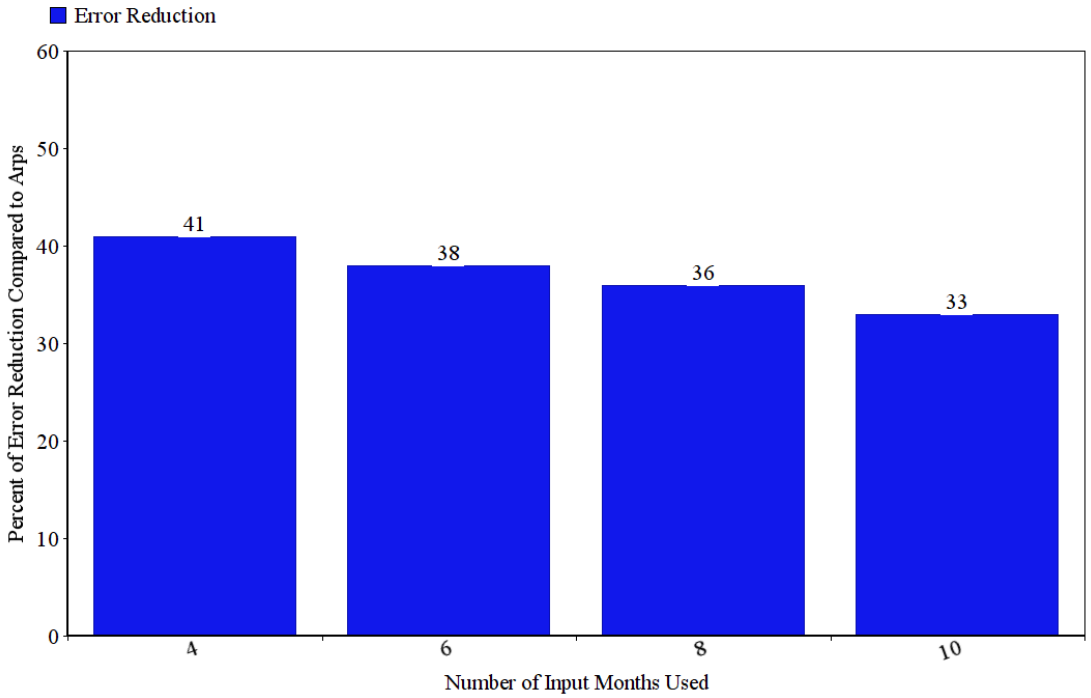

Figure 3-9: The error reduction achieved when using the county-specific DNN models compared to the Arps models on the whole Barnett data set. The error reduction is computed as the weighted average error reduction across all counties in the data set using Equation 3.2 when different values of input months is used.

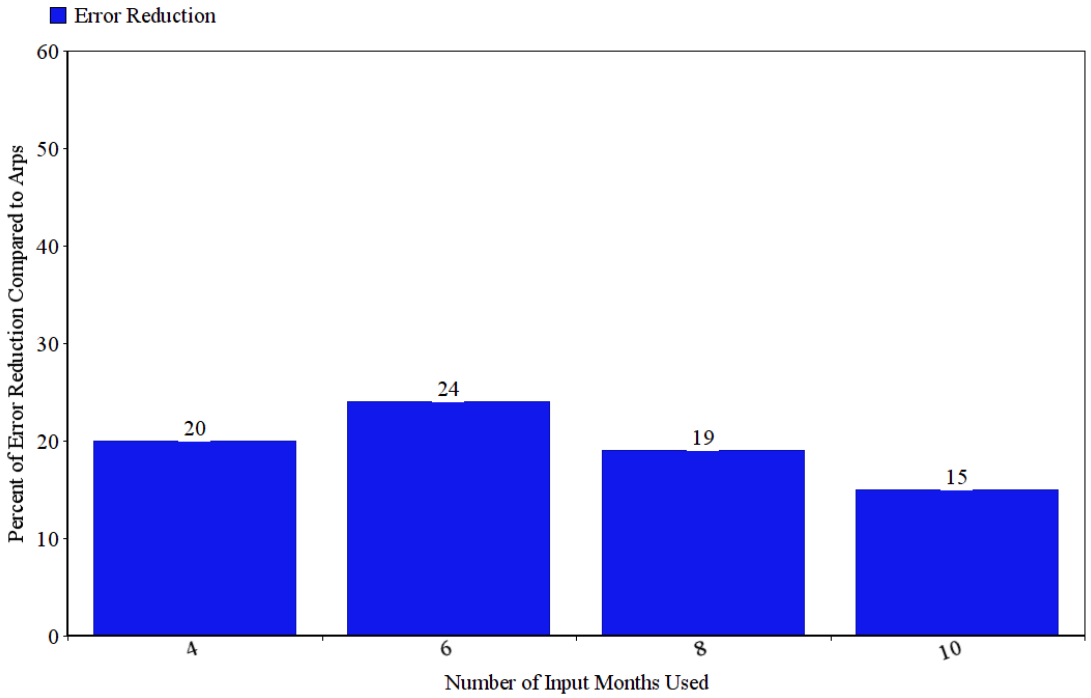

Figure 3-10: The error reduction achieved when using the county-specific DNN models compared to the Arps models on the whole Marcellus data set. The error reduction is computed as the weighted average error reduction across all counties in the data set using Equation 3.2 when different values of input months is used.

data. The DNN developed for this chapter uses more than 5000 parameters to generate its forecast while the Arps model only uses three and unlike Arps parameters which are provided by EIA as described in the Methods Section 3.3, the DNN tunes its parameters directly from the data through the training process in order generate a model that generalizes well on out-of-sample test data. The advancement in computing power along with efficient machine learning libraries and optimizers gave the chance to tune those 5000 parameters on thousands of samples in a few minutes on an average laptop.

As production data continue to grow due to numerous shale gas wells currently in production, the DNN models can be further improved by more data as they scale very well with data. Unlike the Arps model where the ill-posedness limits its ability to improve with more data, as the more the data becomes available the more the fluctuation in production can occur between different wells leading to non-unique parameter estimates.

This study showed using transfer learning to build county-specific DNN models offer a significant error reduction as the county-specific DNN model allowed the DNN to generate a more accurate forecast due to the fact the model can utilize the knowledge gained from other nearby counties then fine-tune its output (forecasting) layer specifically for the county of interest. Besides increasing the accuracy of the forecasts, transfer learning also helps overcome the data scarcity problem faced when a certain county or area of interest might not have enough data to properly train a DNN.

Transfer learning introduced a way of utilizing data from other counties' aggregate DNN model into a county-specific model despite the fact that two DNN models may have different domains D_T and D_S where the domain in this case refers to the geological formation. Although the two models have different domains, there is still similar features in the pattern of production decline and transfer learning allowed the target model $\mathbb{F}_T$ to improve its forecasting based on those similar patterns extracted by the source model $\mathbb{F}_S$. Even in the counties of Somervell, Ellis, Palo Pinto, Jack and Erath where data is not even enough to train a transfer learning model, the model that was trained only on adjacent counties' data was able to achieve significant error

reduction thanks to the ability of DNN pattern extraction.

However, there is still uncertainty on how much the change in geological properties and the reservoir conditions between the two domains D_T and D_S is affecting the results. The uncertainty is caused by the heterogeneity of the geology, which can be heterogeneous even at a small scale (102). Naturally the more the geology of the two domains differs, the less accurate the target model will be. Unfortunately, the data sets available do not contain geological information, which makes it difficult to investigate how the change in geology is affecting the accuracy of the target model.

In addition to the uncertainty emerging from the variation in geology, both the Arps and the DNN models have a major uncertainty regarding the effect of the change in technology and how that might affect the forecasting on a new well especially because these models are data driven and do not take into account the physical properties of each well. Developing a DNN model that utilizes physical properties such as well-bore sensors data, geology or geophysical data to enhance forecasting may introduce further improvement to the forecasting.

Similar to Sun et al. (78) where they used well head pressure as input, a good assumption can be made that further improvement can be achieved by incorporating a well-specific physical property such as perforated length, fracking fluid volume or sand volume, etc., as an input to the DNN as shown in Figure 3-11 since these parameters have an impact on the decline in production and the DNN may be able to learn the relationship between these parameters and the production decline. For example, in general a longer perforated length in a certain well leads to more production (103), where the DNN may be able to learn such a relationship between the perforated length and the rate of decline to enhance its forecasting. Unfortunately, due to data limitation on these properties such hypothesis was not tested.

The described DNN approach has a disadvantage compared to Arps models as DNN forecasting requires multiple models depending on the number of input months used to generate forecasts due to the fact the DNN will tune its weights during training specifically to map the input data of a certain length to the output data. For example, if a well has seven months of data and we want to forecast the next ten

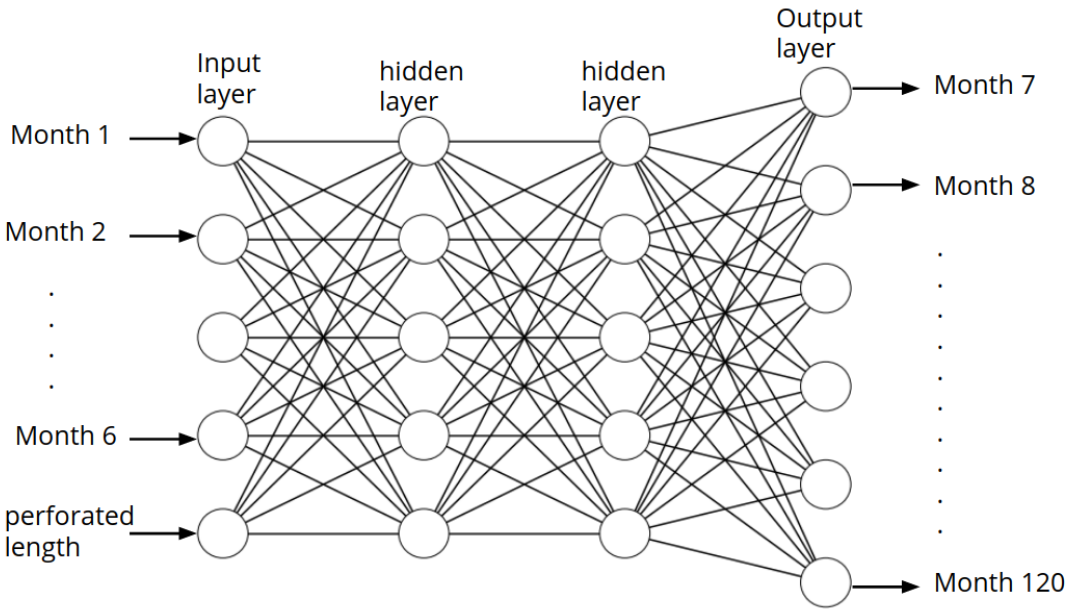

Figure 3-11: Adding well-specific physical properties such as perforated length, fracking fluid volume or sand volume may introduce further improvement to the DNN forecasting model as the DNN may be able to learn the relationship between such parameters and the rate of decline in production.

years of production then only the previously trained seven months DNN model can be used. The same challenge also applies to the output, where if the desired forecasting period is more or less than the desired period, only the previously trained model on that specific forecasting period can be used.

This causes the need for developing multiple DNN models for each field or county of interest in order to get flexibility in choosing the number of input months available to generate the forecast. Although the process of building DNN models requires more work to develop, compared to the Arps forecasting models, the DNN training process to generate the models can easily be automated, and is only needed once at the beginning. Thereafter, these DNN models can be used efficiently to generate the forecasts.

3.6 Conclusions

In this chapter the results show that DNNs can generate much better forecasts than the Arps decline curves by using transfer learning to develop county-specific DNN models fine-tuned to the county of interest as they were able to reduce the forecasting error up to 47% compared to the Arps decline curve model. The new suggested approach improves the current forecasting techniques used which is crucial for decision making and calculating returns on investment. Moreover, the results also show that the need for a large amount of data to produce accurate shale gas forecasts is no longer a hurdle for machine learning forecasting models as transfer learning can be used to overcome the data scarcity arising from a limited number of samples by transferring the knowledge gained from the other counties in the data set.

By offering a mechanism that allows the use of data from a specific area of interest as well as other nearby areas, this approach also offers the chance to easily improve the forecasting models further with more data as more production data will be available in the future. A realistic goal would be to gather all the production data from all the shale formations in the US to build a large model for each formation to serve as a source function $\mathbb{F}_S$ to transfer the knowledge it holds into county-specific DNN models that can produce much better forecasting results than the current widely used Arps model.

Chapter 4

Conclusions

The research work conducted in this thesis focused on developing machine learning based algorithms to address current challenges faced in subsurface energy production forecasting models. The field of production forecasting in both conventional and non-conventional energy resources faces considerable challenges. Some of these challenges arise from the fact that the problem itself is mathematically ill-posed. Another major challenge is that, in some cases the models lack reliable data that is necessary to achieve good accuracy. This research shows that machine learning methods can significantly improve the current models and techniques used to forecast future oil and gas production.

The research work in this thesis employed reinforcement learning to build an algorithm that can solve the history matching problem in parallel. The algorithm can train an artificial deep neural network agent that is capable of finding multiple different solutions to the history matching problem, even when dealing with tens of thousands of uncertain parameters. Solving the history matching problem gives the reservoir engineers the chance to find reliable reservoir models that can be used to forecast production in conventional oil and gas resources in a timely manner. The algorithm showed the capacity to reduce the run-time needed to find one solution by 88% from 18.6 minutes to 2.2 minutes, when using 16 computing cores instead of one. Thanks to such speed-up, we had the opportunity to tackle complex problems and find multiple solutions in a timely manner when tuning 27,000 uncertain parameters.

Additionally, this research shows that DNNs can generate much better forecasts than the Arps decline curves by using transfer learning to develop county-specific DNN models fine-tuned to the county of interest. The county-specific DNN models were able to reduce the forecasting error up to 47% compared to the Arps decline curve model. By using transfer learning, we also showed that the need for a large amount of data to produce accurate shale gas forecasts is no longer a hurdle for machine learning forecasting models. The utilization of transfer learning gave the chance to overcome the data scarcity arising from a limited number of samples by transferring the knowledge gained from the other counties in the data set. By offering a mechanism that allows the use of data from a specific area of interest as well as other nearby areas, this approach also offers the chance to easily improve the forecasting models further with more data as more and more production data becomes available in the future.

4.1 Contributions

The research presented in this thesis has made multiple contributions to the field of subsurface energy production forecasting. Specifically, this work has achieved the following:

- Successful parallelization of the history matching problem using reinforcement learning, resulting in high scalability and efficiency.
- Demonstration of the ability of reinforcement learning to find multiple solutions to the history matching problem by employing a stochastic policy. This property allows reservoir engineers to better quantify uncertainties in the production forecast.
- Establishment of transfer learning as a highly accurate method for creating county-specific models, which outperform traditional Arps decline curve models.
- Successful development of accurate shale gas production forecasting models despite the challenge of data scarcity.

Chapter 5

Future work

In terms of future work, both projects offer future research opportunities to build on the current findings. We plan to further improve the current framework for solving the history matching problem as well as explore new techniques that can help reduce the total run-time needed to solve the problem. We also plan to enhance the forecasting capabilities of the transfer learning project by further improving the source model data as well as developing a mechanism to provide the DNN models with more flexibility.

5.1 History matching future work

5.1.1 Using convolutional neural network

In the current framework, we use Multilayer perceptron (MLP) deep neural networks in order for the agent to map states to useful actions that minimize the error and reduce the objective function. However, convolutional neural networks (CNNs) (104) might provide a better way of mapping states to actions for some problems.

CNN is a variant of MLP that is best suited for images as input data. This is due to the fact that it uses convolutional layers in order to extract certain features from input data (105, 106) as shown in Figure (5-1).

In the future, we plan to encode each state of the reservoir model permeability into an image. This can be done by creating an image where each pixel represent the

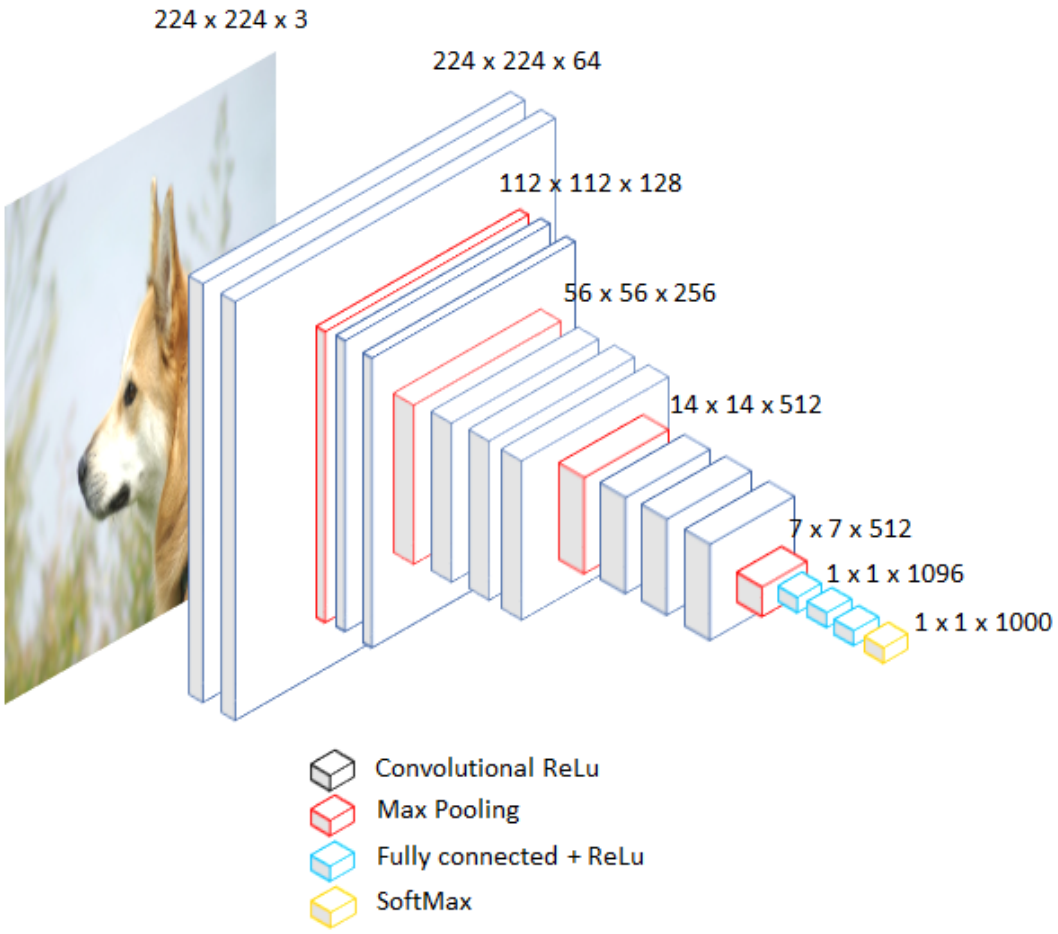

Figure 5-1: An illustration of multi-dimensional convlutional network can detect features in images (107)

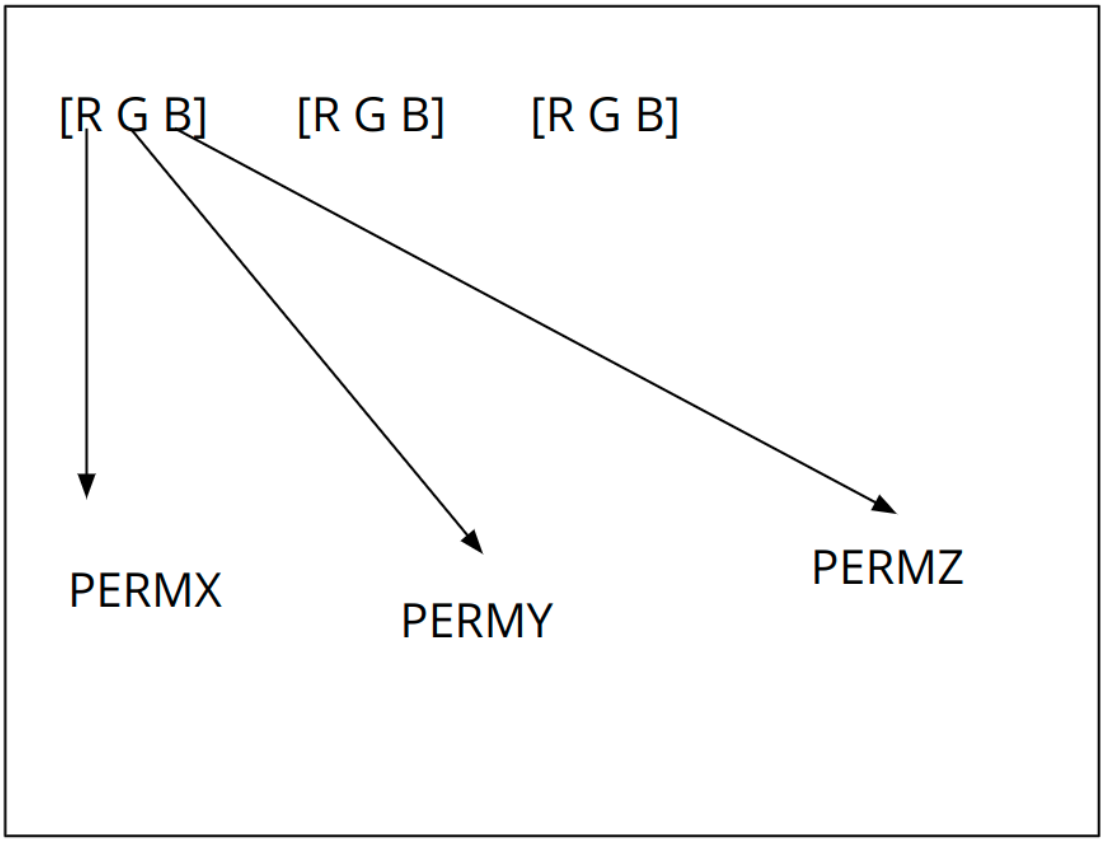

Figure 5-2: Encoding the values of permeability into each pixel creates a unique image of the current state of the system to utilize CNN architecture.

permeability value at a specific cell. Images are represented by a 2-D group of pixels, where each pixel is composed of 3 bytes number that holds the value of red, green and blue colors in the following manner pixel =[Red Green Blue]. These values are then used to generate the color of the pixel.

By encoding the values of permeability into each pixel as follows: pixel value =[PERMX PERMY PERMZ], a unique image of the current state of the system can be generated and fed to the CNN. Figure 5-2 shows an illustration of how to create a unique image for the permeability current state. However, such approach might have its own limitations due to the fact that CNNs usually detects features in images such as edges and in the created permeability image there might not be clear features to detect. This approach might work best for small models with a good geo-correlation between the cells.

5.2 Transfer learning future work

5.2.1 Building flexible DNN models

In terms of future work to improve the current transfer learning framework, we plan to explore techniques that help overcome the DNN model flexibility problem. As discussed in 3.5, the current framework requires the input data to have the same number of data points as the DNN model used for forecasting production.

We plan to develop a tool that automates the training process for the DNN models. The tool can create hundreds of DNN models for all the counties across different input data points. Such approach gives the users the flexibility of generating production forecasts without manipulating the data to fit the already built DNN model.

5.2.2 Improving the source model data

In the future we plan to improve the source model $\mathbb{F}_S$ by creating better data repositories that can be used to train it. The goal is to create a data pool for each shale gas formation in the US shown in Figure 5-3. This data repository can be used to easily develop county-specific models and also serve as data source for other researchers working on similar problems.

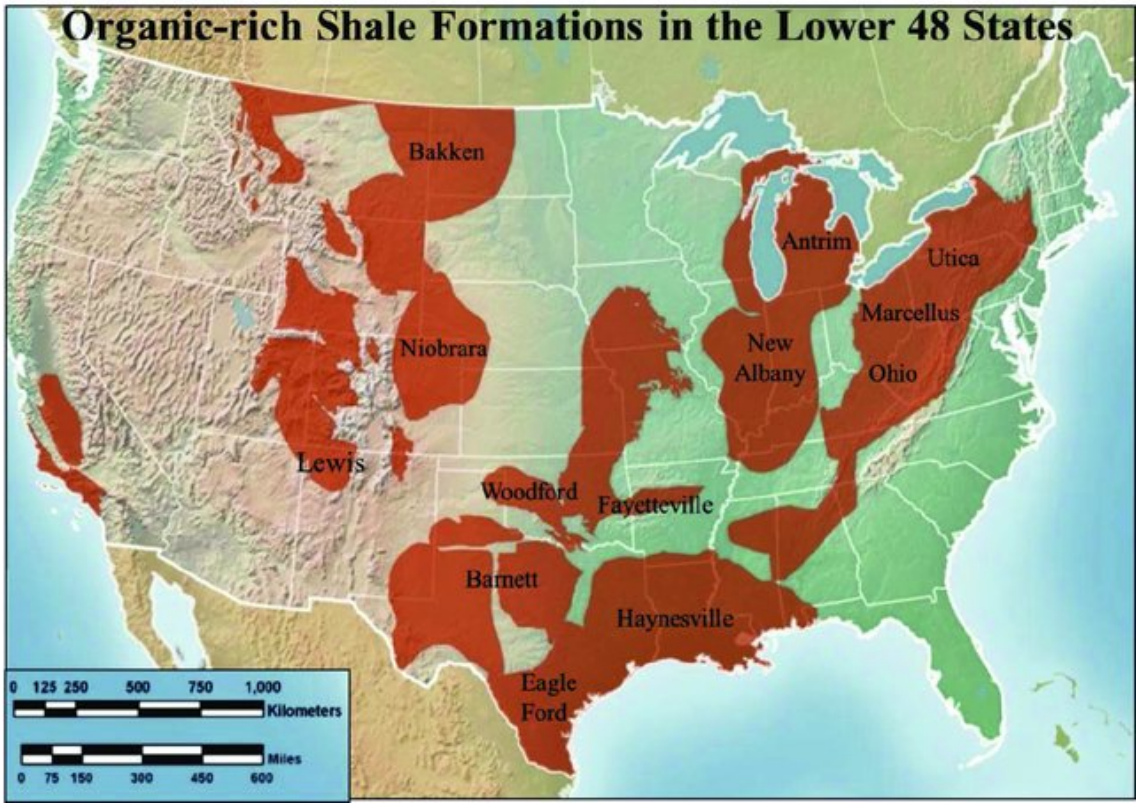

Figure 5-3: A map showing the locations of shale gas formations in the US (108). These formations can be used to build source models $\mathbb{F}_S$

Chapter 6

Appendices

6.1 Appendix A

Chapter 2 Experiments Hyperparameters

$K_\Delta = 100$ for for PermX, PermY and PermZ in SPE1.

$K_\Delta = 100$ for PermX and PermY and $= 1$ for PermZ in SPE9.

$\alpha = 1e^{-3}$

$C_q =$ Identity Matrix (I).

$C_u =$ Identity Matrix (I).

$\lambda = 1$ for SPE9 and $= 0.1$ for SPE1.

Algorithm total time-steps = 20,000 for SPE9 and until at least 1 solution is found for SPE1.

Learning rate $= 1 \times 10^{-5}$ for SPE9 and $= 9 \times 10^{-4}$ for SPE1.

MPI Processes per environment = 4 for SPE9 and = 1 for SPE1.

OMP threads per MPI Process = 2.

$\gamma = 0.99$.

Tolerance ϵ = 1,000 for SPE1 and = 2,500 for SPE9.

Reward for good termination $= 1 \times 10^4$ for SPE1 and $= 2.5 \times 10^6$ for SPE9.

Reward for divergence $= -1 \times 10^4$ for SPE1 and $= -2.5 \times 10^6$ for SPE9.

Reward for exceeding maximum time-steps per episode $= -1 \times 10^4$ for SPE1 and $= -1.25 \times 10^6$ for SPE9.

PPO Clipping Range = 0.2.

Maximum time-steps per episode = 100.

Batch size = 192 for SPE9 and = 32 for SPE1.

6.2 Appendix B

Chapter 2 Reservoir Models Data

SPE9

DX = 300 ft.

DY = 300 ft.

DZ = [20 , 15 , 26 , 15 , 16 , 14 , 8 , 8 , 18 , 12 , 19 , 18 , 20 , 50 , 100] ft.

Porosity = [0.087 , 0.097 , 0.111 , 0.16 , 0.13 , 0.17 , 0.17 , 0.08 , 0.14 , 0.13 , 0.12 , 0.105 , 0.12 , 0.116 , 0.157]

Wells Specs				
Well	i value of well head	j value of well head	Reference depth	Phase
INJE1	24	25	9110	Water
PRODU2	5	1	9110	Oil
PRODU3	8	2	9110	Oil
PRODU4	11	3	9110	Oil
PRODU5	10	4	9110	Oil
PRODU6	12	5	9110	Oil
PRODU7	4	6	9110	Oil
PRODU8	8	7	9110	Oil
PRODU9	14	8	9110	Oil
PRODU10	11	9	9110	Oil
PRODU11	12	10	9110	Oil
PRODU12	10	11	9110	Oil
PRODU13	5	12	9110	Oil
PRODU14	8	13	9110	Oil
PRODU15	11	14	9110	Oil
PRODU16	13	15	9110	Oil
PRODU17	15	16	9110	Oil
PRODU18	11	17	9110	Oil
PRODU19	12	18	9110	Oil
PRODU20	5	19	9110	Oil
PRODU21	8	20	9110	Oil
PRODU22	11	21	9110	Oil
PRODU23	15	22	9110	Oil
PRODU24	12	23	9110	Oil
PRODU25	10	24	9110	Oil
PRODU26	17	25	9110	Oil

SPE1

DX = 300 ft.

DY = 300 ft.

DZ = [20 , 30 , 50] ft.

Porosity = 0.3

Wells Specs				
Well	i value of well head	j value of well head	Reference depth	Phase
INJ	1	1	8335	Gas
PROD	10	10	8400	Oil

Bibliography

[1] Tara Energy. What is energy? a guide to understanding energy. URL https://taraenergy.com/blog/what-is-energy-a-guide-to-understanding-energy/#:~:text=Energy%20is%20so%20important%20in,or%20even%20lifting%20your%20finger.

[2] Exxon Mobil. The importance of energy: Imperial. URL https://www.imperialoil.ca/en-CA/company/about/the-importance-of-energy.

[3] Food and Agriculture Organization. Chapter 1: Energy in the world economy. URL https://www.fao.org/3/X8054E/x8054e04.htm#:~:text=Energy%20is%20a%20foundation%20stone,extraction%2C%20industrial%20production%20and%20transportation.

[4] Pennsylvania State University. Global energy sources. URL https://www.e-education.psu.edu/earth104/node/1345.

[5] International Energy Agency. Charts – data & statistics. URL https://www.iea.org/data-and-statistics/charts.

[6] Henrik Wachtmeister, Petter Henke, and Mikael Höök. Oil projections in retrospect: Revisions, accuracy and current uncertainty. *Applied Energy*, 220: 138–153, 2018. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2018.03.013. URL https://www.sciencedirect.com/science/article/pii/S0306261918303428.

[7] Raj Subrameyer. High level overview of machine learning classification - ml 101, Aug 2020. URL https://www.rajsubra.com/blog/2019/01/23/ml-classification.

[8] Omar S. Alolayan, Abdullah O. Alomar, and John R. Williams. Parallel automatic history matching algorithm using reinforcement learning. *Energies*, 16(2), 2023. ISSN 1996-1073. doi: 10.3390/en16020860. URL https://www.mdpi.com/1996-1073/16/2/860.

[9] Omar S. Alolayan, Samuel J. Raymond, Justin B. Montgomery, and John R. Williams. Towards better shale gas production forecasting using transfer learning. *Upstream Oil and Gas Technology*, 9:100072, 2022. ISSN 2666-2604. doi: https://doi.org/10.1016/j.upstre.2022.100072. URL https://www.sciencedirect.com/science/article/pii/S266626042200010X.

[10] Louis J. Durlofsky. Upscaling and gridding of fine scale geological models for flow simulation. 06 2005.

[11] Knut-Andreas Lie. *An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press, 2019. doi: 10.1017/9781108591416.

[12] Xian-Huan Wen and J.Jaime Gómez-Hernández. Upscaling hydraulic conductivities in heterogeneous media: An overview. *Journal of Hydrology*, 183(1): ix–xxxii, 1996. ISSN 0022-1694. doi: https://doi.org/10.1016/S0022-1694(96)80030-8. URL https://www.sciencedirect.com/science/article/pii/S0022169496800308.

[13] Ruijian Li, A. C. Reynolds, and D. S. Oliver. History Matching of Three-Phase Flow Production Data. *SPE Journal*, 8(04):328–340, 12 2003. ISSN 1086-055X. doi: 10.2118/87336-PA. URL https://doi.org/10.2118/87336-PA.

[14] Sylvester Okotie and Bibobra Ikporo. *Reservoir Engineering: Fundamentals and Applications*. 01 2019. ISBN 978-3-030-02392-8. doi: 10.1007/978-3-030-02393-5.

[15] Jincong He, Jiang Xie, Xian-Huan Wen, and Wen Chen. An alternative proxy for history matching using proxy-for-data approach and reduced order modeling. *Journal of Petroleum Science and Engineering*, 146:392–399, 2016. ISSN 0920-4105. doi: https://doi.org/10.1016/j.petrol.2016.05.026. URL https://www.sciencedirect.com/science/article/pii/S0920410516301966.

[16] Yamada Tomomi. Non-Uniqueness of History Matching. All Days, 04 2000. doi: 10.2118/59434-MS. URL https://doi.org/10.2118/59434-MS. SPE-59434-MS.

[17] Jérémie Bruyelle and Dominique Guérillot. Proxy Model Based on Artificial Intelligence Technique for History Matching - Application to Brugge Field. Day 2 Tue, October 22, 2019, 10 2019. doi: 10.2118/198635-MS. URL https://doi.org/10.2118/198635-MS. D021S010R004.

[18] Boxiao Li, Eric W. Bhark, (ret.) Stephen Gross, Travis C. Billiter, and Kaveh Dehghani. Best Practices of Assisted History Matching Using Design of Experiments. *SPE Journal*, 24(04):1435–1451, 05 2019. ISSN 1086-055X. doi: 10.2118/191699-PA. URL https://doi.org/10.2118/191699-PA.

[19] Célio Maschio and Denis José Schiozer. Bayesian history matching using artificial neural network and markov chain monte carlo. *Journal of Petroleum Science and Engineering*, 123:62–71, 2014. ISSN 0920-4105. doi: https://doi.org/10.1016/j.petrol.2014.05.016. URL https://www.sciencedirect.com/science/article/pii/S0920410514001338. Neural network applications to reservoirs: Physics-based models and data models.

[20] D.J. Schiozer, S.L. Almeida Netto, E.L. Ligero, and C. Maschio. Integration of History Matching And Uncertainty Analysis. *Journal of Canadian Petroleum Technology*, 44(07), 07 2005. ISSN 0021-9487. doi: 10.2118/05-07-02. URL https://doi.org/10.2118/05-07-02.

[21] Karl-Heinz Ilk. On the regularization of ill-posed problems. *Proc. Int. Symp. Figure and Dynamics of the Earth, Moon, and Planets, Prague*, -1:365, 01 1987.

[22] Mohammad Sayyafzadeh, Manouchehr Haghighi, and Jonathan N. Carter. Regularization in History Matching Using Multi-Objective Genetic Algorithm and Bayesian Framework. All Days, 06 2012. doi: 10.2118/154544-MS. URL https://doi.org/10.2118/154544-MS. SPE-154544-MS.

[23] Alireza Shahkarami. Artificial intelligence assisted history matching – proof of concept. 2012.

[24] Ibnu Arief. *Computer assisted history matching: A comprehensive study of methodology*. PhD thesis, 2013.

[25] N. Yamashita and M. Fukushima. *On the Rate of Convergence of the Levenberg-Marquardt Method*. Springer Vienna, Vienna, 2001. ISBN 978-3-7091-6217-0.

[26] Mehrdad G. Shirangi and Alexandre A. Emerick. An improved tsvd-based levenberg–marquardt algorithm for history matching and comparison with gauss–newton. *Journal of Petroleum Science and Engineering*, 143: 258–271, 2016. ISSN 0920-4105. doi: https://doi.org/10.1016/j.petrol.2016.02.026. URL https://www.sciencedirect.com/science/article/pii/S0920410516300687.

[27] L. Sanghyun and K. D. Stephen. Optimizing Automatic History Matching for Field Application Using Genetic Algorithm and Particle Swarm Optimization. Day 1 Tue, March 20, 2018, 03 2018. doi: 10.4043/28401-MS. URL https://doi.org/10.4043/28401-MS. D011S002R004.

[28] Hao Li and Siddharth Misra. Reinforcement learning based automated history matching for improved hydrocarbon production forecast. *Applied Energy*, 284:116311, 2021. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2020.116311. URL https://www.sciencedirect.com/science/article/pii/S0306261920316950.

[29] Vibeke Haugen, Geir Nævdal, Lars-Jøergen Natvik, Geir Evensen, Aina M. Berg, and Kristin M. Flornes. History Matching Using the Ensemble Kalman Filter on a North Sea Field Case. *SPE Journal*, 13(04):382–391, 12 2008. ISSN 1086-055X. doi: 10.2118/102430-PA. URL https://doi.org/10.2118/102430-PA.

[30] Xian-Huan Wen and Wen H. Chen. Real-Time Reservoir Model Updating Using Ensemble Kalman Filter With Confirming Option. *SPE Journal*, 11(04):431–442, 12 2006. ISSN 1086-055X. doi: 10.2118/92991-PA. URL https://doi.org/10.2118/92991-PA.

[31] Xian-Huan Wen and Wen H. Chen. Some Practical Issues on Real-Time Reservoir Model Updating Using Ensemble Kalman Filter. *SPE Journal*, 12 (02):156–166, 06 2007. ISSN 1086-055X. doi: 10.2118/111571-PA. URL https://doi.org/10.2118/111571-PA.

[32] Binghuai Lin, Paul Crumpton, and Ali Dogru. Parallel Implementation of Ensemble Kalman Smoother for Field-Scale Assisted History Matching. Day 4 Thu, March 09, 2017, 03 2017. doi: 10.2118/183755-MS. URL https://doi.org/10.2118/183755-MS. D041S045R001.

[33] Shusei Tanaka, Zhenzhen Wang, Kaveh Dehghani, Jincong He, Baskar Velusamy, and Xian-Huan Wen. Large Scale Field Development Optimization Using High Performance Parallel Simulation and Cloud Computing Technology. Day 3 Wed, September 26, 2018, 09 2018. doi: 10.2118/191728-MS. URL https://doi.org/10.2118/191728-MS. D031S030R007.

[34] Pallav Sarma, Jim Owens, Wen Chen, and Xian-Huan Wen. Massively distributed simulation and optimization on commercial compute clouds. *Society of Petroleum Engineers - SPE Reservoir Simulation Symposium 2015*, 3:1529–1536, 01 2015. doi: 10.2118/173280-MS.

[35] Society of Petroleum Engineers. Spe comparative solution project. https://www.spe.org/web/csp/index.html.

[36] Open Porous Media data repository. Opm data repository. "https://github.com/OPM/opm-data", 2022. Accessed: 2022-03-30.

[37] J.E. Killough. Ninth SPE Comparative Solution Project: A Reexamination of Black-Oil Simulation. All Days, 02 1995. doi: 10.2118/29110-MS. URL https://doi.org/10.2118/29110-MS. SPE-29110-MS.

[38] Aziz S. Odeh. Comparison of Solutions to a Three-Dimensional Black-Oil Reservoir Simulation Problem (includes associated paper 9741). *Journal of Petroleum Technology*, 33(01):13–25, 01 1981. ISSN 0149-2136. doi: 10.2118/9723-PA. URL https://doi.org/10.2118/9723-PA.

[39] Maxim Lapan. *Deep Reinforcement Learning Hands-on: Apply Modern RL Methods, with Deep Q-networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Expert insight. Packt Publishing, 2018. ISBN 9781788834247. URL https://books.google.com/books?id=9ZGvuAEACAAJ.

[40] Ahmad Hammoudeh. A concise introduction to reinforcement learning. 02 2018. doi: 10.13140/RG.2.2.31027.53285.

[41] JB Montgomery, SJ Raymond, FM O'Sullivan, and JR Williams. Shale gas production forecasting is an ill-posed inverse problem and requires regularization. *Upstream Oil and Gas Technology*, 5:100022, 2020. ISSN 2666-2604. doi: https://doi.org/10.1016/j.upstre.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2666260420300220.

[42] Kai Zhang, Niantian Lin, Chao Fu, Dong Zhang, Xing Jin, and Chong Zhang. Reservoir characterisation method with multi-component seismic data by unsupervised learning and colour feature blending. *Exploration Geophysics*, 50(3): 269–280, 2019. URL https://doi.org/10.1080/08123985.2019.1603078.

[43] Ruslan Miftakhov, Abdulaziz Al-Qasim, and Igor Efremov. Deep Reinforcement Learning: Reservoir Optimization from Pixels. Day 2 Tue, January 14, 2020, 01 2020. doi: 10.2523/IPTC-20151-MS. URL https://doi.org/10.2523/IPTC-20151-MS. D021S052R002.

[44] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461.

[45] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[46] Atgeirr Flø Rasmussen, Tor Harald Sandve, Kai Bao, Andreas Lauser, Joakim Hove, Bård Skaflestad, Robert Klöfkorn, Markus Blatt, Alf Birger Rustad, Ove Sævareid, Knut-Andreas Lie, and Andreas Thune. The open porous media flow reservoir simulator. *Computers & Mathematics with Applications*, 81:159–185, 2021. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2020.05.014. URL https://www.sciencedirect.com/science/article/pii/S0898122120302182. Development and Application of Open-source Software for Problems with Numerical PDEs.

[47] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. *CoRR*, abs/1712.00378, 2017. URL http://arxiv.org/abs/1712.00378.

[48] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[49] Elmar Diederichs. Reinforcement learning - a technical introduction. *Journal of Autonomous Intelligence*, 2:25, 08 2019. doi: 10.32629/jai.v2i2.45.

[50] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

[51] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[52] Saeed S. Alahmari, Dmitry B. Goldgof, Peter R. Mouton, and Lawrence O. Hall. Challenges for the repeatability of deep learning models. *IEEE Access*, 8: 211860–211868, 2020. doi: 10.1109/ACCESS.2020.3039833.

[53] Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359 (6377):725–726, 2018. doi: 10.1126/science.359.6377.725. URL https://www.science.org/doi/abs/10.1126/science.359.6377.725.

[54] Ruijian Li, A. C. Reynolds, and D. S. Oliver. History Matching of Three-Phase Flow Production Data. *SPE Journal*, 8(04):328–340, 12 2003. ISSN 1086-055X. doi: 10.2118/87336-PA. URL https://doi.org/10.2118/87336-PA.

[55] Leila Heidari, V. Gervais, Mickaele Le Ravalec, and Hans Wackernagel. History matching of reservoir models by ensemble kalman filtering: The state of the art and a sensitivity study. *AAPG Memoir*, 01 2011. doi: 10.1306/13301418M963486.

[56] Mohamed Shams, Ahmed. H. El-Banbi, and Helmy Sayyouh. A Comparative Study of Proxy Modeling Techniques in Assisted History Matching. Day 3 Wed, April 26, 2017, 04 2017. doi: 10.2118/188056-MS. URL https://doi.org/10.2118/188056-MS. D033S020R003.

[57] B. M. Negash, Mohammed A. Ayoub, Shiferaw Regassa Jufar, and Aban John Robert. History matching using proxy modeling and multiobjective optimizations. In Mariyamni Awang, Berihun Mamo Negash, Nur Asyraf Md Akhir, Luluan Almanna Lubis, and Abdul Ghani Md. Rafek, editors, *ICIPEG 2016*, pages 3–16, Singapore, 2017. Springer Singapore. ISBN 978-981-10-3650-7.

[58] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. 2019. doi: 10.48550/ARXIV.1911.10635.

[59] Pieter Jan 't Hoen, Karl Tuyls, Liviu Panait, Sean Luke, and J. A. La Poutré. An overview of cooperative and competitive multiagent learning. In *Proceedings of the First International Conference on Learning and Adaption in Multi-Agent Systems*, LAMAS'05, page 1–46, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540330534. doi: 10.1007/11691839_1. URL https://doi.org/10.1007/11691839_1.

[60] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[61] US Energy Information Administration. URL:https://www.eia.gov/. Accessed: 2021-01-30.

[62] A. Agosta, G. Boccara, B. Heringa, N. Browne, and D. Dediu. Mckinsey & company global gas outlook to 2050. https://www.mckinsey.com/industries/oil-and-gas/our-insights/global-gas-outlook-to-2050, 2021. Accessed: 2021-05-30.

[63] W. Lee and R. Sidel. Gas reserves estimation in resource plays. *Society of Petroleum Engineers*, 2, 2010. doi: 10.2118/130102-PA.

[64] C. Clarkson, J. Williams-Kovacs, F. Qanbari, and M. Heidari Sureshjanid. History-matching and forecasting tight/shale gas condensate wells using combined analytical, semi-analytical, and empirical methods. *Journal of Natural Gas Science and Engineering*, 26, 2015. doi: /10.1016/j.jngse.2015.03.025.

[65] L. Lee, J. Lim, D. Yoon, and H. Jung. Prediction of shale-gas production at duvernay formation using deep-learning algorithm. *Society of Petroleum Engineers*, 24, 2019.

[66] D. Han, J. Jung, and S. Kwon. Comparative study on supervised learning models for productivity forecasting of shale reservoirs based on a data-driven approach. *Applied Science*, 10, 2020.

[67] L. Tan, L. Zuo, and B. Wang. Methods of decline curve analysis for shale gas reservoirs. *Energies*, 11, 2018. doi: 10.3390/en11030552.

[68] P. Manda and D. Nkazi. The evaluation and sensitivity of decline curve modelling. *Energies*, 13:2765, 2020. doi: 10.3390/en13112765.

[69] T. Patzek, F. Male, and M. Marder. Gas production in the barnett shale obeys a simple scaling theory. *Proceedings of the National Academy of Sciences*, 2013.

[70] R. De Holanda, E. Gildin, and P. Valko. Combining physics, statistics, and heuristics in the decline-curve analysis of large data sets in unconventional reservoirs. *Society of Petroleum Engineers*, 2018. doi: 10.2118/185589-PA.

[71] A. Duong. Rate-decline analysis for fracture-dominated shale reservoirs. *Society of Petroleum Engineers*, 2011. doi: 10.2118/137748-PA.

[72] A. Clark, L. Lake, and T. Patzek. Production forecasting with logistic growth models. In *Society of Petroleum Engineers*, 2011. doi: 10.2118/144790-MS.

[73] J. Arps. Analysis of decline curves. *Transactions of the American Institute of Mining Engineers*, 160, 1945. doi: 10.2118/945228-G.

[74] Q. Zhao, H. Wang, Q. Sun, X. Jiang, R. Yu, L. Kang, and X. Wang. A logical growth model considering the influence of shale gas reservoirs and development characteristics. *Natural Gas Industry B*, 7, 2020. doi: https://doi.org/10.1016/j.ngib.2020.05.005.

[75] J. Montgomery, S. Raymond, F. O'Sullivan, and J Williams. Shale gas production forecasting is an ill-posed inverse problem and requires regularization. *Upstream Oil and Gas Technology*, 5, 2020. doi: 10.1016/j.upstre.2020.100022.

[76] S. Al-Fattah and R. Startzman. Predicting natural gas production using artificial neural network. *Society of Petroleum Engineers*, 2001. doi: 10.2118/68593-MS.

[77] Y Li and Y Han. Decline curve analysis for production forecasting based on machine learning. In *SPE Symposium: Production Enhancement and Cost Optimisation, Kuala Lumpur, Malaysia*, 2017.

[78] J. Sun, X. Ma, and M. Kazi. Comparison of decline curve analysis dca with recursive neural networks rnn for production forecast of multiple wells. In *SPE Western Regional Meeting*, 2018. doi: https://doi.org/10.2118/190104-MS.

[79] Railroad Commission of Texas. Online research queries. `https://www.rrc.state.tx.us/`, 2019. Accessed: 2020-01-30.

[80] Pennsylvania Department of Environmental Protection. Pa oil and gas mapping. `https://www.dep.pa.gov/DataandTools/Reports/Oil%20and%20Gas%20Reports/Pages/default.aspx`, 2020. Accessed: 2021-03-30.

[81] US Energy Information Administration. Oil and gas supply module of the national energy modeling system 2018b. 2018.

[82] Abdolrahim Ataei, Eghbal Motaei, Mohammad Ebrahim Yazdi, Rahim Masoudi, and Aamir Bashir. Rate Transient Analysis RTA and Its Application for Well Connectivity Analysis: An Integrated Production Driven Reservoir Characterization and a Case Study. volume Day 1 Tue, October 23, 2018 of *SPE Asia Pacific Oil and Gas Conference and Exhibition*, 10 2018. doi: 10.2118/192046-MS. URL `https://doi.org/10.2118/192046-MS`. D011S001R001.

[83] K. Okuszko, B. Gault, and L. Mattar. Production decline performance of cbm wells. *Journal of Canadian Petroleum Technology*, 2008. doi: https://doi.org/10.2118/08-07-57.

[84] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2, 1944. doi: 10.1090/qam/10666.

[85] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11, 1963. doi: 10.1137/0111030.

[86] M. Newville, T. Stensitzki, M. Allen, D.and Rawlik, A. Ingargiola, and A. Nelson. Non-linear least-squares minimization and curve-fitting for python. `URL: https://lmfit.github.io/lmfit-py/`, 2020. Accessed: 2020-01-30.

[87] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: Data mining, inference, and prediction, second edition. *Springer Series in Statistics*, 2016.

[88] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15, 2014.

[89] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations, San Diego*, 2017.

[90] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning, 2020.

[91] Q. Yang, Y. Zhang, W. Dai, and SJ Pan. *Transfer Learning*. Cambridge University Press, 2020. doi: 10.1017/9781139061773.

[92] S. Pan and Q. Yang. A survey on transfer learning. *Institute of Electrical and Electronics Engineers Transactions on Knowledge and Data Engineering*, 22, 2010. doi: 10.1109/TKDE.2009.191.

[93] A. Zisserman and K Simonyan. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations , San Diego, Ca, USA*, 2015.

[94] J. Deng, W. Dong, R. Socher, and L. Li. Imagenet: a large-scale hierarchical image database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), Miami, Florida, USA*, 2009.

[95] J. Brownlee. How to classify photos of dogs and cats (with 97% accuracy). `https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos\protect\@normalcr\relax-of-dogs-and-cats/`, 2019. Accessed: 2020-09-30.

[96] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. ISBN 978-0262035613.

[97] M. Nielsen. *Neural Networks and Deep Learning*. 2020. "`https://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf`" Accessed: 2021-05-30.

[98] C. Bishop. Pattern recognition and machine learning. *Springer*, 2006.

[99] J. Brownlee. Use early stopping to halt the training of neural networks at the right time. `https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time\protect\@normalcr\relax-using-early-stopping/`, 2019. Accessed: 2021-05-30.

[100] M. Huston. American association for the advancement of science. *American Association for the Advancement of Science*, 359, 2018. doi: 10.1126/science.359.6377.725.

[101] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, and L Bharambe, A. andvan der Maaten. Exploring the limits of weakly supervised pretraining. *CoRR*, abs/1805.00932, 2018. URL `http://arxiv.org/abs/1805.00932`.

[102] J. Montgomery and F. O'Sullivan. Spatial variability of tight oil well productivity and the impact of technology. *Applied Energy*, 195, 2017. doi: 10.1016/j.apenergy.2017.03.038.

[103] Y. Xia, T. Xu, Y. Yuan, and X. Xin. Effect of perforation interval design on gas production from the validated hydrate-bearing deposits with layered heterogeneity by depressurization. *Geofluids, Hindawi*, 2020. doi: 10.1155/2020/8833884.

[104] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL `https://arxiv.org/abs/1511.08458`.

[105] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[106] L. Alzubaidi, J. Zhang, and A.J. Humaidi. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8:53, 2021. doi: https://doi.org/10.1186/s40537-021-00444-8. URL `https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8`.

[107] Victor Roman. Convolutional neural networks in practice - develop and implement your first cnn with keras!, 2020. URL `https://towardsdatascience.com/convolutional-neural-networks-in-practice-406426c6c19a`.

[108] Angela Goodman, Isis Fukai, Robert Dilmore, Scott Frailey, Grant Bromhal, Daniel Soeder, Charlie Gorecki, Wes Peck, Traci Rodosta, and George Guthrie. Methodology for assessing co2 storage potential of organic-rich shale formations. *Energy Procedia*, 63:5178–5184, 12 2014. doi: 10.1016/j.egypro.2014.11.548.