TKP4580 - Specialization Project, Process Systems Engineering

Extrapolation With Neural Network-Based Virtual Flow Metering By Including First Principle Models

Victoria Glott

Submission date: 19.12.2019
Supervisor: Johannes Jäschke
Co-Supervisor: Timur Bikmukhametov

Department of Chemical Engineering
Norwegian University of Science and Technology

Abstract

Virtual Flow Metering (VFM) is becoming an increasingly attractive method for estimating the flow of oil and gas. Data-driven VFM requires a large amount of data to train machine learning algorithms. This is problematic for VFM where the available training data often origins from wells with different trajectories and operating conditions. Neural networks are capable of generalizing the function inside the subspace spanned by the training samples but will struggle if presented with new data outside the trained range. This project will, therefore, investigate the extrapolation capabilities of neural network-based Virtual Flow Metering. Two methods will be investigated. The first method is based on feature engineering and the objective was to transform the raw data into physically meaningful features as well and including first principle model solutions as input. The second method aims to shift the original distribution of the training and validation sets by incorporating first principle model solutions into the target variable, thus reducing the extrapolation range. The first principle models which will investigate are the Bernoulli choke model, Al-Safran choke model, and the "No Pressure Wave" tubing model. The data was produced synthetically with LedaFlow, and measurement noise was included to make the data set realistic. Ten case studies where performed, including one baseline model. Method 1 consists of five case studies that investigate various features and how the first principle models affect performance. Method 2 also covers five case studies where different first principle model solutions were incorporated into the target variable. It has been shown that Method 1 and 2 can increase the extrapolation capabilities and performance of a neural network-based VFM. However, Method 1 proved to be superior when the distribution of training and validation sets were more equal, while Method 2 resulted in the highest performance when the target distributions for the training and validation sets were more separated.

Contents

List of Figures

List of Tables

1 Introduction

Oil and gas have become, and still is, the most important source of energy. However, as fields start to deplete and become economically marginal, optimizing the production of oil and gas is crucial for maintaining the field viable [AL-Qutami et al., 2018]. Monitoring real-time production can provide information about reservoir size and how the well stream parameters should be altered in order to maximize production. In addition, it can contribute to determining the positioning of new wells and installations [Marshall and Thomas, 2015].

Well testing and multiphase flow meters (MPFM) are two widely used techniques to monitor the flow of oil and gas. However, these methods are linked to large capital and operational costs. MPFM also has a limited operational envelope. Virtual Flow Metering (VFM) has become an increasingly attractive and represents a more cost-effective solution alternative which is able to meet the monitoring demands [AL-Qutami et al., 2018]. By utilizing existing measurements, VFM can estimate the oil, gas and water rates without measuring it directly. As the name suggests, VFM does not require any additional hardware installations thus eliminating the need for physical maintenance.

Data-driven VFM, which will be the focus of this project, estimates the flow using machine learning models which maps the relationship between the system state variables and corresponding flow rates. The state variable typically includes pressure and temperature readings at the bottomhole and wellhead and choke opening. Being a machine learning-based alternative, the risk of bias and error increases when the VFM is operating outside the trained range [Marshall and Thomas, 2015]. It is commonly accepted that neural networks have poor extrapolation skills and are often considered as "a prisoner of its training data set" [Minns and Hall, 1996]. This is often ignored when deploying machine learning models into the real world and could be a significant disadvantage during the operational phase of a VFM [Varoonchotikul, 2003].

A complete oil and gas producing system typically consist of a reservoir, several individual wells, wellhead valve, flowlines, separators, pumps and transportation pipelines. The well is responsible for transporting oil, gas, and water from the reservoir to the surface where the production is typically controlled by a wellhead valve. The flowline transports the produced fluids to a processing facility where the oil, gas, and water are separated. The pumps and compressors further transport the oil and gas through pipelines to the sales point [Ghalambor, 2007]. Overall, oil and gas production systems are complex, but the scope of this project will be narrowed down to the well, wellhead valve, flowline, and their corresponding available measurements.

This project will aim to investigate several methods that could potentially improve the extrapolation capabilities of a neural network-based VFM. It is also desirable to explain why certain methods improve

performance. Therefore the training and validation data sets are required to be selected carefully and systematically. The data was therefore produced synthetically using LedaFlow which is a simulation tool provided by Kongsberg Digital[Kongsberg, 2001]. There are two main methods which will be considered:

1. Method 1: Feature engineering and including principle model solutions as inputs.
2. Method 2: Incorporating first principle model solutions into the target variable.

There are two scenarios where being able to extrapolate with a neural network-based VFM will be advantageous. Firstly, if the VFM is intended to operate as a stand-alone solution, the available data will origin from wells with different compositions, trajectories, and operating conditions. Secondly, assuming the VFM is trained based on the estimates from the MPFM, and the VFM operates as a backup or supplementary solution. Then, if the MPFM fails or drift, the VFM will eventually enter a zone of operating conditions that are different from the distribution it was trained upon.

2 Theory

The theory section is meant to provide a background regarding the case studies which will be presented and is organized as follows. The first section gives a brief overview of methods, working principle and preliminary work on Virtual Flow Metering. Then the basic working principle behind neural networks is described as well as previous work on extrapolating with neural networks. Bayesian hyperparameter optimization as a tuning technique will also be presented. Since this study aims to determine whether the inclusion of first principle models can improve the performance, the Bernoulli choke model, Al-Safran choke model and the "No pressure wave" tubing equation will be described.

2.1 Virtual Flow Metering

Accurate multiphase flow rate measurements are important in reservoir management, production optimization, and rate allocation [Marshall and Thomas, 2015]. There are different methods in the industry that are used to measure the flow such as multiphase flow meters(MPFM) and test separators, both having their advantages and disadvantages [Robertson, 2014]. To perform a well test, produced fluids from one well are isolated and sent to a test separator. The test separators typically require a separate flowline which ensures that a sample from each well is transported to the separator without shutting down the field [Bikmukhametov and Jäschke, 2019]. Combining the resulting fractions from the separator and the duration of the well test gives a snapshot of the production rate [Marshall and Thomas, 2015]. In contrast, MPFM exploits a combination of multiple measurement principles and provides real-time monitoring of the flows [Barbariol et al., 2019]. A major disadvantage with MPFM is the large capital cost and physical maintenance. It is also sensitive to corrosion, blockages and requires careful calibration both in the start-up phase and during its lifetime [Marshall and Thomas, 2015].

Due to an increased interest in real-time monitoring of the flow, several new methods have emerged to estimate the rate both accurate and cost-effective. Virtual Flow Metering is a method for estimating produced oil, gas and water without measuring the rates directly and is becoming an increasingly attractive alternative in the oil and gas industry [Bikmukhametov and Jäschke, 2020b]. By utilizing existing field measurements, it can provide a real-time estimation of the flow [Robertson, 2014]. Typical measurements used in VFM are:

- Bottomhole pressure, P_{BH}
- Bottomhole temperature, T_{BH}
- Wellhead pressure upstream and downstream of the valve, P_{WHCU} and P_{WHCD}
- Wellhead temperature upstream and downstream of the valve, T_{WHCU} and T_{WHCD}

- Choke opening, C_{op}

In general, VFM represents a more affordable alternative compared to MPFM and well testing. Since VFM is based on existing measurements, it does not require any additional installation of hardware. Therefore, a VFM eliminates the need for maintenance such as handling corrosion and sand plugging. VFM can also be a useful tool for estimating future production and predicting the outcome of different scenarios. However, since VFM is based on sensor measurements, the reliability can be diminished if the sensors are calibrated poorly or fail. [Marshall and Thomas, 2015].

There exist various methods of how to estimate multiphase flows using VFM and multiple companies have developed commercial VFM. Several of these methods are being used in the industry, either as a standalone solution or as a backup system to a MPFM [Bikmukhametov and Jäschke, 2020a]. Based on model methodology VFM can be distinguished into two categories,

- First principle VFM
- Data-driven VFM

First principle VFM is based on physical models which describe the relationship between existing measurements and corresponding rates. The models utilize sensor reading along the flow path such as pressure and temperatures in the wellbore and the pressure drop across the choke. The models are calibrated by optimizing the tuning parameters to calculate the current rate for the individual flows [Holmås and Løvli, 2011]. By now, first principle VFM is the most popular choice in the industry. [Bikmukhametov and Jäschke, 2020a].

Similarly to first principle VFM, data-driven VFM is also based on collecting available field data to fit a mathematical model. In contrast to first principle VFM, data-driven VFM utilizes a black-box mathematical model, referred to as machine learning model [Bikmukhametov and Jäschke, 2020a]. Data-driven VFM is based on the relationship between the system state variables; pressure, temperature and choke opening, and the system output; corresponding flowrates. Since data-driven VFM does not require any exact knowledge about the behavior of the system, the process of developing complex multiphase flow models is eliminated. However, being a machine learning-based alternative, data-driven VFM introduces the risk of bias or error if it is operating outside the trained range [Solomatine and Ostfeld, 2008]. Nevertheless, VFM has gained increased attention in the oil and gas industry because of its performance and accurate predictions.

Data-driven VFM requires preprocessing of the data to train the machine learning model. Dealing with missing values, which most machine learning algorithms do not allow, is crucial. Abnormalities such as outliers and noise in the data set must be taken care of before the training phase, as well as during

the operational phase. Feature engineering is a popular technique used in machine learning and is also applicable to data-driven VFM. Transforming the raw data into physically meaningful features, like the pressure drop over the choke or tubing, could potentially represent the underlying relationship between input and output more accurate and enhance the performance.

After the data is preprocessed, the next step is to develop the machine learning algorithm. There have been tested several algorithms, like radial basis function network [T. A. Al-Qutami et al., 2017], gradient boosting algorithm with regression trees [Bikmukhametov and Jäschke, 2019] and various types of neural networks such as Long short-term memory neural networks [Andrianov, 2018]. After the model is trained the performance must be verified to ensure that the performance is sufficient.

2.2 Neural Networks

Machine learning is a branch of artificial intelligence that identifies patterns and makes decisions without human intervention. Neural networks are one of the most popular machine learning algorithms. Forecasting, pattern recognition, and decision making are among the main applications of neural networks [Mak et al., 2019]. During the last years, neural networks have gained large attention when repeatedly outperforming other machine learning methods [Schmidhuber, 2015]. The increase in popularity is caused by the improvements in processing power, the development of sophisticated algorithms, and the availability of data.

2.2.1 Working principle

Artificial neural networks(ANN) are, as the name suggests, computational networks that try to simulate the decision process similar to how nerve cells in a human body interact. For regression problems, the objective is to minimize the error between the target and predicted values. There are several metrics which describes the error and one of the most commonly used in regression problems is the mean squared error [Lars Hulstaert, 2017]:

$$J = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{1}$$

where $\hat{y}_\mathrm{i}$ denotes the predicted value, $\mathrm{x_i}$ - the input, $\mathrm{y_i}$ - the true value and J - the objective function.

A standard neural network consists of many simple neurons that are interconnected. Neurons are the decision-makers in a neural network and decide the importance of each feature. The architecture of the neural network is built up by a specific number of layers with each layer containing a fixed number of neurons. The input layer is the set of neurons which first receives the input and defines the beginning of

the workflow. The output layer produces the results for the given inputs while are all the intermediate layers are known as hidden layers. An illustration of an architecture is shown in Figure 1.

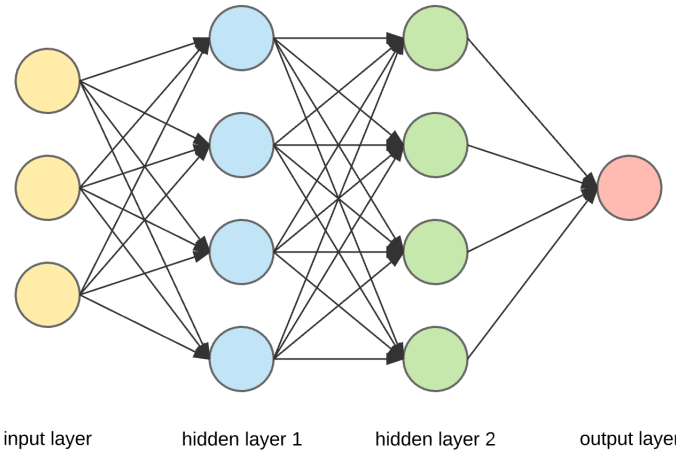


Figure 1: An illustration of a neural network with one input layer, two hidden layers and an output layer.

Each neuron has a specific weight function, which either amplifies or dampens the input based on the significance of the input with respect to the learning objective [Schmidhuber, 2015]. Whether a neuron is activated is controlled by an activation function, which determines if the sum of inputs, x_i and weights, w_i,

$$Y = \sum_{i=1}^{N} (x_i w_i) \tag{2}$$

are above a certain threshold. Intuitively, the activation of the neuron determined by the significance of the input to the target variable. There are different types of activation functions. ReLU is one of the most commonly used and is defined as linear for all positive values, and zero for all negative. The simple mathematical form yields easy computational operation and is thus fast and efficient. Besides, ReLU also is advantageous in terms of not suffering from vanishing or exploding gradients [Jiang et al., 2018].

An important goal of machine learning is the ability to generalize the training data to produce accurate predictions on unseen data. This is crucial because the expected performance on a new data set is not necessarily consistent with the training accuracy. Like all machine learning models, neural networks are susceptible to overfitting which occurs when the model is too closely fitted to the training set [Lawrence and Giles, 2000]. There are different techniques to prevent overfitting, and all methods which aim to increase the generalization are collected under the generic term regularization. Regularization can include various modifications such as changing the network architecture or tuning the hyperparameters.

A popular technique is to add a loss term to the objective function to penalize large weight updates which might only benefit the training set,

$$J = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \lambda w_2^2 \tag{3}$$

where λ denotes the regularization coeffecient. By penalizing each weight update, the regularization will encourage the neural network to produce a model that maps the relationship between inputs and outputs whilst keeping the weights small. However, introducing regularization will lead to an additional hyper-parameter which require careful tuning.

Neural networks are highly dependent on the set of hyper-parameters, and the accuracy will fluctuate under different sets of hyper-parameters. One of the most important hyper-parameter is the learning rate. Learning rate is the hyper-parameter which decides the adjustment of weights with respect to the loss gradient:

$$w = w - lr\frac{\partial}{\partial w}J(w) \tag{4}$$

where lr denotes the learning rate, J - the objective function and w - the weights. Choosing a learning rate which is too small will result in slower convergence and increase the run-time drastically if the data set is large. Besides, the algorithm can get stuck in local minimums which leads to poor generalization. If the learning rate is set too high, the gradient descent can overshoot the minimum and fail to converge [Zhang et al., 2019].

To minimize the objective function it is also necessary to choose a suitable optimization algorithm. Stochastic Gradient Decent(SGD) is the core technique behind optimizing the objective function in a neural network. The gradient provides information about the direction the function should take to reach the minimum. However, a problem with SGD is that the step size is equal for all parameters. Several attempts have been made to find a more adaptable optimizer based on SGD. Adaptive Moment Estimation (Adam) is one of the most popular methods and unlike SGD, it uses an adaptive learning rate for each parameter. RMSProp maintains per-parameter learning rates which are based on the average of the magnitude of the weight gradients. RMSProp also performs well on non-stationary problems and noisy problems [Basha et al., 2018].

2.2.2 Neural Network Extrapolation

A major drawback of neural networks is that they show poor performance for extrapolation tasks. Making accurate predictions with a neural network is possible as long as the input range does not extend the

input range the network was trained upon. As to say, if the network was trained to recognize digits and is then presented to a picture of a car, the network will return a digit. This is because the network does not directly learn function nor the characteristics of the function, instead it simply maps the input to the output. Previous work has found that adding nodes to the hidden layers increases accuracy, while additional layers in extrapolation tasks did not increase the network's ability to extrapolate. This suggests that changing the neural network architecture might have a minimal impact on how well the neural network performs extrapolation [Haley and Soloway, 1992]. For regions outside the training range, the output of a neural network is not reliable and will often produce arbitrary results. As shown in Figure 2 a neural network has been taught the function $f(x) = 1 - cos(x)$. Within the training range the neural network map the relationship accurately, while for the data set which was outside the trained scope resulted in arbitrary predictions [Jhedengren, 2018].

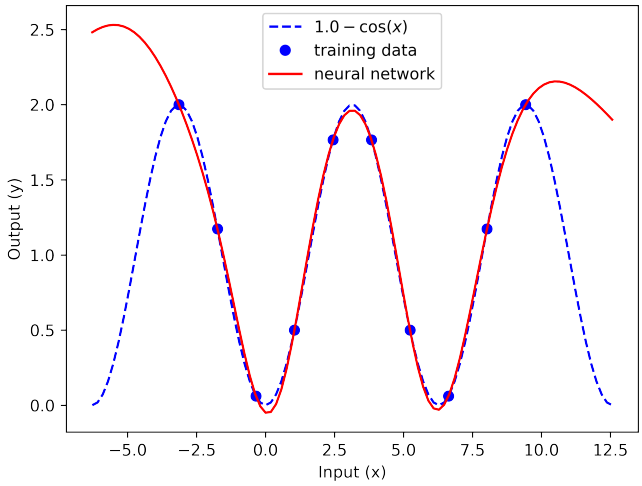


Figure 2: Results of training an artificial neural network on 10 data points. The neural network is able to interpolate but does not perform well in extrapolation [Jhedengren, 2018].

However, the neural network can, to some extent perform extrapolation in some regions which is close to the training set. [Barnard and Wessels, 1992]. Quality control is an area where neural network extrapolation has been applied. In quality control, one does not necessarily know all the possible scenarios which can lead to failure. Therefore it is impossible to create a training set witch fully characterizes the problem. Extrapolation can be viewed as a more general task, and for many applications like VFM, it would be advantageous is extrapolation could be performed. A severe challenge with extrapolation is that the required number of training samples increases exponentially with input space dimensionality. In contrast, interpolation can successfully be solved even though the input dimensionality is large

[Barnard and Wessels, 1992].

There have been suggested a variety of methods to improve the extrapolation capabilities of a neural network. A method which showed promising results was to manipulate the standardization range for input and outputs. By setting maximum and minimum values of a possible range could prevent saturation of the activation function for each node in the network [Varoonchotikul, 2003] However, this requires that the training and testing sets are known before training the network which will be disadvantageous for an online system. There also different techniques such as covariate shift, where the independent input variables are manipulated to overlap. Concept shift, where the relationship between the independent and target variables are shifted. In Prior Probability Shift, the target variables in the training and validation data are shifted to mutually overlap [Quionero-Candela et al., 2009].

2.2.3 Bayesian Hyperparameter Tuning

Machine learning models frequently involve tuning of model hyperparameters. Neural network, especially, requires careful tuning of all its hyper-parameters. If the hyperparameters are tuned poorly, the performance will be significantly affected. One way of finding the hyper-parameters is by hand-tuning. This can become extremely time-consuming, besides, the optimal set of parameters are often counter-intuitive. Grid-search is the most widely used method for finding the optimal parameters. The hyperparameters are found by iterating through every combination for a given set of parameters. By testing all the possible combinations in the search space the method will consequently eventually find the optimal set of hyperparameters. However, grid search will suffer from being an extremely time consuming process because the number of joint values grows exponentially with the number of hyperparameters. Also, it does not take into account the fact that some hyperparameters will have a less significance on the performance. Random search is also a common way of tuning the hyper-parameters and works by selecting the tested sets of hyperparameters randomly. It is similar to grid-search, but yet is has proven to yield slightly better results [Yakowitz and Lugosi, 1990].

Recently, Bayesian hyper-parameter optimization has been proven to be a superior alternative. Instead of testing every combination of the search space, the Bayesian optimization will select the next set of hyperparameters based on the most promising alternative. Bayesian hyper-parameters optimization is, in other words, a more intelligent method compared to grid-search and random-search [Lévesque et al., 2016]. The method is based on a probabilistic model where the objective is to find the best set of hyperparameters, x^*, that minimize the function f(x) for a given probability distribution M_{GP},

$$x^* = \underset{x}{\operatorname{argmin}} f(x \mid M_{GP}) \tag{5}$$

The idea behind Bayesian hyperparameter optimization is to use a surrogate model, M_{GP}, which approximates the target function, f(x), over the domain space, where x denotes a set of hyperparameters [Lévesque et al., 2016]. The surrogate model can be viewed as an approximation of the objective function, and is used because the true function is expensive to evaluate. The Gaussian process regression model, M_{GP}, will serve as the surrogate model and starts by specifying a normal prior distribution over the whole search space. The distribution will have a corresponding variance and mean which is typically zero. As more hyperparameters gets explored, the surrogate will map the corresponding variance for the whole search space [Snoek et al., 2012].

As we are interested in selecting the next set of hyperparameters more intelligently, an acquisition function,f(x | M_{GP}), is used. The acquisition function is used for proposing a new set of hyperparameters which is likely to improve the accuracy of the machine learning model based on the posterior distribution, M_{GP}. One of the most popular acquisition function is the expected improvement(EI) and is defined as,

$$f(x \mid M_{GP}) = \mathbb{E}[max(0, f' - f(x))] \tag{6}$$

Where f' is the best point for f(x) observed so far. Since the distribution is normal, Equation 6 can be computed analytically. The expected improvement is in general high for points we expect to be good and for points which have a high uncertainty while it is generally low for points which have already been evaluated. The acquisition function also serves as a balance between exploitation and exploration. In other words, if the function should either focus on exploring larger parts of the search space or exploit smaller regions. If the exploration-exploitation trade-off is too large, the acquisition function spends most of its time exploring areas that have a small expected improvement thus searching very inefficient. If the exploration-exploitation trade-off is set too small, the probability of getting stuck in sub-optimal peaks is significant which will lead to a non-optimal set of hyperparameters [Lévesque et al., 2016].

2.3 First Principle Models

In this section, the three models which are used in combination with the neural network are presented. Two choke models will be evaluated, the Bernoulli choke model and Al-Safran choke model. In terms of hybrid-machine learning, the Bernoulli choke equation will serve as the simplest description for the flow over the choke. On the other hand, Al-Safran will represent a more accurate model for calculating the rates. Moreover, the "No pressure wave" tubing equation will be presented and will be used for estimating the flow of oil and gas in the wellbore.

2.3.1 Bernoulli Choke Model

Bernoulli equation describes a one dimension flow, single phase, incompressible, steady and frictionless flow along a streamline. By the conservation of momentum Bernoulli equation can be written as a function of two points on a streamline [Ghiaasiaan, 2019],

$$P_1 - P_2 = \frac{\rho}{2}(U_2^2 + U_1^2) \tag{7}$$

Where P_1 and P_2 denotes the pressure, - U_2 and U_1 the velocity, at two points along a streamline. ρ denotes the density of the phase.

By substituting $U=\frac{Q}{A}$, we can express the equation in terms of volumetric flow rate [Yoon et al., 2006],

$$Q = C_d A_1 \sqrt{\frac{2(P_1 - P_2)}{\rho\left(\left(\frac{A1}{A2}\right)^2 - 1\right)}} \tag{8}$$

Where A_1 denotes the pipe cross sectional area before the choke, A_2 - the choke throat area and C_d - the discharge coefficient.

The discharge coefficient is used for correction of the rate and is defined as the ratio of the actual discharge to the theoretical discharge. For multiphase flows, the discharge coefficient will be dependent on whether the flow is critical or subcritical. However, since the discharge coefficient is a function of the choke opening it is possible to simply make it equal to C_{op} [Bikmukhametov and Jäschke, 2020b]. Also, since the flow consists of two phases, ρ is substituted by the mixture density, ρ_{mix}. In order to express the flow rate in terms of known variables, P_1 and P_2 can be substituted by the wellhead pressure upstream and downstream, P_{WHCU} and P_{WHCD}. Further, the ratio of the cross-sectional areas can be substituted by

the inverse of the choke opening,

$$Q = C_{op} A_1 \sqrt{\frac{2(P_{WHCU} - P_{WHCD})}{\rho_{mix}\left(\frac{1}{C_{op}^2} - 1\right)}} \tag{9}$$

Despite its simplicity, it captures the main dynamics of choke flow. Since the choke model was developed for a single-phase flow, it has been suggested several methods in the literature to correct the equation, but since the exact solution is not as important for the methods considered, it will not be included. A severe limitation of the Bernoulli equation is that the flow can be overestimated in the presence of gas, and could be disadvantageous for describing the qualitative solution [Haug, 2012].

2.3.2 Al-Safran Choke Model

Al-Safran and Kelkar developed a two-phase flow slip model for describing the multiphase flow across a choke. The equation is based on the work of Sachdeva et al. (1986) and Perkins (1993). The model is based on a one-dimensional balance equation of mass, momentum, and energy and assumes constant quality and incompressible liquid phase. The model can deal with both critical and sub-critical flows, and takes into account the slippage phenomenon between the gas and liquid phase at the choke throat which contributes to a more accurate description of critical flow rates [Al-Safran and Kelkar, 2009]. The selection of the slip model will depend on whether the flow is critical or subcritical. However, since the flows investigated in this report are in the critical region, only the relevant equations will be presented.

The mass flow can be obtained by Equation (10),

$$\dot{m}^2 = \frac{C A_2^2 p_1 \left(\alpha(1 - r_c) + \frac{n}{n-1}(1 - r_c^{\frac{n-1}{n}})\right)}{x_g v_{g1}\left(r_c^{-\frac{1}{n}} + \alpha\right)^2 \left(x_g + \frac{1}{R}(1 - x_g)\right)} \tag{10}$$

where r_c denotes the critical pressure ratio, C - a correction factor, A_2 - the choke throat area, p_1 - the upstream pressure, R - the slip ratio, n - the polytropic gas-expansion exponent, x_g - the gas quality or mass fraction, ν_{g1} - the upstream gas specific volume, and α - a hypothetical parameter defined as,

$$\alpha = \frac{R(1 - x_g)\nu_L}{x_g \nu_{g1}} \tag{11}$$

where ν_L denotes the liquid specific volume.

The critical pressure ratio, r_c, is defined by Equation (12), and requires an iterative procedure in order

to be solved

$$r_c^{1-\frac{1}{n}} = \frac{\alpha(1-r_c)+\frac{n}{n-1}}{\frac{n}{n-1}+\frac{n}{2}(1+\alpha r_c^{\frac{1}{n}})^2} \tag{12}$$

The polytropic gas-expansion exponent can be calculated by Equation (13).

$$n = \frac{x_g k C_{vg} + (1-x_g)C_L}{x_g C_{vg} + (1-x_g)C_L} \tag{13}$$

where C_{vg} denotes the gas specific heat constant at constant volume, C_L - the liquid specific-heat constant and k - the gas specific-heat ratio.

The calculation of the slip ratio, R, depends on whether the flow is critical or sub-critical, and the critical slip ratio is defined by Equation (14)

$$R = \sqrt{1 + x_g\Big(\frac{\rho_L}{\rho_g}\Big)}(1 + 0.6e^{-5.0x_g}) \tag{14}$$

where ρ_L denotes the liquid density and ρ_g - the gas density.

2.3.3 No-pressure-wave(NPW) Tubing Model

The Drift-Flux model is one of the most commonly used multiphase models used, and is based on mass and momentum balances as well as slip relations. Although widely used, finding efficient numerical solutions for the dynamic formulation of the model is considered challenging. Simplifications such as assuming a quasi-equilibrium momentum balance and neglecting the effect of acoustic waves have resulted in what is called "reduced DMF" or "No Pressure Wave"(NPW) models [Ambrus et al., 2016]. The NPW equation is given as,

$$\frac{dP}{dl} = \frac{\xi_{mix}\rho_{mix}U_{mix}^2}{2gD_{tubing}} + \rho_{mix}gsin\theta \tag{15}$$

where P denotes the fluid pressure, ξ_{mix} - the friction factor, U_{mix} - the mixture velocity, D_{tubing} - the tubing diameter, l - the pipe axial coordinate and θ - the pipe inclination.

To solve Equation 15 the equation must be discretized and solved numerically. However, since the qualitative solution is most relevant for machine learning purposes, a simplified model is desirable. By

averaging over the fluid properties over the pipe axial coordinate and integrating an analytical solution can be obtained [Bikmukhametov and Jäschke, 2020b],

$$Q_{mix}^{tubing} = \sqrt{\frac{(P_{BH} - P_{WH} - \bar{\rho}_{mix} g H)\pi^2 D_{tubing}^5 g}{8\xi_{mix} L_{tubing} \bar{\rho}_{mix} sin\theta}} \tag{16}$$

Relevant notation has been introduced in Equation 16, where P_{BH} and P_{WH} is the bottomhole and wellhead pressure. L_{tubing} is the total length of the wellbore.

The average density can be found by,

$$\bar{\rho}_{mix} = \frac{\rho_{mix,WH} + \rho_{mix,BH}}{2} \tag{17}$$

If the friction factor is assumed to constant the solution can be solved analytically. If not, the friction factor will be dependent on the Reynolds number and can be found by solving the Colebrook equation with an iterative method [Colebrook, 1939],

$$\frac{1}{\sqrt{\xi}} = -2log_{10}\Big(\frac{\epsilon}{3.7 D_{tubing}} + \frac{2.51}{Re\sqrt{\xi}}\Big) \tag{18}$$

Where ϵ is the relative roughness and Re is the Reynolds number.

3 Methods

This section describes the basis in which the training and validation sets were built upon such as well trajectories, inclinations, GOR, boundary properties, and measurement positions. Further on, the methods which will be considered in the case studies are also presented as well as the motivation for each method. It will also be provided additional information on how the fluid properties, used in the first principle models, were obtained.

3.1 Training and validation sets

The training and validation sets were produced from LedaFlow which is a simulation tool provided by Kongsberg and resolves multiphase hydrodynamics of oil, gas, and water along a pipeline in one dimension [Kongsberg, 2001]. LedaLink was used as a gateway between LedaFlow and Python to exchange boundary conditions, run simulation files and extract important information. All the wells are equipped with a wellbore, choke valve and a short flowline to measure the upstream properties. The data in the training set were produced from three different wells with different inclinations and composition which resulted in a specific GOR. All the wells in both the training set and validation set had the same reservoir and boundary pressures and temperatures and equal length. The specifications are presented in Table 1 and an illustration of the three wells used for training can be found in Figure 3a.

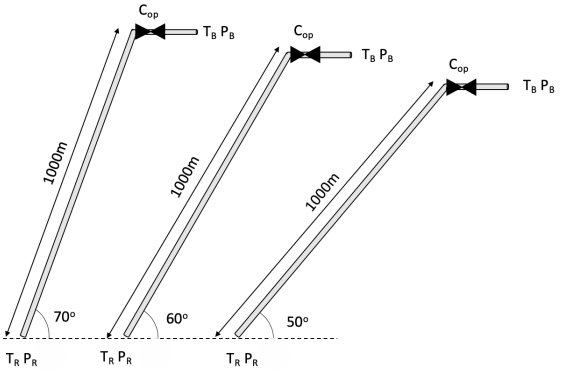
Table 1: Specifications for traning data. P_R and T_R is the reservoir pressure and temperature. P_B and T_B is the boundary pressure and temperature at flowline.

Well	Inclination[^{o}]	GOR[-]	P_R[Bar]	P_B[Bar]	T_R[C]	T_B[C]	Length[m]
1	60	20	250	40	76.85	6.85	1000
2	70	40	250	40	76.85	6.85	1000
3	50	60	250	40	76.85	6.85	1000

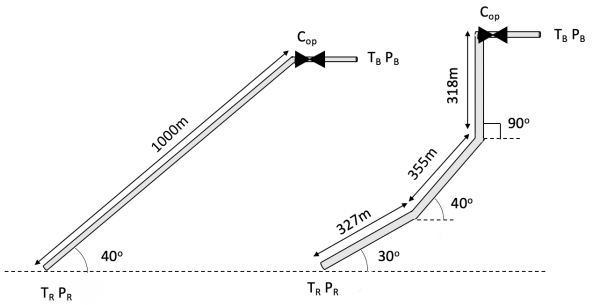
Since the objective is to investigate the extrapolation capabilities of a neural network-based virtual flow meter, the first validation set was designed with an inclination of 40° and GOR of 80 as shown in Table 17. Since a realistic well often will consist of different inclinations, the second validation set was constructed of three different sections. At the bottom of the well, the inclination is 40°, 50° at the middle part and 90° in the top. Similar to the validation set 1, GOR was specified to be 80. All the properties for the validation sets can be found in Table 17, and an illustration of the wells can be found in Figure 3b

Table 2: Specifications for validation data. P_R and T_R is the reservoir pressure and temperature. P_B and T_B is the boundary pressure and temperature at flowline.

Well	Inclination[°]	GOR[-]	P_R[Bar]	P_B[Bar]	T_R[C]	T_B[C]	Length[m]
1	40	80	250	40	76.85	6.85	1000
2	30, 40, 90	80	250	40	76.85	6.85	1000



(a) Illustration of the wells used to produce data for the training set.



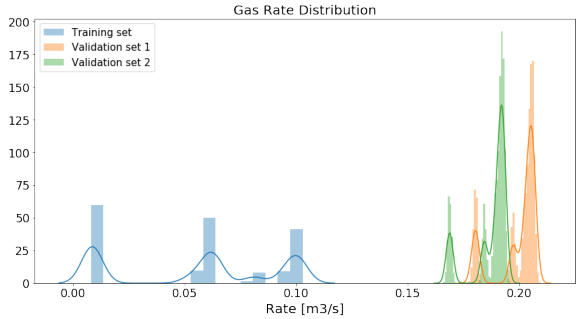
(b) Illustration of the wells used to produce data for the validation set.

In addition, an error was added to make the data set more realistic and was drawn from a normal distribution with a standard deviation of 0.05. All the measurements were logged at the exact same position for both the traning and validation sets. Measurements, given in meters from bottom of the well, are shown in Table 3.

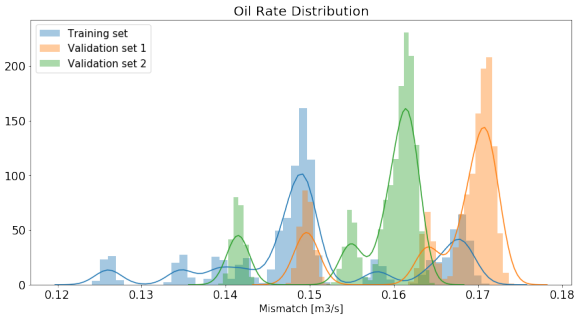
Table 3: Measurement positions given in meters from reservoir. BH denotes the bottom hole measurement, LM is the second measurement and UM is the third measurement after the bottom hole. CU and CD is the choke upstream and downstream measurements at the wellhead.

$Wellbore_{BH}$[m]	$Wellbore_{LM}$[m]	$Wellbore_{UM}$[m]	$Wellhead_{CU}$[m]	$Wellhead_{CD}$[m]
25	327	682	990	1010

At first sight, the training and validation set does not look too different. However, by closer inspection, the problem in terms of extrapolation reveals to be far more complicated. Figure 4a and 4b shows the distribution of the oil and gas rates. It is apparent that the oil rate is less affected by the GOR, while the gas rate is more sensitive and is almost twice as large in the validation set compared to the training set. The large deviation in gas rates could potentially provide a major challenge in predicting the correct quantitative rate.



(a) Gas rate distribution for the traning set and validation set 1 and 2.



(b) Oil rate distribution for the traning set and validation set 1 and 2.

3.2 Calculation of fluid properties

Multiflash was used for generating the PVT tables which was constructed from oil and gas compositions. Based on these compositions, Multiflash calculates properties under different pressure and temperature conditions. The compositions which created the basis for the PVT can be found in Appendix 18. The PVT tables were also used to find properties such as,

- Oil and gas density, ρ_{oil} and ρ_{gas}
- Oil and gas viscosity, μ_{oil} and μ_{gas}
- Oil and gas heat capacities, $C_{p,gas}$, $C_{p,oil}$
- Gas mass fraction, x_g

Based on the pressure and temperature measurements in the well, the properties were found by interpolation and were used to calculate the first principle model equations. It is important to note that the properties will change as the reservoir depletes. Therefore, the exact fluid composition will not always be available and could potentially be a source of error if data was measured when the compositions were unknown. However, since the data for this project was produced synthetically, it is assumed that the compositions are known.

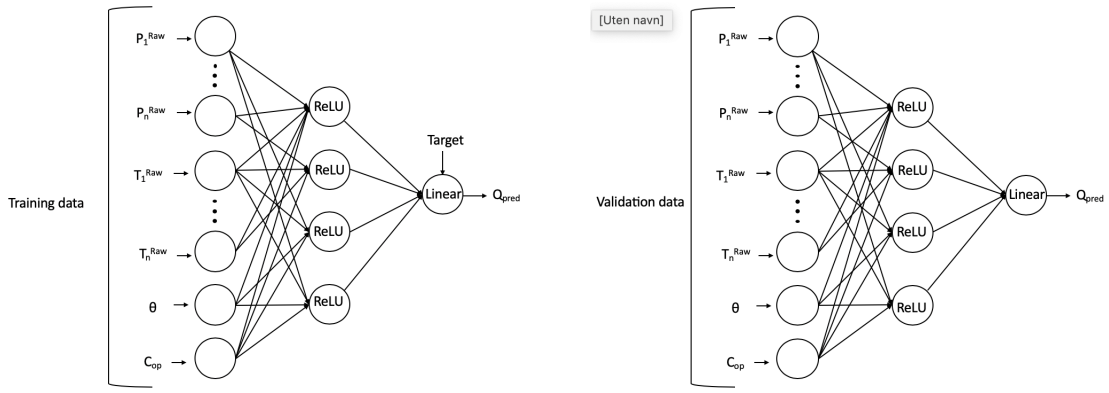
3.3 Neural network methods

The methods are based on the work done by Bikmukhametov and Jäschke(2020), which proposed several techniques to improve the explainability and transparency of the machine learning model. It was found that these approaches improved the performance compared to using the raw measurements directly. The idea is the transform the raw data into more physically meaningful features, which can represent the underlying problem more exact and potentially improve the accuracy. It was also proposed to include the first principle solutions as an input which could improve the model's ability to capture the governing

dynamics of the system and quantitative rates. Moreover, it was also suggested to train the model on the mismatch between the true rate and the solution provided by the first principle model.

3.3.1 Baseline: Raw data

By now, the most common approach in Virtual Flow Metering is using the raw data directly as an input. The raw measurement includes pressures and temperatures at the bottomhole, downstream and upstream of the choke, inclination and choke opening. Figure 5a shows the training phase, where the neural network is trained to minimize the error on the target values, and Figure 5b shows the validation phase of the Baseline Model.



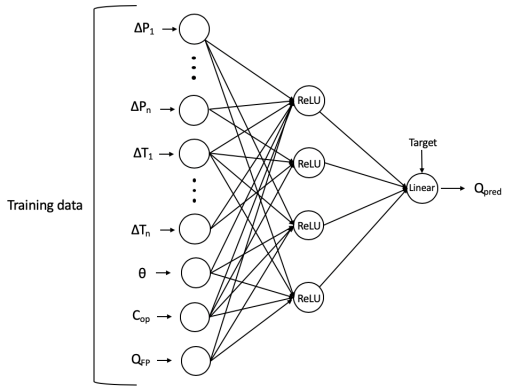
(a) Training phase of Baseline Model

(b) Validation phase of Baseline Model

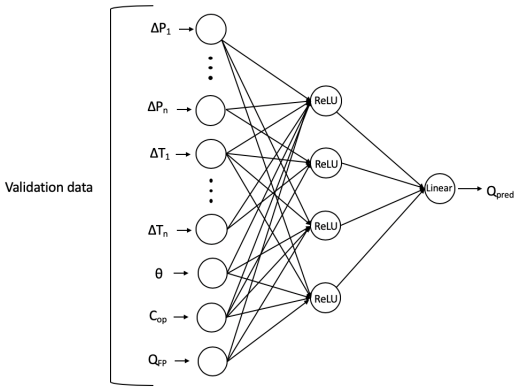
3.3.2 Method 1: Feature engineering

Instead of feeding the raw data directly into the machine learning model, the data can be transformed into features. The idea is the create features that optimize the information density thus reducing the input dimensionality and possibly represent the underlying pattern better. Also, introducing features may allow the neural network to discover and exploit relationships the raw data cannot directly provide. In addition, first principle model solutions can also be included. This form of data enrichment can improve the neural network's ability to capture the quantitative relationship of oil and gas rates, as well as supplementing information about the governing dynamics.

Figure 6a illustrates the training phase, and Figure 7b shows the validation phase of Method 1.



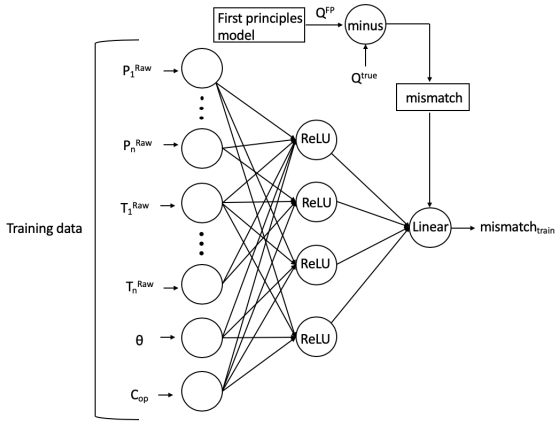
(a) Training phase of method 1



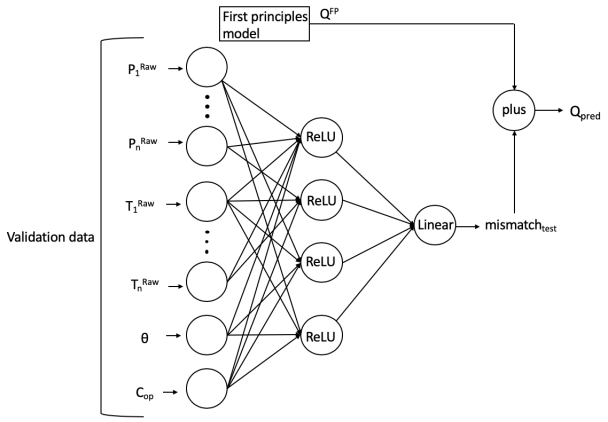
(b) Test phase of method 1

3.3.3 Method 2: Incorporating first principle model solutions into the target variable.

In Method 2 the target variable is the mismatch between the true rate and the first principle solution, as shown in Figure 7a. The idea behind using the mismatch as a target variable is that it can alter the target distribution thus reducing the extrapolation range. During the validation phase as shown in Figure 7b, the neural network predicts the mismatch. The sum of the first principle solution and the predicted mismatch will is then the prediction of the flow rate.



(a) Training phase of method 1



(b) Test phase of method 1

3.4 Tuning pipeline

In machine learning a data set is usually divided into a training, validation and test set. However, since the data is produced synthetically, the training and validation were produced separately and are therefore distinct and partition of the data set is unnecessary. Since the oil and gas rates may have a different correlation with the inputs the architecture and hyperparameters will consequently

differ. It was therefore used individual neural networks for oil and gas. The procedure proposed by [Bikmukhametov and Jäschke, 2020b] is used for tuning all the models. First, the neural network is trained with a fixed number of epochs and using Bayesian Hyper-parameter optimization, the set of optimal hyperparameters was obtained. Then the neural network was reinitialized and trained from scratch with early stopping to avoid overfitting.

4 Case studies

In total there were conducted ten case studies. The case studies are structured in a way such that each method systematically introduces new and increased features and first principle solutions.

The **Baseline Model** uses the raw data directly as an input without any pre-processing, as shown if Table 4. Feeding the raw data into the neural network represents the most common way of training a VFM, and will therefore so serve as the case, which is used to determine the relative improvement for all the case studies.

Case 1.1 if the first case in Method 1. The raw data were transformed into choke features, as shown in Table 4. To investigate how the performance is affected by only the features, no first principle model solutions are included. The true oil and gas rate were used as target variables.

Case 1.2 also transforms the raw data into choke features. In addition, the Bernoulli choke equation was included as an input as shown in Table 4. The idea is that the first principle model solutions can give additional information about the quantitative relationship of the oil and gas flow rates. The true oil and gas rate were used as target variables.

Case 1.3 is similar to Case 1.2 but considers the Al-Safran choke model solution as an input, as shown in Table 4. By using the Al-Safran choke model, it is possible to investigate how different first principle model solutions affects the performance.

Case 1.4 builds on Case 1.3. In addition to the features in Case 1.3, the pressure and temperature drop were also included as well as the "No Pressure Wave" tubing equation solutions, as shown in Table 4. The hypothesis is that by including more features, the neural network's ability to capture the underlying relationship for the system will increase.

Case 1.5 also builds on Case 1.4. However, the well was divided into three sections, bottom, middle, and top. Where each section has a corresponding pressure and temperature drop and first principle model solution obtained from the "No pressure Wave" equation as shown in Table 4

Case 2.1 is the first case that considers Method 2. As in the Baseline Model, the raw data were used directly as an input. The Bernoulli choke model was incorporated in the target variables, and the neural network was trained on the mismatch, as shown in Table 4. The idea is that by including first principle model solutions in the target variable, the distribution will shift, thus increasing the characteristics of interpolation.

Case 2.2 is similar to Case 2.1. Instead, the Al-Safran was integrated into the target variable as shown in Table 4. By using the Al-Safran choke model, it is possible to investigate how different first principle

model solutions affect the transformed target distribution.

Case 2.3 considers the integration of the "No pressure wave" tubing equation into the target variable. The raw data was used directly as an input, and the friction factor is assumed to be constant, as shown in Table 4.

Case 2.4 is conceptually identical to Case 2.3, except for that the friction factor is assumed to be a function of the Reynolds number, as shown in Table 4. Case 2.4 will examine how the friction factor affects the performance and the transformed distributions.

Table 4: Summary of case studies conducted.

Case	Input	First principle model	Training target
Baseline	P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	not used	Q_{oil}, Q_{gas}
1.1	ΔP_{choke}, ΔT_{choke}, C_{op}, θ	not used	Q_{oil}, Q_{gas}
1.2	ΔP_{choke}, ΔT_{choke}, C_{op}, θ, $Q^{Bernoulli}_{choke,gas}$, $Q^{Bernoulli}_{choke,oil}$	not used	Q_{oil}, Q_{gas}
1.3	ΔP_{choke}, ΔT_{choke}, ΔP_{tubing}, ΔT_{tubing},C_{op}, θ, $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$	not used	Q_{oil}, Q_{gas}
1.4	ΔP_{choke}, ΔT_{choke}, ΔP_{tubing}, ΔT_{tubing}, C_{op}, θ $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$, $Q^{NPW}_{tubing,oil}$, $Q^{NPW}_{tubing,gas}$	not used	Q_{oil}, Q_{gas}
1.5	ΔP_{choke}, ΔT_{choke}, C_{op}, $\Delta P_{tubing,bottom}$, $\Delta P_{tubing,middle}$ $\Delta P_{tubing,top}\theta$ $\Delta T_{tubing,bottom}$, $\Delta T_{tubing,middle}$, $\Delta T_{tubing,top}$, $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$, $Q^{NPW}_{bottom,tubing,gas}$ $Q^{NPW}_{bottom,tubing,oil}$, $Q^{NPW}_{middle,tubing,oil}$, $Q^{NPW}_{middle,tubing,gas}$, $Q^{NPW}_{top,tubing,oil}$, $Q^{NPW}_{top,tubing,gas}$	not used	Q_{oil}, Q_{gas}
2.1	P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Bernoulli choke model	$Q^{mismatch}_{oil}$, $Q^{mismatch}_{gas}$
2.2	P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Al Safran choke model	$Q^{mismatch}_{oil}$, $Q^{mismatch}_{gas}$
2.3	P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Tubing model ξ = const	$Q^{mismatch}_{oil}$, $Q^{mismatch}_{gas}$
2.4	P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Tubing model ξ = f(Re)	$Q^{mismatch}_{oil}$, $Q^{mismatch}_{gas}$

Since the extent of the case studies is comprehensive, the presentation of the results and discussion will be structured in the following way:

- Case studies
    1. Summary of method
    2. Results
    3. Discussion regarding the performance of the particular case, compared to the previous methods. For Method 2, the distribution before and after the transformation of target variables will be presented, as well as the Pearson correlation matrix for validation set 1.
- Results summary
- Overall discussion and analysis about general performance in Method 1 and 2, and weaknesses in the methodology.

4.1 Baseline Model

In the Baseline model the raw data is used directly as an input. A summary of the input feature and target variables can be found in Table 5.

Table 5: Inputs, first principle models and traning targets for Baseline Model.

Input	First principle model	Training target
P_{BH}, T_{BH}, $_{WHCU}$, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	not used	Q_{oil}, Q_{gas}

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 8 and 9 for validation set 1 and 2, respectively.

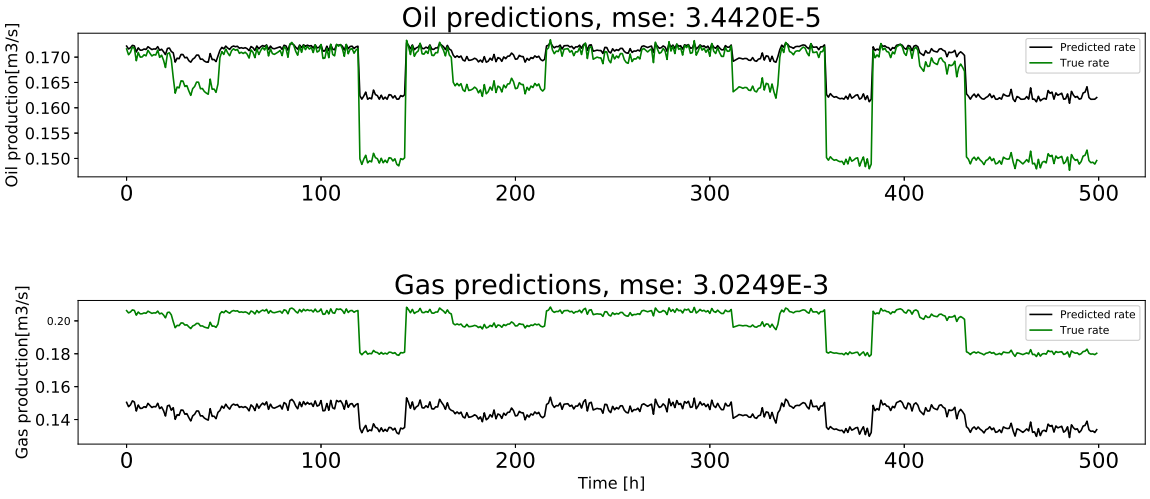


Figure 8: Validation set 1: Oil and gas predictions for Baseline Model.

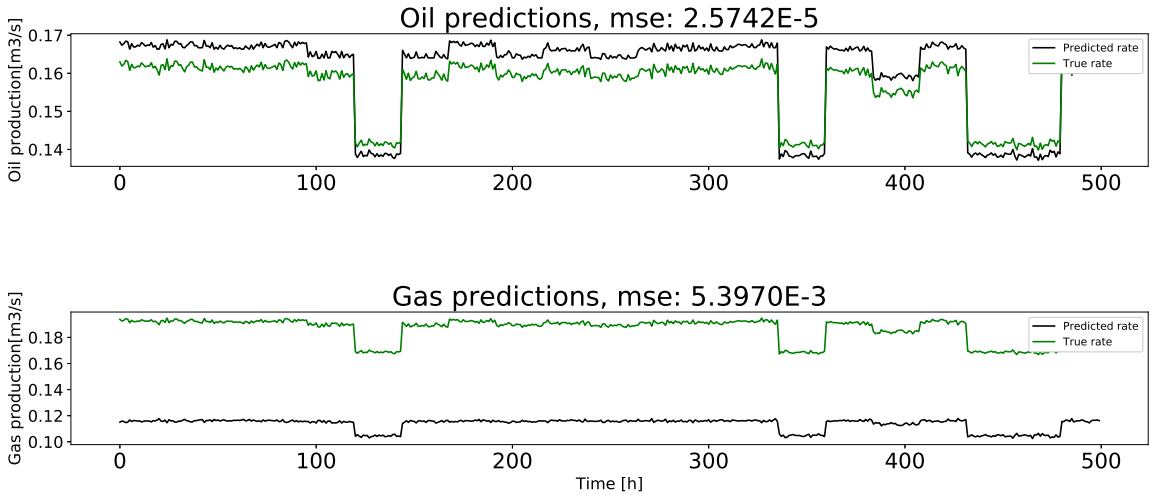


Figure 9: Validation set 2: Oil and gas predictions for Baseline Model.

From Figure 8 and 9, it is clear that the model captures the governing dynamics of the system. However, the predictions are strongly biased, especially for the gas rate. The bias in the gas predictions is presumably a result of the limited exploration capabilities of a neural network. Since the gas rate is the most affected by an increased GOR, as shown in Figure 4a, the predictions will be more demanding in terms of extrapolation.

The anticipated result was that the overall accuracy of the validation set 2 well would yield a lower accuracy due to a more complex well trajectory. However, the oil prediction for validation set 2 is more accurate compared to the validation set 1. A possible explanation is that the average inclination of validation set 2(63°), is more similar to the inclinations in the training set. Therefore, the inclination drives the data in validation set 2 closer to the characteristics of interpolation.

4.2 Case 1.1

In Case 1.1, the raw data were transformed into physically meaningful features, more precisely, into the choke pressure and temperature drop. Feature engineering is motivated by the fact that transforming the raw data could represent the underlying problem more accurate, resulting in improved model accuracy on new data. A summary of the input feature and target variables can be found in Table 6.

Table 6: Inputs, first principle models and traning targets for Case 1.1.

Input	First principle model	Training target
ΔP_{choke}, ΔT_{choke}, C_{op}, θ	not used	Q_{oil}, Q_{gas}

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 10 and 11 for validation set 1 and 2, respectively.

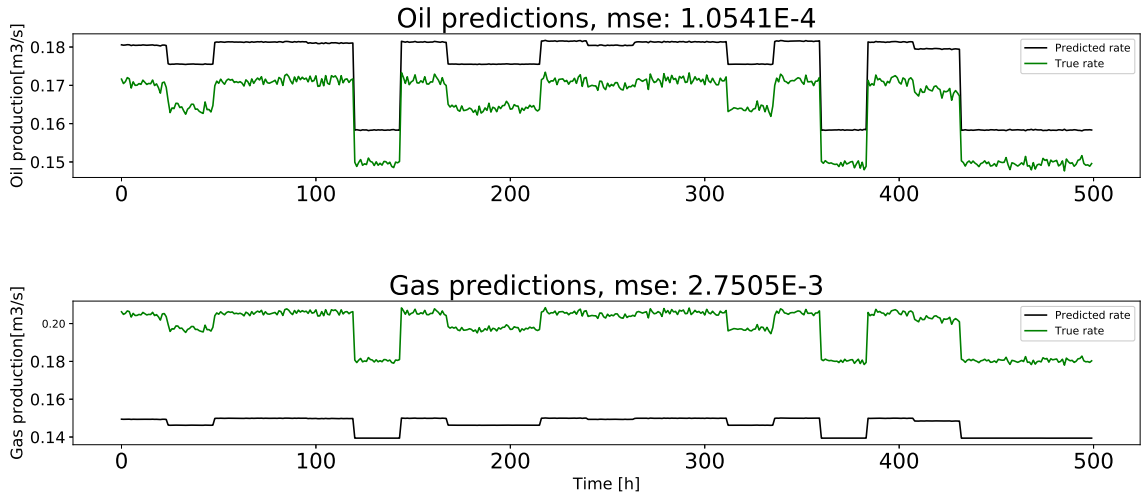


Figure 10: Validation set 1: Oil and gas predictions for Case 1.1

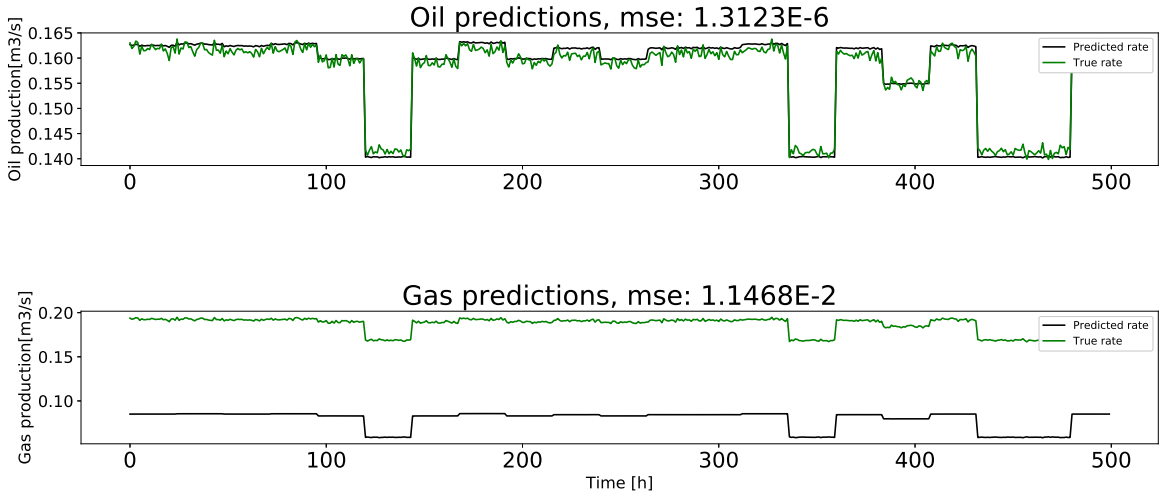


Figure 11: Validation set 2: Oil and gas predictions for Case 1.1

As discussed, by transforming the raw data into features can allow the neural network to exploit new relationships and improve dynamic behaviour. In other words, the relative change in the true flow due to change in choke position should be reflected in the predictions. This is confirmed for the oil rate prediction in validation set 2 where predictions follow the trend quite accurately. Similarly, even though the bias increased in validation set 1, the predicts are more consistent with the system dynamics.

At first, it was counter-intuitive that the oil predictions improved significantly for validation set 2, while

the error increased for validation set 1. Even though physically meaningful features were introduced, the bottom hole raw measurements were removed. Since the well in validation set 2 is more complex in terms of trajectory, it should intuitively require more features which are representative for the flow along the whole well. However, as previously suggested, the improved accuracy is because the average inclinations increase the characteristics of interpolation, thus increasing the performance.

Another conspicuous change is the major decrease of accuracy for the gas predictions in validation set 2. In the same way for the gas, the upstream pressure is also dependent on the inclination, thus the pressure drops over the choke for the training and validation set 2 will be of similar magnitude. Since the corresponding gas rates for the training set are small, the predicted gas rates for validation set 2 will, consequently, also be of the same magnitude. In conclusion, including only choke features will result in a more local model that is less able to generalize the inputs.

4.3 Case 1.2

Similarly to Case 1.1, the raw data were transformed into features describing the conditions over the choke. In addition, the first principle solution from the Bernoulli choke model was included as a feature. Including first principle solutions could provide additional information by capturing the governing choke dynamics and improve the quantitative behavior. A summary of the input feature and target variables can be found in Table 7.

Table 7: Inputs, first principle models and traning targets for Case 1.2

Input	First principle model	Training target
ΔP_{choke}, ΔT_{choke}, C_{op}, θ, $Q^{Bernoulli}_{choke,gas}$, $Q^{Bernoulli}_{choke,oil}$	not used	Q_{oil}, Q_{gas}

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 12 and 13 for validation set 1 and 2, respectively.

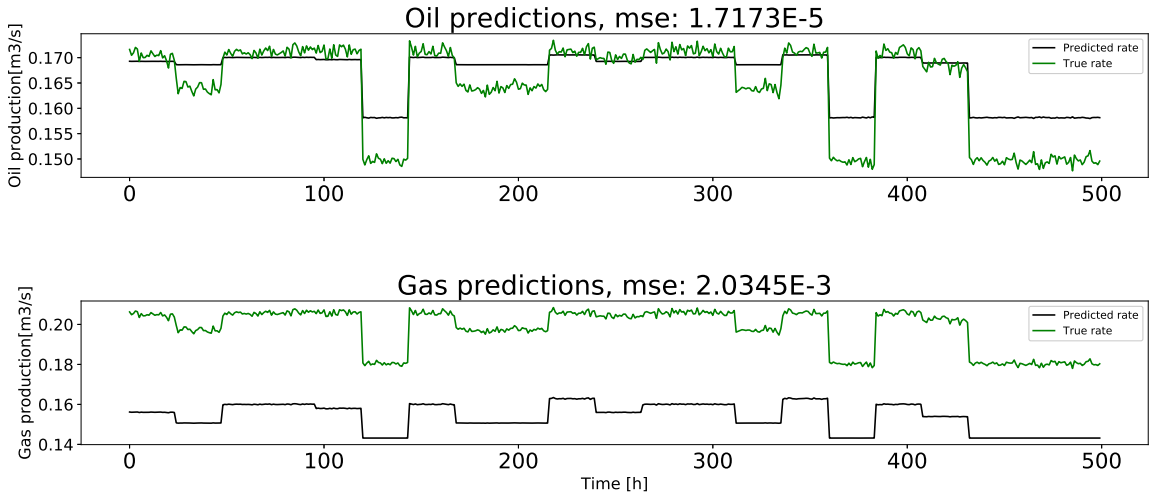


Figure 12: Validation set 1: Oil and gas predictions for Case 1.2

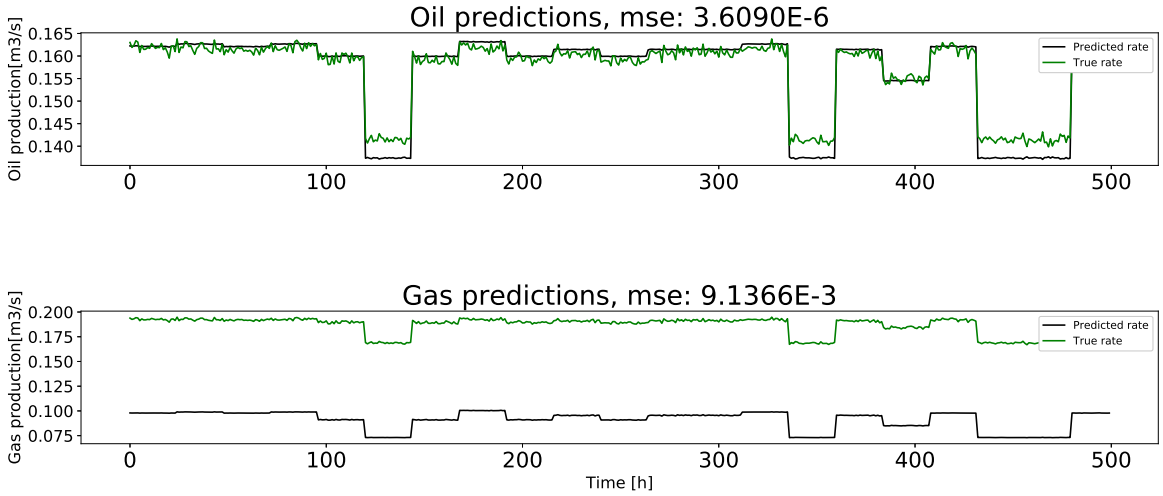


Figure 13: Validation set 2: Oil and gas predictions for Case 1.2

The most noticeable improvement is in oil predictions for validation set 1. Compared to Case 1.1, the predictions are more accurate in terms of the quantitative behavior, yielding the highest accuracy yet, which could support the hypothesis that including first principle solutions can improve the neural network's ability to predict rates of more precise magnitudes. However, the oil predictions for validation set 2 resulted in a larger error when including the first principle solution. Even though the deviation is small, it is expected that improvements in the oil rates accuracy should increase collectively.

There is an increased improvement for the gas predictions for validation set 1 and 2. Despite being marginal, the predictions are less biased and more accurate in terms of quantitative behavior. In other

words, the system dynamics for gas can be seen more clearly compared to the previous case. Using the solution of the Bernoulli choke model as an input will also give the neural network an indication of the gas quantity, which is reflected in an increased magnitude of the predictions.

4.4 Case 1.3

As in the previous cases, the raw data were transformed into choke features. In order to compare how different first principle models affects the performance, the Al-Safran choke model solution was included as a feature. A summary of the input feature and target variables can be found in Table 8.

Table 8: Inputs, first principle models and traning targets for Case 1.3

Input	First principle model	Training target
ΔP_{choke}, ΔT_{choke},C_{op}, θ, $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$	not used	Q_{oil}, Q_{gas}

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 14 and 15 for validation set 1 and 2, respectively.

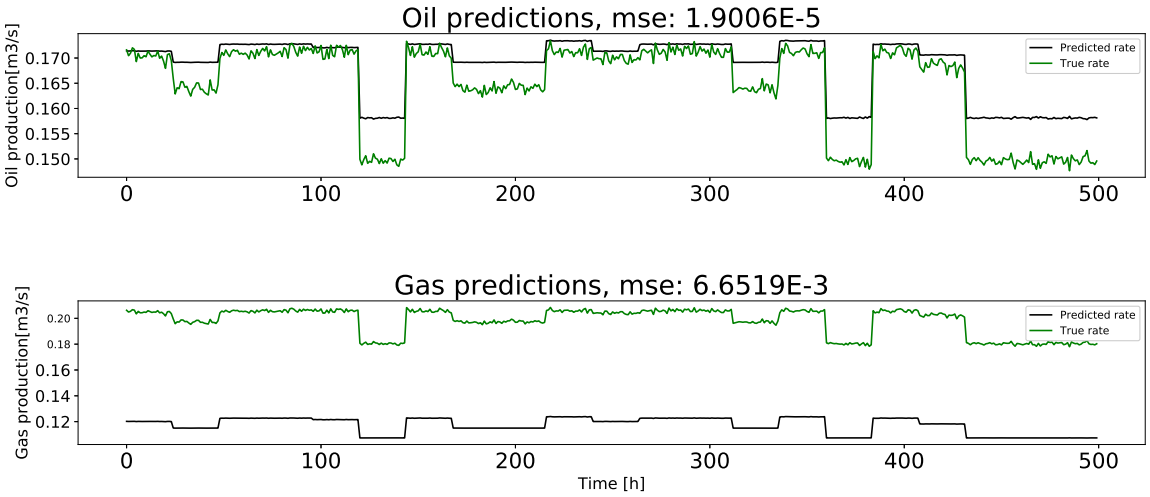


Figure 14: Validation set 1: Oil and gas predictions for Case 1.3

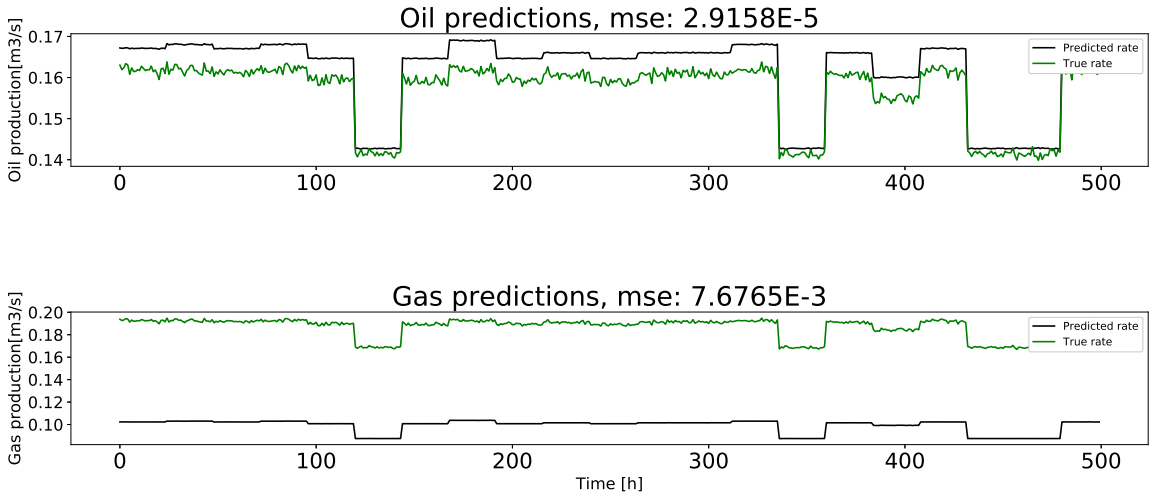


Figure 15: Validation set 2: Oil and gas predictions for Case 1.3

Including the solution of the Al-Safran choke model, yields similar accuracy compared to including the Bernoulli choke model solution. That aside, there is an overall reduction in performance. For the oil, the prediction is generally underestimated. The Al-Safran choke model will estimate the flow more accurate compared to the Bernoulli choke model, which again will make the inputs more qualitative accurate. In contrast, the Bernoulli choke model will overpredict the flow in the presence of gas. Since both validation sets have a larger GOR, Bernoulli will generally result in a larger estimated flow compared to Al-Safran. Since the neural network will map the trend *larger first principle model inputs yields larger outputs*, the Bernoulli equation will consequently result in predictions of larger magnitude. While on the other hand, the Al-Safran choke model will yield more accurate inputs, but does not have the same impact as the Bernoulli choke equation.

With this in mind, it makes sense that the predictions are generally underpredicted. This deviation is especially apparent for the gas predictions in validation set 1 where the accuracy was three times higher when using the Bernoulli choke model solution as an input. However, the deviation is not apparent for the gas predictions in validation set 2, where Case 1.2 and 1.3 yielded an almost identical accuracy.

However, the Bernoulli choke model might only benefit this particular case when the validation targets of gas are larger compared to the training set. Therefore, it cannot be concluded that using the Bernoulli choke model is more beneficial compared to Al-Safran.

4.5 Case 1.4

In Case 1.4 the raw data were transformed into choke and tubing features. Since wells often differ in trajectory and inclination the neural network might benefit from this additional information. Further on, the solutions from the NPW equation were also included as an input feature. A summary of the input feature and target variables can be found in Table 9.

Table 9: Inputs, first principle models and traning targets for Case 1.4

Input	First principle model	Training target
ΔP_{choke}, ΔT_{choke}, ΔP_{tubing}, ΔT_{tubing}, C_{op}, θ $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$, $Q^{NPW}_{tubing,oil}$, $Q^{NPW}_{tubing,gas}$	not used	Q_{oil}, Q_{gas}

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 16 and 17 for validation set 1 and 2, respectively.

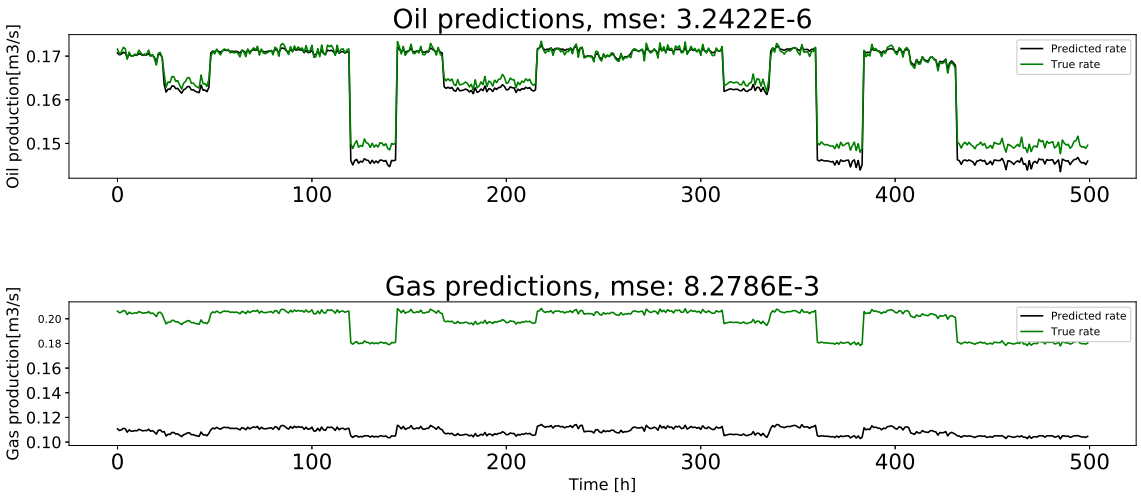


Figure 16: Validation set 1: Oil and gas predictions for Case 1.4

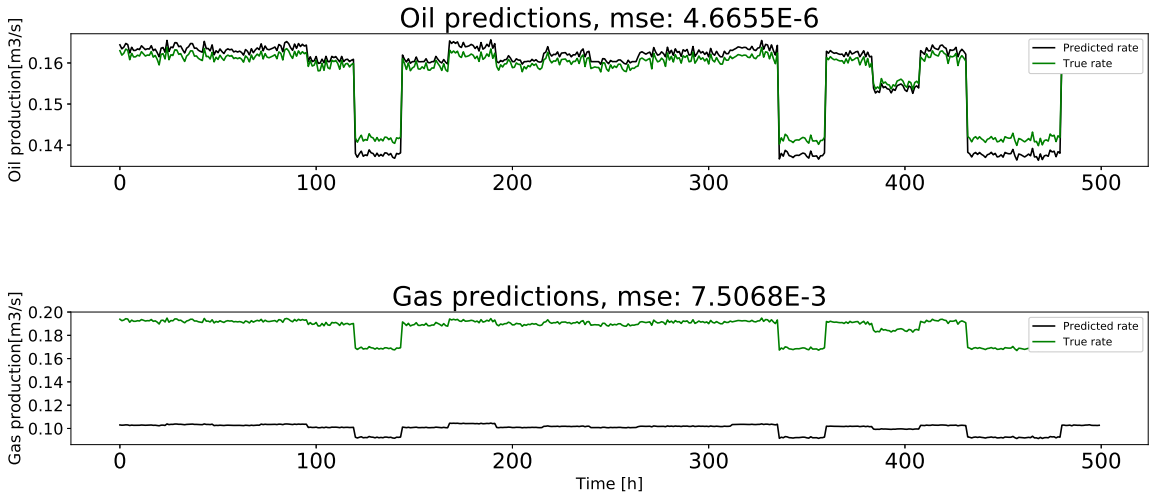


Figure 17: Validation set 2: Oil and gas predictions for Case 1.4

Including the tubing features yielded the highest accuracy for the oil predictions for validation set 1 so far. The performance of the oil predictions in validation set 2 also increased compared to the previous case where no tubing features were included. In general, the oil predictions precisely captures the dynamics of the choke as well as having a minimal bias. The could confirm that including physically meaningful features as well as first principle solutions for different parts of the system increases the neural network's ability to estimate the flow more accurately.

On the contrary, the increased accuracy of the oil rate prediction is not reflected in the gas predictions. This result suggests that the gas rate may be less correlated with the tubing features. It can be because tubing features are too general such that the underlying pattern is too vague to be correlated with the gas predictions. Nevertheless, the gas prediction in validation set 2 yielded a marginal improvement, which could support the hypothesis that a well with complex trajectory requires features which describe the system more comprehensively.

4.6 Case 1.5

As in the previous case, the raw data were transformed into choke and tubing features. For Case 1.5, the well was divided into three sections, bottom, middle and top with corresponding pressure and temperature drops. In addition, the first principle solution for the different parts of the wells were incorporated as inputs. A summary of the input feature and target variables can be found in Table 10.

Table 10: Inputs, first principle models and traning targets for Case 1.5

Input	First principle model	Tra
ΔP_{choke}, ΔT_{choke}, C_{op}, $\Delta P_{tubing,bottom}$, $\Delta P_{tubing,middle}$ $\Delta P_{tubing,top}\theta$ $\Delta T_{tubing,bottom}$, $\Delta T_{tubing,middle}$, $\Delta T_{tubing,top}$, $Q^{AlSafran}_{choke,gas}$, $Q^{AlSafran}_{choke,oil}$, $Q^{NPW}_{bottom,tubing,oil}$, $Q^{NPW}_{bottom,tubing,gas}$ $Q^{NPW}_{middle,tubing,oil}$, $Q^{NPW}_{middle,tubing,gas}$, $Q^{NPW}_{top,tubing,oil}$, $Q^{NPW}_{top,tubing,gas}$	not used	

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 18 and 19 for validation set 1 and 2, respectively.

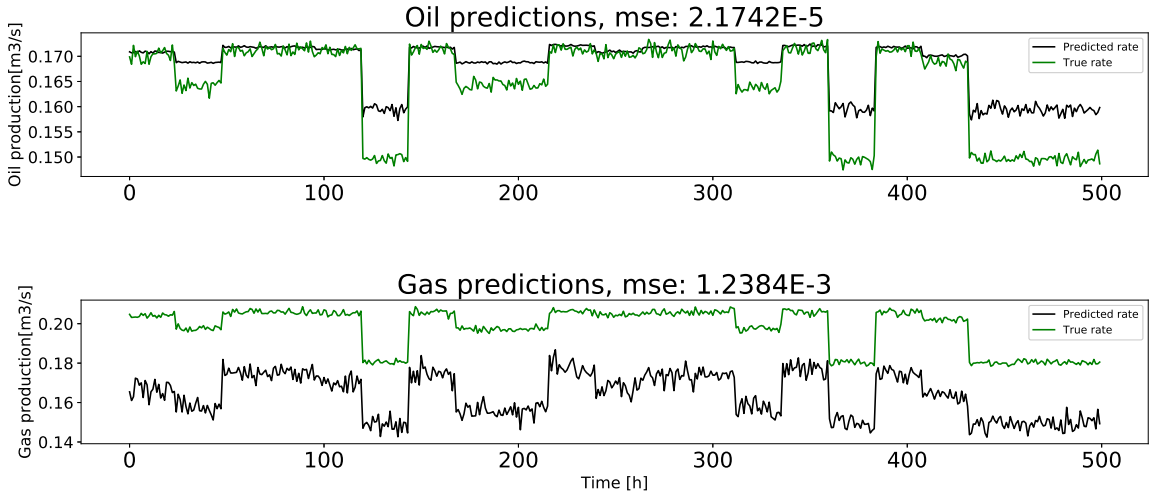


Figure 18: Validation set 1: Oil and gas predictions for Case 1.5

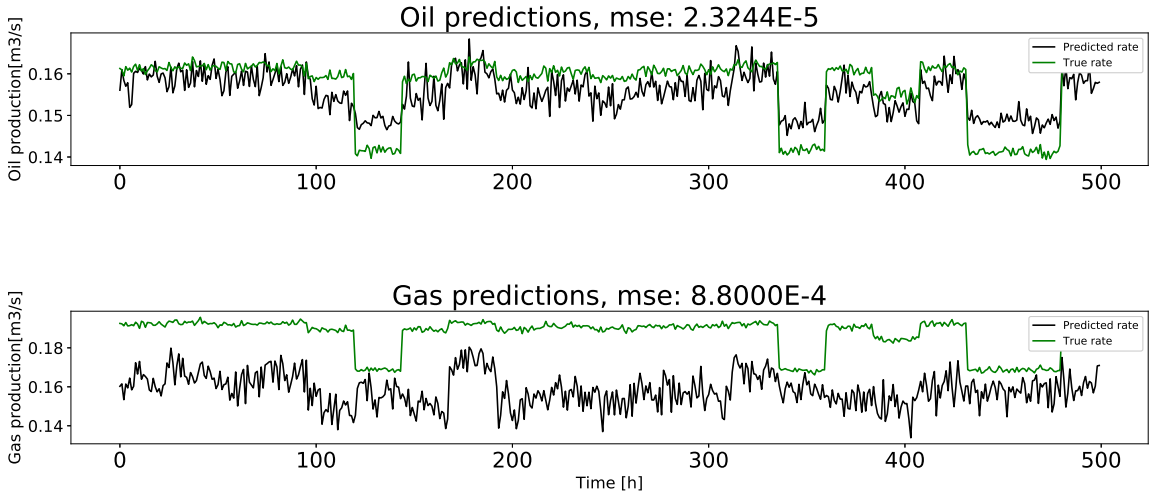


Figure 19: Validation set 2: Oil and gas predictions for Case 1.5

Case 1.5 yielded the most drastic change in terms of accuracy. While in the previous cases the gas predictions were characterized by a high bias and large error, Case 1.5 resulted in a large improvement in the overall accuracy. Despite the noticeable variance, the gas predictions are now resembling the true rate more accurately compared to the previous cases. By introducing features from different parts of the well, the neural network is trained on the pressure and temperature development along the well. This could increase the neural network's ability to distinguish different trajectories and how it affects the overall gas rate. However, measurements along the well are usually not available in a real system, which makes the case less realistic.

On the other hand, the oil predictions overall decreased in accuracy compared to the previous cases. This suggests that the total oil rate is less correlated with the individual flow solutions and corresponding pressure drops along the well. In general, including more features which has less significance to the target variable can make the neural network learn incorrect behaviour of the system. In addition, the neural network will also require more time to filter out unnecessary information which can disturb the training process.

4.7 Case 2.1

In Case 2.1, Method 2 was investigated with the Bernoulli choke model as first principle solution. Unlike Method 1, the raw data is now used directly as an input. The target variables are the mismatch between the actual gas rate and the Bernoulli choke model, as proposed in Method 2. A summary of the input feature, target variables, and first principle solutions is shown in Table 11.

Table 11: Inputs, first principle models and traning targets for Case 2.1

Input	First principle model	Training target
P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Bernoulli choke model	$Q_{oil}^{mismatch}$, $Q_{gas}^{mismatch}$

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 20 and 21 for validation set 1 and 2, respectively.

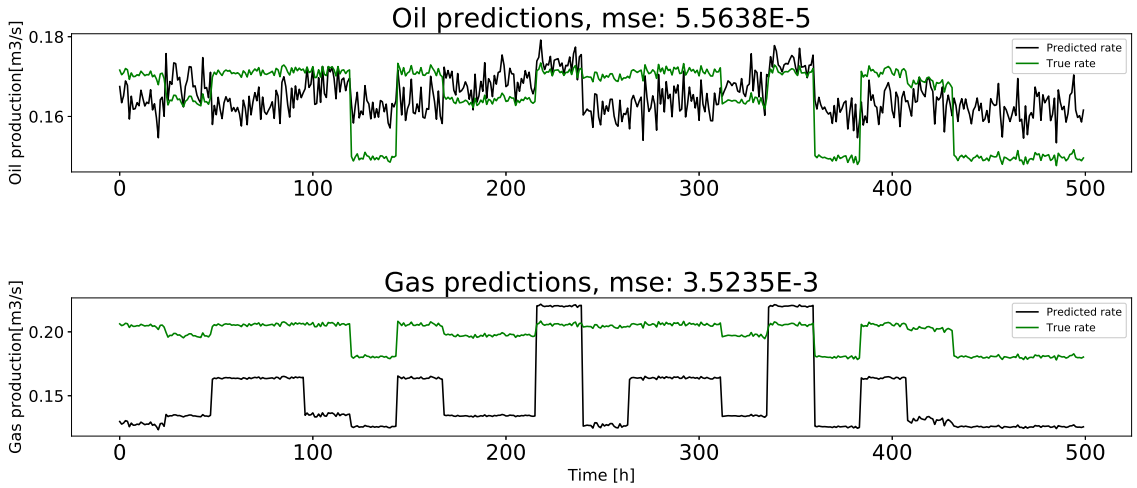


Figure 20: Validation set 1: Oil and gas predictions for Case 2.1.

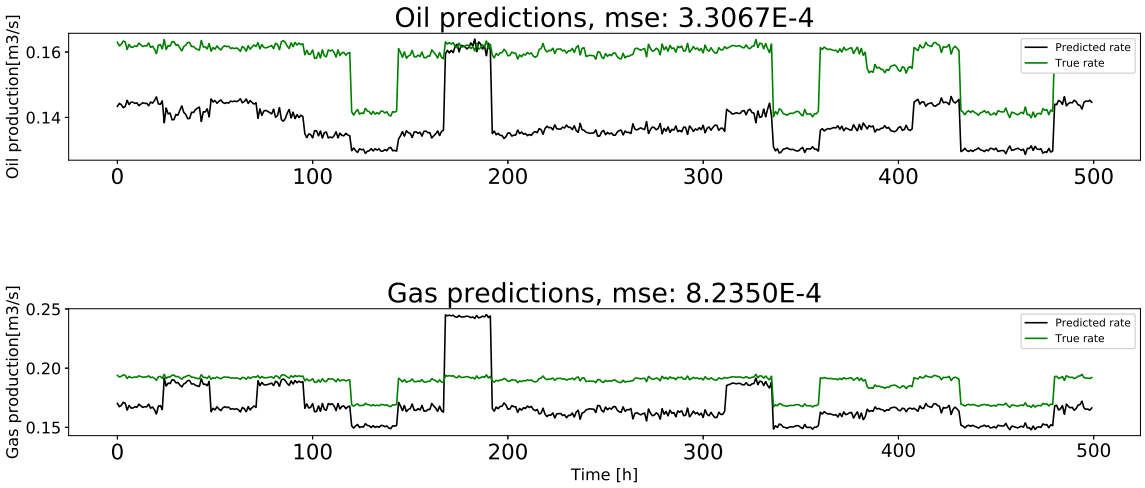


Figure 21: Validation set 2: Oil and gas predictions for Case 2.1

There is a significant contrast of the general trends between the feature engineering cases and Case 2.1, where the model was trained on the mismatch between the true rate and the Bernoulli first principle solutions. For instance, the quantitative behavior for the gas predictions is improved, which supports the hypothesis that including first principle model solutions can shift the distribution. This is evident when looking at the original and transformed target distributions in Figure 22.

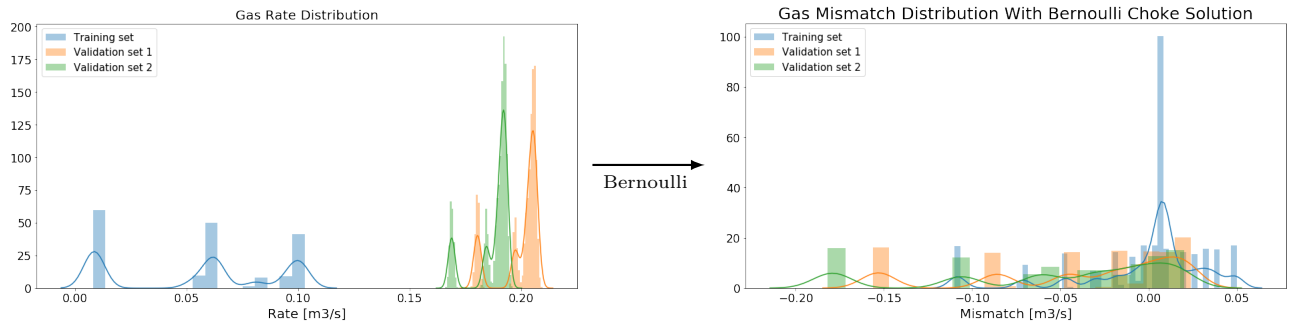


Figure 22: Original (left) distribution of gas rates and transformed (right) distribution after incorporating Bernoulli choke model into the target variable.

Since the Bernoulli choke model can overpredict the flow in the presence of gas, the calculated flow rates in the validation sets, which have a higher GOR, will consequently yield larger mismatches. As a result, the transformed distribution will therefore contain outliers in the validation sets, as seen in Figure 22. Due to neural networks limited extrapolation capabilities, the mismatch is, therefore, predisposed to be underestimated, thus the gas predictions in Figures 20 and 20 are essentially the Bernoulli choke model solution.

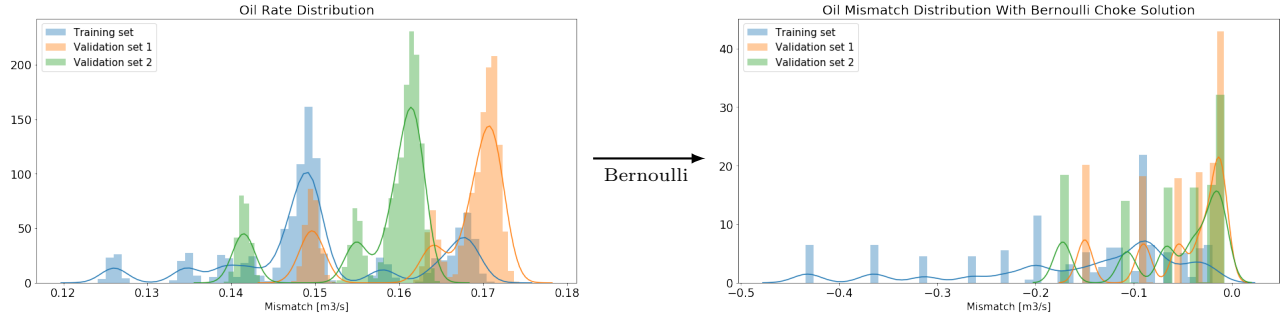


Figure 23: Original (left) distribution of gas rates and transformed (right) distribution after incorporating Bernoulli choke model into the target variable.

As for the oil predictions, Method 2 does not improve the performance compared to the baseline model. By looking at the initial and transformed distribution in Figure 23, it is clear that by including the Bernoulli choke model solution into the target variables increases the target space and relative skewness.

This will affect the regression intercept and coefficients associated with the model, which can reduce the performance [Vasudev, 2017].

In addition, including the Bernoulli choke solution in the target variable will increase the information density and therefore reduce the correlation between the input and output targets. This is evident when looking at the Pearson correlation matrix in Figure 24. When the input-output correlation decreases, it will be harder for the neural network to generalize the pattern. The only input variable with an increased correlation is the choke opening, which is logical since the Bernoulli choke equation is linearly dependent on the choke opening and could be the reason for a more aggressive choke characteristic in the predictions.

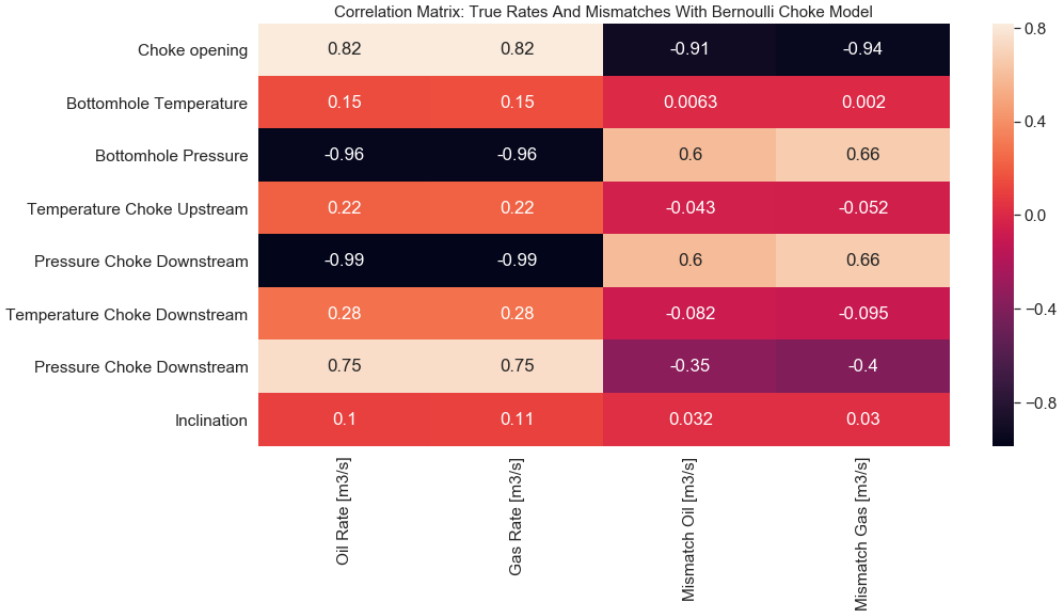


Figure 24: Pearson correlation matrix for true rates and mismatches calculated with the Bernoulli choke model.

4.8 Case 2.2

In Case 2.2, Method 2 was used with the Al-Safran choke model as the first principle solution. The raw data is being used directly as an input, and the target variables are the mismatch between the true gas rate and the solution from the Al-Safran choke model, as proposed in Method 2. A summary of the input feature, target variables, and first principle solutions is shown in Table 12.

Table 12: Inputs, first principle models and traning targets for Case 2.2

Input	First principle model	Training target
P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Al Safran choke model	$Q_{oil}^{mismatch}$, $Q_{gas}^{mismatch}$

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 25 and 26 for validation set 1 and 2, respectively.

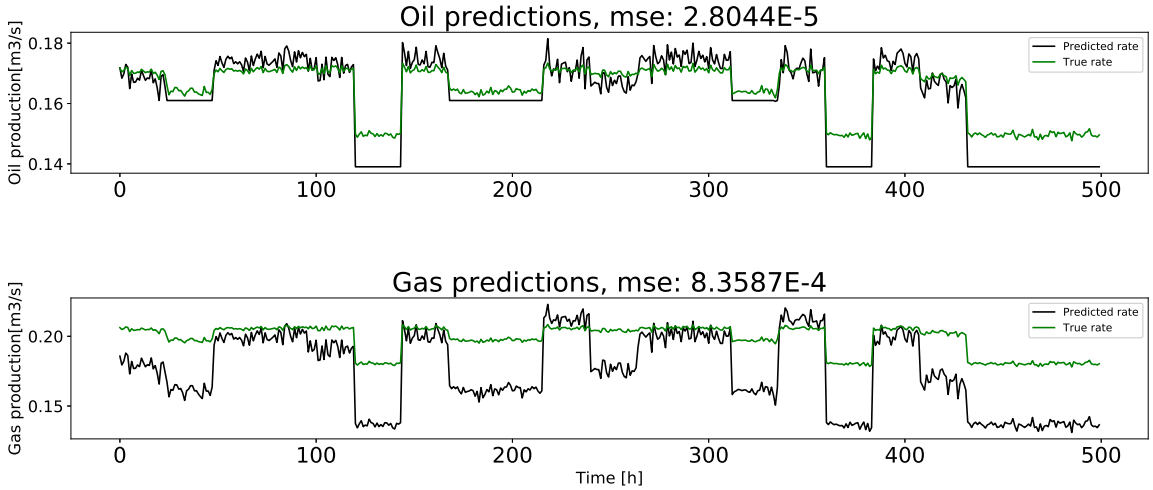


Figure 25: Validation set 1: Oil and gas predictions for Case 2.2.

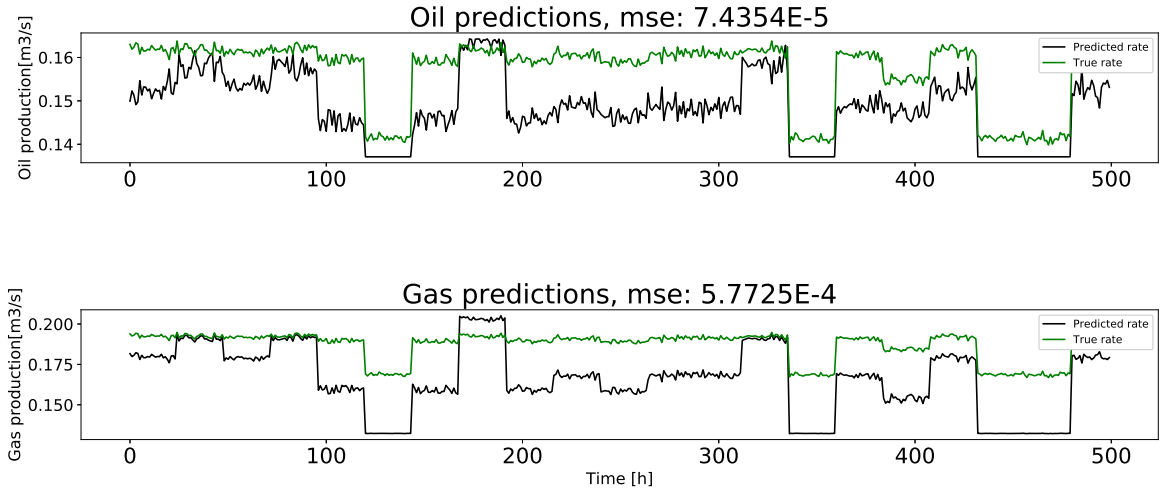


Figure 26: Validation set 2: Oil and gas predictions for Case 2.2.

Replacing the target variable to the mismatch between the true rate and the Al-Safran choke model results in improved results for both oil and gas for each validation set compared to the previous case. Since the Al-Safran Choke model yields more accurate solutions compared to the Bernoulli choke model, the mismatch will be more similar in the training and validation, which reduces the extrapolation range. This is evident in the original and transformed target distribution of gas and as shown in Figure 27. Compared to the transformed distribution with Bernoulli, as shown in Figure 22, the target space is both reduced, less skewed, and more mutually overlapping.

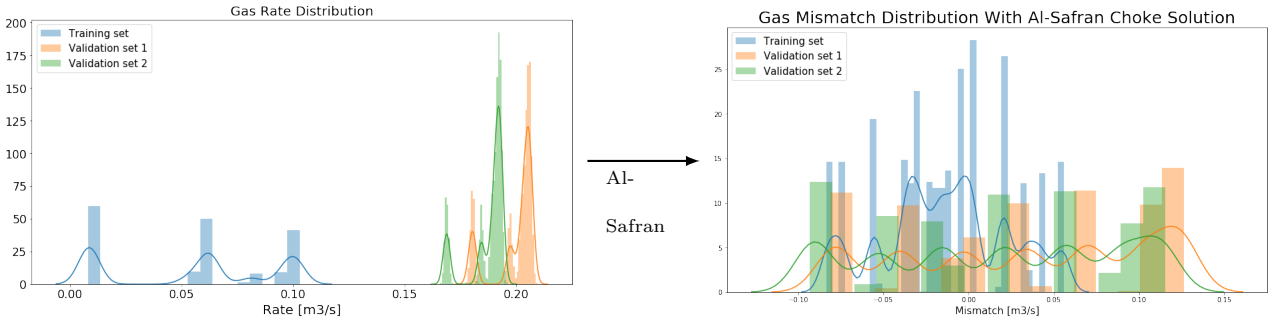


Figure 27: Original (left) distribution of gas rates and transformed (right) distribution after incorporating Al-Safran choke model into the target variable.

Compared to the previous case, oil predictions are also more accurate. This can also be seen for Figure 28, where Al-Safran yields a smaller shift in the posterior distribution. Still, the transformed distribution is less overlapping compared to the original distribution.

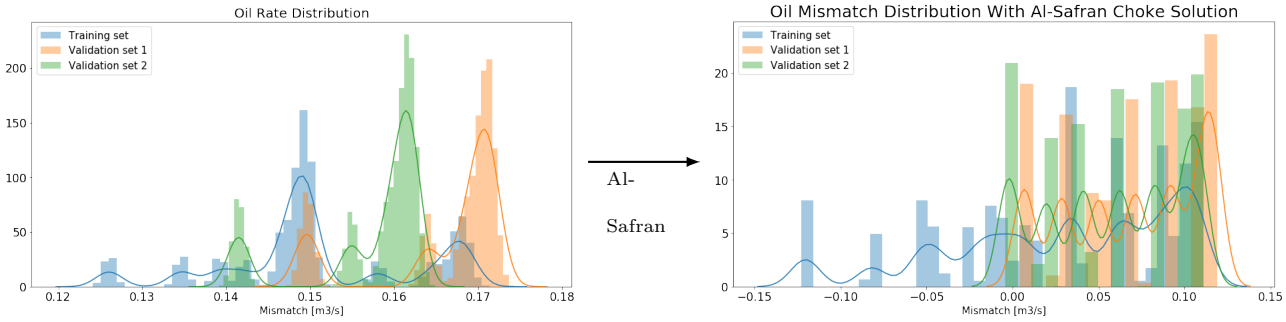


Figure 28: Original (left) distribution of oil rates and transformed (right) distribution after incorporating Al-Safran choke model into the target variable.

There are, however, some regions where the validation sets exceed the range of the training set, which in terms of extrapolation, is more difficult for the neural network to predict. Also, since the target variable now contains information about both the true rate and the Al-Safran solution, the input-output correlation could be harder to capture. This is also confirmed by looking at the Pearson correlation matrix in Figure 34.

Figure 29: Pearson correlation matrix for true rates and mismatches calculated with the Al-Safran Choke model.

As expected, the overall correlation decreases when using mismatch as the target variable compared to using the true rate. In addition, the relative change in each correlation value disproportionate, which can make it harder for the neural network to generalize the correct underlying pattern. However, the relative decrease in the correlations is smaller compared to Case 2.1 with Bernoulli, which is reflected by an increased performance.

4.9 Case 2.3

In Case 2.3, the input for the neural network is the raw data, and the target variables are the mismatch between the true gas rate and the solution for the NPW tubing equation, as proposed in Method 2. The friction factor is assumed to be constant. A summary of input features, target variables, and first principle solutions can are shown in Table 13.

Table 13: Inputs, first principle models and traning targets for Case 2.3

Input	First principle model	Training target
P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Tubing model $\xi = \text{const}$	$Q_{oil}^{mismatch}$, $Q_{gas}^{mismatch}$

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 30 and 31 for validation set 1 and 2, respectively.

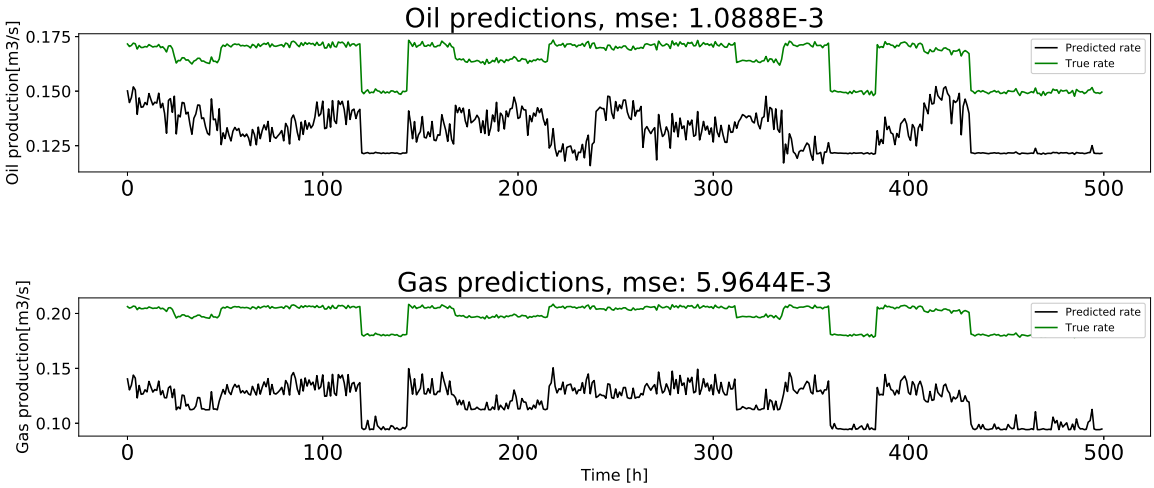


Figure 30: Validation set 1: Oil and gas predictions for Case 2.3

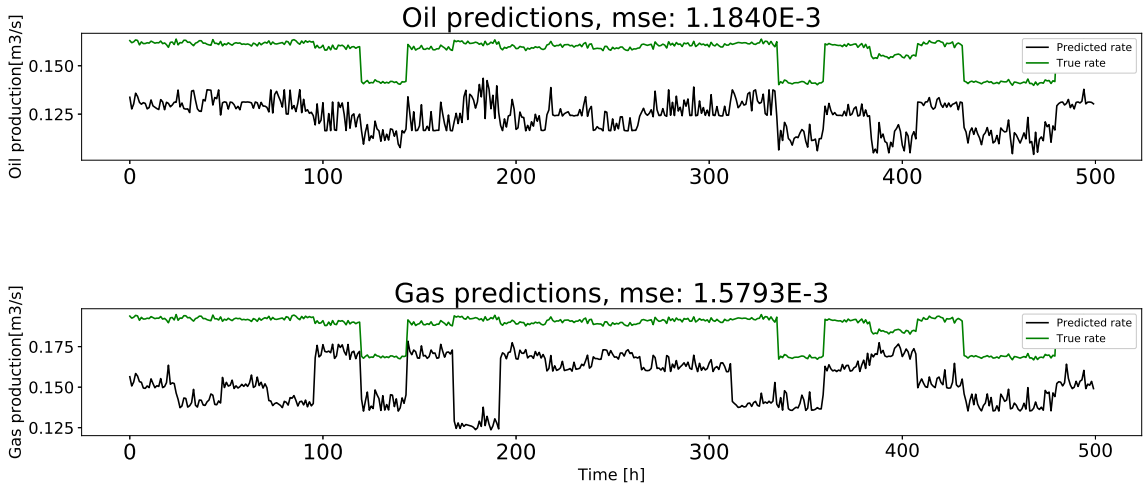


Figure 31: Validation set 2: Oil and gas predictions for Case 2.3

Compared to using choke models as first principle solution, which consistently over or under predicted the flow rate, the dynamics is now less aggressive. For both validation sets the predictions are overall under predicted. By looking at the distribution of the oil mismatches, it is clear that including the principle solutions in the training targets have shifted the distance between traning and validation targets as shown in Figure 32.

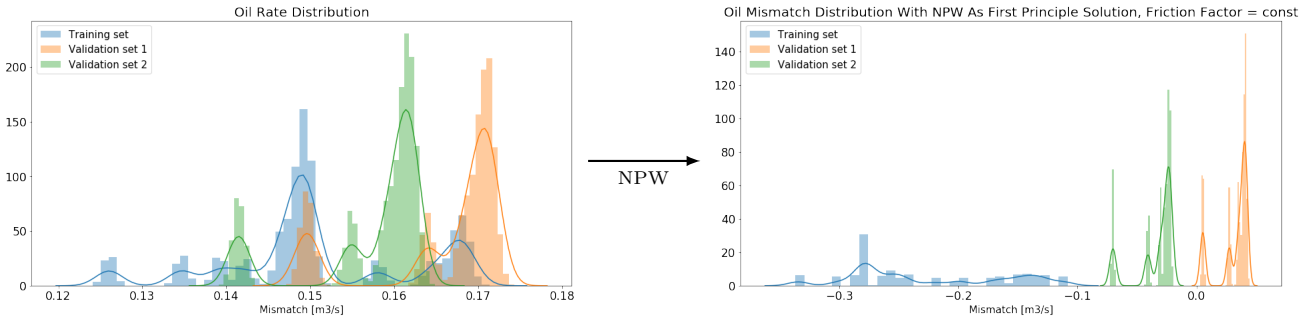


Figure 32: Original (left) distribution of oil rates and transformed (right) distribution after incorporating the NPW tubing equation with a variable friction factor into the target variable.

From Figure 32, it is clear that the predicted mismatches for oil will be underestimated, thus resulting in overall lower flow rates. This suggests that if the original training set and validation targets have similar or overlapping distribution, including first principle solutions in the training target can distort the distribution and result in a significant error.

The predictions of gas in validation set 2 are notably improved. This is also evident in the distribution for the gas mismatch, as shown in Figure 33. As discussed earlier, the well in validation set 2 is most similar to the training set in terms of the average inclination. Hence, the pressure drop will be more similar to the training set, and consequently, yield more alike first principle solutions and mismatches as seen in Figure 33.

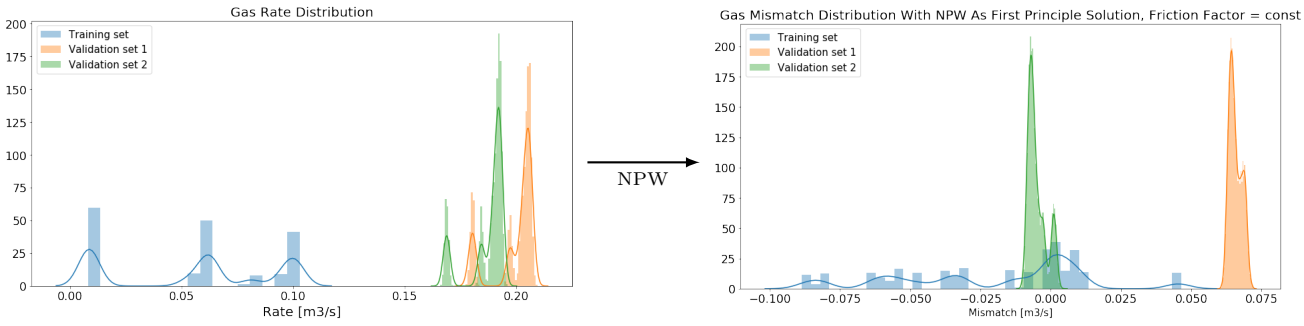


Figure 33: Original (left) distribution of gas rates and transformed (right) distribution after incorporating the NPW tubing equation with a variable friction factor into the target variable.

Even though the error increases compared to Case 2.1 and 2.2, the predictions are more quantitative stable. Since the pressure drop over the tubing is larger relative to the choke pressure drop, the predictions are less fluctuating around the true rates. The results imply that it can be beneficial to use a first principle model that describes the system more globally. However, the NPW tubing model did not succeed in shifting the target variables for validation set 1 inside the training distribution. Still, it is relatively closer compared to the initial distribution, which is reflected in more quantitative correct predictions.

When looking at the correlation matrix, it is evident that by including the NPW tubing equation, the relative correlations for oil are unaffected. Which in itself is good; however, by considering that the transformed distribution is worse than the original, the overall performance will decrease. For the gas in validation set 1, the correlation increased for most variables. This will have a positive effect on the performance, which is supported by looking at the gas predictions from Figure 30. Still, as seen from Figure 33, the transformed distribution for gas in validation set 1 is outside the range of the training target, which consequently decreases the performance.

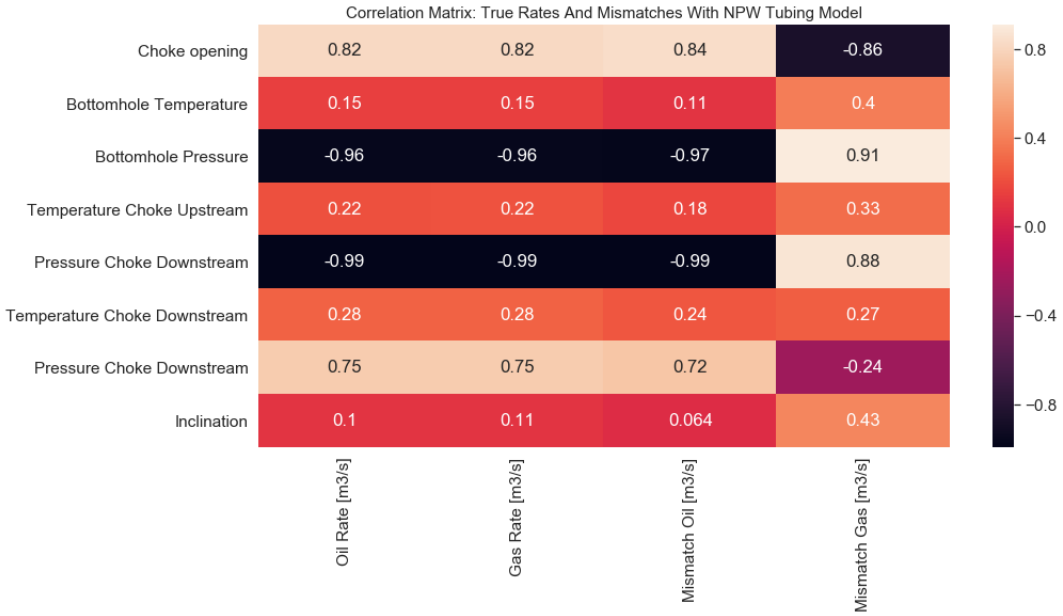


Figure 34: Pearson correlation matrix for true rates and mismatches calculated with the NPW tubing model and constant friction factor.

4.10 Case 2.4

In Case 2.4, the input for the neural network is the raw data, and the mismatch between the NPW tubing equation and true rate is the target variable. The friction factor is assumed to be a function of the Reynolds number. A summary of input features, target variables and first principle solutions can be

found in Table 14.

Table 14: Inputs, first principle models and traning targets for Case 2.4

Input	First principle model	Training target
P_{BH}, T_{BH}, P_{WHCU}, T_{WHCU}, P_{WHCD}, T_{WHCD}, C_{op}, θ	Tubing model $\xi = f(Re)$	$Q_{oil}^{mismatch}$, $Q_{gas}^{mismatch}$

The oil and gas predictions and true rates for the first 500 hours are shown in Figure 35 and 36 for validation set 1 and 2, respectively.

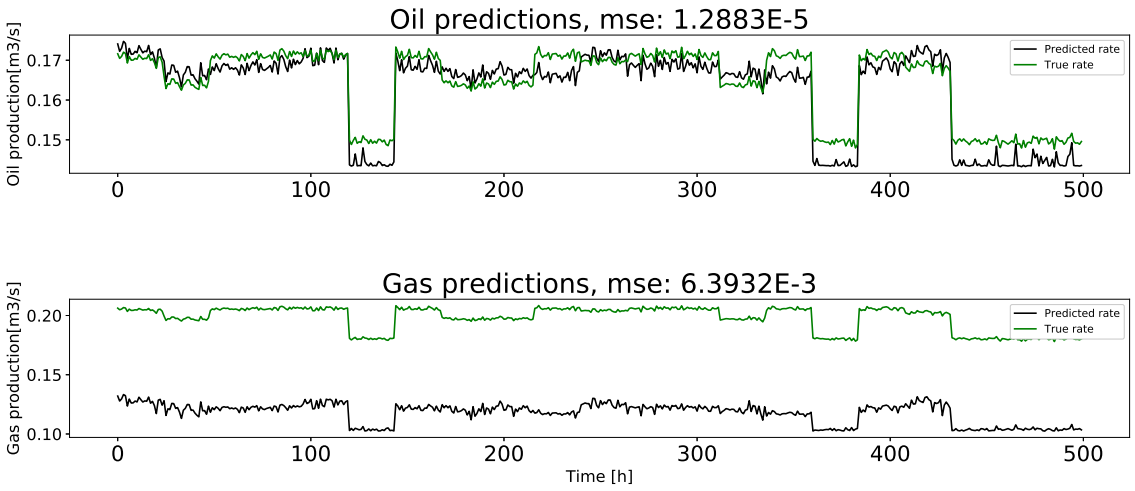


Figure 35: Validation set 1: Oil and gas predictions for Case 2.4

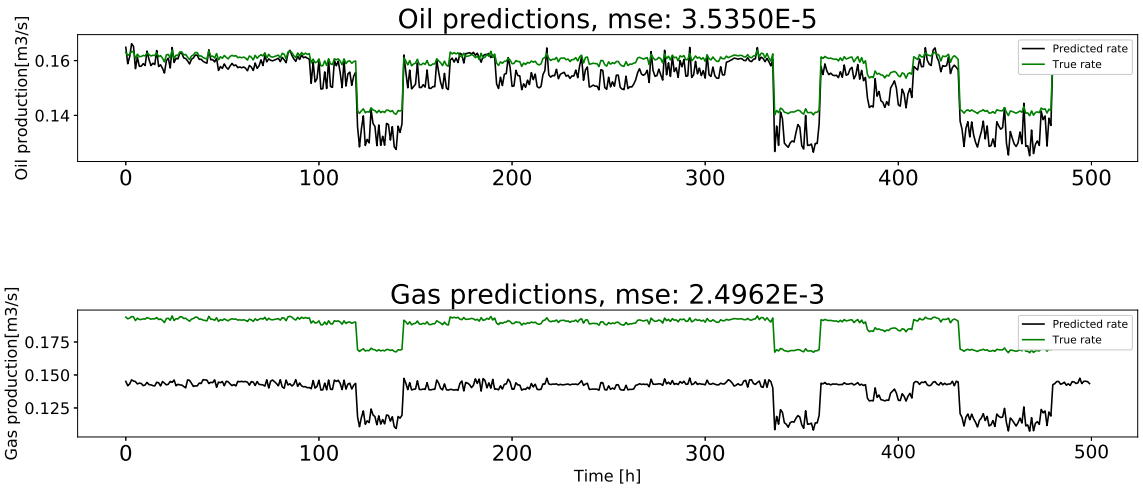


Figure 36: Validation set 2: Oil and gas predictions for Case 2.4

Including the friction factor as a variable when calculating the NPW equation resulted in significant improvements for oil compared to Case 2.3, both considering bias and matching the choke dynamics. As discussed before, a more accurate first principle model will yield an improved mismatch distribution. Compared to the previous case where a variable friction factor was assumed to be constant, the transformed distribution for the training and validation set are more mutually overlapping, as seen in figure 37. This is also expected since the friction factor will have a preponderant effect on the oil flow. Although performance for oil was not ideal compared to Method 1, it can be concluded that the inclusion of the friction factor will be beneficial to describe the oil rate more precise.

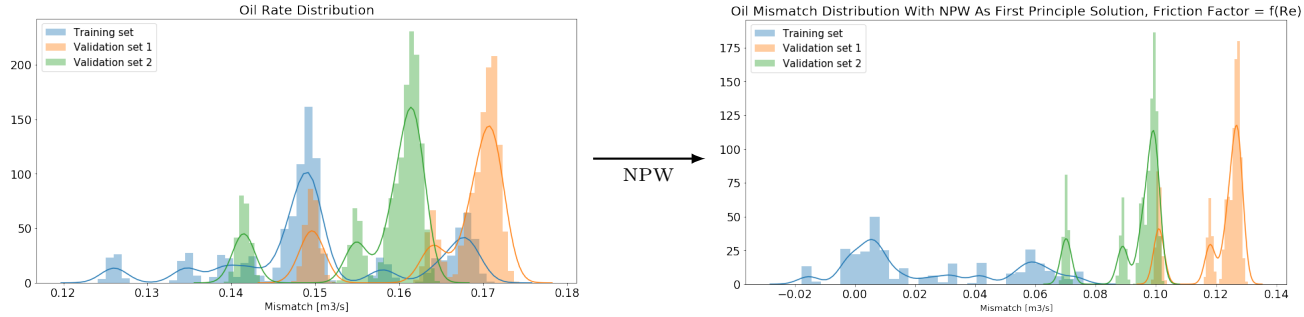


Figure 37: Original (left) distribution of oil rates and transformed (right) distribution after incorporating the NPW tubing equation with a variable friction factor into the target variable

In contrast, the gas predictions did not improve compared to the previous case. As seen in Figure 38, the transformed distributions are closer. Comparing the distribution in Case 2.3, as found in Figure 33, the validation targets are separated further from the training targets. This is also reflected in the results, due to a expanded extrapolation range the mismatch will be underestimated, thus resulting in constantly underpredicted gas rates. Since the gas flow is not as dependent on the friction, the NPW tubing equation can yield less accurate solution thus skewing the distribution.

As for the correlation matrix, the result is identical to Case 2.3. This suggests that the inclusion of friction factor has an insignificant effect on the correlations.

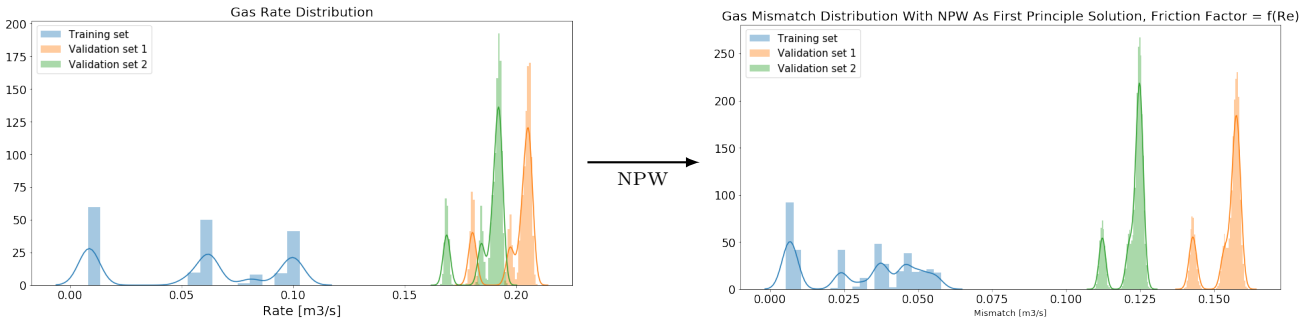


Figure 38: Original (left) distribution of gas rates and transformed (right) distribution after incorporating the NPW tubing equation with a variable friction factor into the target variable.

Figure 39: Pearson correlation matrix for true rates and mismatches calculated with the NPW tubing model.

5 Case Study Results

Table 15 presents the mean squared error for both oil and gas for each validation set. The improvement compared to the baseline model is marked as green. Case 1.1 yielded the highest accuracy for the oil predictions in validation set 2, Case 1.4 - for the oil in validation set 2, while Case 2.2 yielded the highest accuracy for the gas predictions in both validation sets.

Table 15: Summary of the mean squared error for all case studies. Green indicates that the case improved accuracy compared to the baseline model. The case which yielded the best performance is highlighted as bold text

Case	Validation set 1		Validation set 2	
	Oil	Gas	Oil	Gas
Baseline	3.4420e-5	3.0249e-3	2.5742e-5	5.3970e-3
1.1	1.0541e-4	2.705e-3	**1.3123e-6**	1.1468e-2
1.2	1.1713e-5	2.0345e-3	3.609e-6	9.1366e-3
1.3	1.906e-5	6.65e-3	2.9158e-5	7.6765e-3
1.4	**3.2422e-6**	8.2785e-3	4.6655e-6	7.5068e-3
1.5	2.1742e-5	1.2383e-3	1.3851e-4	8.8000e-4
2.1	5.5638e-5	3.5235e-3	3.3067e-4	8.2350e-4
2.2	2.8044e-5	**8.3587e-5**	7.4354e-5	**5.772e-4**
2.3	1.0888e-3	5.9644e-3	1.1849e-4	1.5793e-3
2.4	1.2883e-5	6.3932e-3	3.5350e-5	2.4962e-3

6 Case Study Discussion

The Baseline Model confirms that neural networks struggle when encountering data that is outside the trained range. Including features should improve the neural network's ability to generalize the data by giving additional information about the quantitative relationship for both the total flow as well as the individual rates of oil and gas. As a general trend, including first principle solutions as inputs in Method 1 improved the performance. The governing dynamics of the system, although biased, also improved. The oil prediction was overall more accurate for Method 1. Especially, Case 1.4 resulted in very accurate oil rate predictions, and compared to the Baseline Model, the error was reduced by a factor of ten. This proves that the inclusion of first principle models will improve the performance, compared to the Baseline Model, if the features are well correlated with the target variables. In addition, the features allowed the neural network to discover and exploit relationships that the raw data couldn't directly provide, which is confirmed by improved dynamic behaviour.

The overall prediction accuracy for gas in Method 1 was weak compared to the Baseline Model. For validation set 2, the input features did not represent the overall increase in flow rate due to properties like average inclination being too close to the training data. By taking this into account and comparing validation set 2 to Case 1.1 where only the features from the raw data were used, the gas prediction improved by the inclusion of the first principle models. On the other hand, the gas predictions for validation set 2 showed an overall increased performance compared to the Baseline Model by systematically introducing new features. Also, in Case 1.5, where both choke features and tubing features for three different sections of the well were included, benefited the gas prediction for both validation sets and yielded the highest gas accuracy for Method 1. This indicates that including information about several parts of the well could benefit the neural network's ability to capture the governing dynamics along the well and how it subsequently affects the rates.

For the gas predictions, Methods 2 generally resulted in more accurate and quantitative correct solutions compared to the Baseline Model. Including first principle solutions in the target variables has been proven to shift the target variables in the training and validation sets relatively closer, thus decreasing the extrapolation range. As a general trend, more accurate first principle solutions result in more similar mismatches hence increasing the characteristics of interpolation, which again will increase the performance. A limitation of Method 2 is that it generally decreases the input-output correlations. This reduces the neural networks' ability to generalize the pattern and the overall performance will consequently decrease. The results of Method 2 were also more characterized by the first principle solutions compared to when the true rate was used as a target variable, which was especially evident when using the choke models. In other words, the mismatch predictions were diminished compared to the first principle solutions.

As for the oil predictions, Method 2 yielded an overall lower accuracy compared to Method 1. Since the oil rate distribution for training and validation set already was mutual overlapping, the first principle solutions can distort the original distribution, thus shifting the target variables further apart.

Since extrapolating with neural networks, in general, tends to produce arbitrary predictions, the results might be misleading [Lohninger, 2012]. For the gas predictions, improvements by introducing new features can, therefore, be due to randomness and not an increased ability to identify the underlying pattern. There are, therefore, uncertainties linked to the results of the gas predictions in Method 1, were the extrapolation range was extensive. Therefore, further investigations are needed to establish a general trend for gas. On the other hand, since the oil distributions were more overlapping, it is safer to conclude that prediction accuracy increases if more and well-correlated features are systematically included.

Using Bayesian hyperparameter optimization as a tuning technique has proven one of the most superior methods to tune a neural network. However, one precondition is that the error function can be approximated as a Gaussian process. To put it another way, similar combinations of hyperparameters should result in a similar performance. However, as extrapolation predictions tend to be sometimes random, this condition might not hold, such that solution converges to an arbitrary local optimum. This was confirmed by hand-tuning the models, where the accuracy often fluctuated drastically under different, yet similar, sets of parameters. However, this also implies that better performance can be achieved by using a different tuning technique.

In this work, a low diversity of features could be a shortcoming in the data set. Even though noise was added to make the data set realistic, the input range was still very narrow, especially considering the GOR and inclination. Although GOR was not directly included as an input, it was indirectly incorporated into the model using the PVT tables for calculating the first principle solutions. The training data set consisted of only three unique GOR's and inclinations. That may be too few for the neural network to map the underlying relationship. Besides, the GOR and inclinations randomly coupled, which means they did not increase simultaneously, which can make it even harder for the neural network to generalize the underlying pattern, especially when faced with a validation set outside the trained range.

7 Conclusion and future work

This project aimed to investigate several methods that could potentially improve the extrapolation capabilities of a neural network-based VFM. Method 1 investigated the possibility of introducing physically meaningful features that could describe the underlying pattern accurately. First principle model solutions were also included in order to improve the neural networks ability to capture the quantitative relationship between oil and gas rates as well as supplementing additional information about the governing dynamics. Method 2 aimed to decrease the relative distance between the traning and validation set distributions by transforming the target variable into the mismatch between the true rate and first principle model solution.

As hypothesized for Method 1, transforming the raw data into features would allow the neural network to exploit different relationships that the raw data can not directly provide. Method 1 resulted in overall stable predictions and improved performance, especially for the oil rate predictions. There is a clear trend between systematically introducing well-correlated features and performance improvement. It is, however, difficult to point out which first principle solution yielded the overall highest accuracy, although a general recommendation is to include at least one of them. Predicting flow rates in the correct quantitative range when the extrapolation range is large, as in the gas predictions, proved to be a significant challenge. However, by taking the initial distribution into account, the first principle model solutions contributed to improving the accuracy and dynamics compared to the Baseline Model. It can, therefore, be concluded that Method 1 gives a clear advantage over current methods that only use raw data as input without any pre-processing. In Case 1.5, where multiple tubing features were included, the gas predictions improved considerably and yielded the highest accuracy in Method 1. These measurements are, however, usually not available in a real production system, and the method is, therefore, less realistic.

Method 2 revealed that the initial distribution could be transformed by incorporating first principle model solutions into the target variable. The new distribution could, consequently, be more mutually overlapping, thus reducing the extrapolation range. Especially, the findings proved to be an advantage for the gas predictions were the original target distribution of the training and validation sets was separated. A general recommendation is to use a first principle model that estimates the flows accurately to reduce the target deviation. The importance of using accurate first principle models is proved by the comparing Case 2.1 and 2.2, wherein Case 2.2 the Al-Safran choke model resulted in higher performance compared to Case 2.1 with the Bernoulli choke model. It was also found that if the initial target distributions are already overlapping, such as for the oil, the inclusion of first principle models in the target variable can, in the worst scenario, increase the extrapolation range.

Considering the findings in a practical sense, it was previously introduced two scenarios where extrap-

olation with a neural network-based VFM would be beneficial. The first scenario was that the VFM would work as a backup or a supplementary solution. Here, it is assumed that the VFM would be trained based on the estimates from the MPFM. As the fields start to deplete, the gas production will increase, and the operating conditions will be different from the distribution it was trained upon. If we consider a smaller time frame, the operation conditions will not be so different from the training set. Method 1 would then be the preferred solution. The second scenario was that if a VFM would operate as a standalone solution, the training data will originate from different wells, such that the input training and target distributions will differ from the actual operating conditions. The recommended method will then depend on the available data. If the available data are from wells that are significantly different from the new well, Method 2 is beneficial. By including first principle model solutions in the target variables in the available training data and the current measurements from operating well, the performance will improve by shifting the distributions.

All things considered, it is possible to improve extrapolating capabilities of a neural network-based VFM by using the methods proposed which can contribute to improving its performance. Whether Method 1 or 2 should be adopted depends on the available training data. There is still room for major improvements. The results are promising but should be trained and validated by a more diverse data set in terms of inclinations, trajectories, GOR, and boundary properties. Since this project only considers validation sets with larger oil and gas flow rates compared to the training set, the performance under different validation set distributions should also be investigated. Combining the stability of Method 1 and the qualitatively accurate predictions from Method 2 into an ensemble method can also be beneficial. Moreover, the first principle models can be advanced to calculate the mismatch more precisely. In conclusion, incorporating first principle solution results in more explainable and transparent neural networks that could contribute to making Virtual Flow Metering more accurate, efficient, and reliable.

List of symbols

Table 16: List of Symbols.

Symbol	Description
P_{BH}	Bottomhole pressure
T_{BH}	Bottomhole temperature
P_{WHCU}	Pressure choke upstream
T_{WHCU}	Temperature choke upstream
P_{WHCD}	Pressure choke downstream
T_{WHCD}	Temperature choke downstream
C_{op}	Choke opening
J	Cost function
$\hat{y}$	Predicted value in neural network
w_i	Weights in neural network
λ	Regularization term
M_{GP}	Surrogate model
P_1	Pressure along streamline
P_2	Pressure along streamline
U_1	Velocity along streamline
U_1	Velocity along streamline
A_1	Pipe cross-sectional area
A_2	Choke throat area
C_d	Discharge coeffecient
ρ_{mix}	Density of mixture
C	Correction factor for Al-Safran choke model
α	Hypothetical parameter
n	Polytropic gas-expansion exponent
x_g	Gas quality
ν_{g1}	Upstream gas specific volume
C_{vg}	Gas specific heat constant at constant volume
C_L	Liquid specific-heat constant
R	Split ratio
l	Pipe axial coordinate
θ	Pipe inclination
U_{mix}	Mixture velocity

D_{tubing}	Tubing diameter
g	Gravity constant
H	Tubing height
ϵ	Relative roughness
Re	Reynolds number
P_R	Reservoir pressure
P_B	Boundary pressure at flowline
T_R	Reservoir temperature
T_B	Boundary temperature at flowline
μ_{oil}	Viscosity oil
μ_{gas}	Viscosity gas
Q	Volumetric flow rate

List of abbreviations

Table 17: List of abbreviations

Acronym	Description
VFM	Virtual Flow Metering
NPW	No Pressure Wave
MPFM	Multiphase Flow Meter
ReLU	Rectified Linear Unit
lr	Learning rate
SGD	Stochastic Gradient Decent
RMSProp	Root Mean Square Propagation
ADAM	Adaptive Moment Estimation
GOR	Gas-Oil Ratio
BH	Bottomhole
LM	Lower middle
UM	Upper middle
CD	Choke downstream
CU	Choke upstream

References

[AL-Qutami et al., 2018] AL-Qutami, T. A., Ibrahim, R., Ismail, I., and Ishak, M. A. (2018). Virtual multiphase flow metering using diverse neural network ensemble and adaptive simulated annealing. *Expert Systems with Applications*, 93:72–85.

[Al-Safran and Kelkar, 2009] Al-Safran, E. and Kelkar, M. (2009). Predictions of Two-Phase Critical Flow Boundary and Mass Flow Rate Across Chokes. *SPE Production & Operations*, 24.

[Ambrus et al., 2016] Ambrus, A., Aarsnes, U. J. F., Karimi Vajargah, A., Akbari, B., van Oort, E., and Aamo, O. M. (2016). Real-time estimation of reservoir influx rate and pore pressure using a simplified transient two-phase flow model. *Journal of Natural Gas Science and Engineering*, 32:439–452.

[Andrianov, 2018] Andrianov, N. (2018). A Machine Learning Approach for Virtual Flow Metering and Forecasting. *3rd IFAC Workshop on Automatic Control in Offshore Oil and Gas Production OOGP 2018*, 51(8):191–196.

[Barbariol et al., 2019] Barbariol, T., Feltresi, E., and Susto, G. A. (2019). Machine Learning approaches for Anomaly Detection in Multiphase Flow Meters. *5th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2019*, 52(11):212–217.

[Barnard and Wessels, 1992] Barnard, E. and Wessels, L. (1992). Extrapolation and Interpolation in Neural Network Classifiers.

[Basha et al., 2018] Basha, S. H. S., Ghosh, S., Babu, K., Dubey, S. R., Pulabaigari, V., and Mukherjee, S. (2018). *RCCNet: An Efficient Convolutional Neural Network for Histological Routine Colon Cancer Nuclei Classification*.

[Bikmukhametov and Jäschke, 2019] Bikmukhametov, T. and Jäschke, J. (2019). Oil Production Monitoring using Gradient Boosting Machine Learning Algorithm The authors gratefully acknowledge the financial support from the center for research-based innovation SUBPRO, which is financed by the Research Council of Norway, major industry partners, and NTNU. *12th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems DYCOPS 2019*, 52(1):514–519.

[Bikmukhametov and Jäschke, 2020a] Bikmukhametov, T. and Jäschke, J. (2020a). First Principles and Machine Learning Virtual Flow Metering: A Literature Review. *Journal of Petroleum Science and Engineering*, 184.

[Bikmukhametov and Jäschke, 2020b] Bikmukhametov, T. and Jäschke, J. (2020b). On How to Combine Machine Learning and Physics to Get Accurate and Explainable Models.

[Colebrook, 1939] Colebrook, C. F. (1939). Turbulent Flow in Pipes, with particular federence to the Transition Region between Smooth and Rough Pipe Laws.

[Ghalambor, 2007] Ghalambor, A. (2007). Petroleum Production System. In *Petroleum Production Engineering: A Computer-Assisted Approach*.

[Ghiaasiaan, 2019] Ghiaasiaan, S. (2019). Two-Phase Flow Modeling. pages 162–198.

[Haley and Soloway, 1992] Haley, P. and Soloway, D. (1992). Extrapolation Limitations of Multilayer Feedforward Neural Networks. *NASA Langley Research Center*.

[Haug, 2012] Haug, R. K. (2012). Multiphase Flow Through Chokes - Evaluation of five models for prediction of mass flow rates.

[Holmås and Løvli, 2011] Holmås, K. and Løvli, A. (2011). FlowManager^{TM} dynamic: A multiphase flow simulator for online surveillance, optimization and prediction of subsea oil and gas production. pages 241–254.

[Hvass Pedersen, 2018] Hvass Pedersen, M. E. (2018). Hyper-Parameter Optimization.

[Jhedengren, 2018] Jhedengren (2018). Results of training an artificial neural network.

[Jiang et al., 2018] Jiang, X., Pang, Y., Li, X., Pan, J., and Xie, Y. (2018). Deep neural networks with Elastic Rectified Linear Units for object recognition. *Neurocomputing*, 275:1132–1139.

[Kongsberg, 2001] Kongsberg (2001). LedaFlowTM.

[Lars Hulstaert, 2017] Lars Hulstaert (2017). Understanding objective functions in neural networks. *towardsdatascience*.

[Lawrence and Giles, 2000] Lawrence, S. and Giles, C. (2000). *Overfitting and neural networks: Conjugate gradient and backpropagation*, volume 1.

[Lohninger, 2012] Lohninger (2012). *Fundamentals of Statistics*.

[Lévesque et al., 2016] Lévesque, J.-C., Gagné, C., and Sabourin, R. (2016). *Bayesian Hyperparameter Optimization for Ensemble Learning*.

[Mak et al., 2019] Mak, K. K., Lee, K., and Park, C. (2019). Applications of machine learning in addiction studies: A systematic review. *Psychiatry Research*, 275:53–60.

[Marshall and Thomas, 2015] Marshall, C. and Thomas, A. (2015). Maximising Economic Recovery - A Review of Well Test Procedures in the North Sea. In *SPE-175518-MS*, page 19, SPE. Society of Petroleum Engineers.

[Minns and Hall, 1996] Minns, A. W. and Hall, M. J. (1996). Artificial neural networks as rainfall-runoff models. *Hydrological Sciences Journal*, 41(3):399–417.

[Quionero-Candela et al., 2009] Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. (2009). Dataset Shift in Machine Learning.

[Robertson, 2014] Robertson, P. M. (2014). Dynamic Estimation for Controlling a Subsea Production System - Virtual Flow Metering using B-spline Surrogate Models.

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

[Snoek et al., 2012] Snoek, J., Larochelle, H., and Adams, R. (2012). PRACTICAL BAYESIAN OPTIMIZATION OF MACHINE LEARNING ALGORITHMS.

[Solomatine and Ostfeld, 2008] Solomatine, D. and Ostfeld, A. (2008). Data-Driven Modelling: Some Past Experiences and New Approaches. *Journal of Hydroinformatics - J HYDROINFORM*, 10.

[T. A. Al-Qutami et al., 2017] T. A. Al-Qutami, R. Ibrahim, and I. Ismail (2017). Hybrid neural network and regression tree ensemble pruned by simulated annealing for virtual flow metering application. In *2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 304–309.

[Varoonchotikul, 2003] Varoonchotikul, P. (2003). *Flood Forecasting using Artificial Neural Networks*. A.A . Balkema Pubhshers.

[Vasudev, 2017] Vasudev, R. (2017). How to deal with Skewed Dataset in Machine Learning?

[Yakowitz and Lugosi, 1990] Yakowitz, S. and Lugosi, E. (1990). Random Search in the Presence of Noise, with Application to Machine Learning. *Siam Journal on Scientific and Statistical Computing*, 11.

[Yoon et al., 2006] Yoon, H., Ishii, M., and Revankar, S. (2006). Choking flow modeling with mechanical and thermal non-equilibrium. *International Journal of Heat and Mass Transfer*, 49(1):171–186.

[Zhang et al., 2019] Zhang, X., Chen, X., Yao, L., Ge, C., and Dong, M. (2019). *Deep Neural Network Hyperparameter Optimization with Orthogonal Array Tuning*.

A Compositions

Table 18: Summary for compositions used for creating the PVT tables with a specific GOR.

	GOR: 20	GOR: 40	GOR:60	GOR: 80
Nitrogen	0,535	0,41	0,37	0,673
Carbon Dioxide	0,09	0,452	0,352	0,1
Methane	14,67	25,55	26,54	38,64
Ethane	1,82	1,345	5,283	3,563
Propane	1,142	2,13	4,22	3,85
I-Butane	0,66	0,512	1,49	0,554
N-Butane	1,1	2,53	0,93	1,834
I-Pentane	0,78	0,5	0,83	0,565
N-Pentane	0,6	0,356	1,5	0,684
C6	1,53	5,42	1,83	0,992
C7	3,6	1,23	5,11	1,69
C8	5,1	3,67	6,95	2,17
C9	2,2	1,87	3,81	1,74
C10+	66,173	54,025	40,785	42,945

B Code

Since parts of the code are reused for all the cases, the full code for the Baseline model, as well as the functions for calculating the first principle models will be presented. The code on Bayesian hyperparameter optimization builds existing code from GitHub. [Hvass Pedersen, 2018].

B.1 Basic Structure

```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import collections
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import cross_val_score, train_test_split
import pyfas as fa
from scipy.interpolate import interp2d
import math
from scipy.interpolate import griddata
from skopt.plots import plot_histogram, plot_objective_2D
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import math
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import InputLayer, Input
from tensorflow.python.keras.layers import Reshape, MaxPooling2D
from tensorflow.python.keras.layers import Conv2D, Dense, Flatten
from tensorflow.python.keras.callbacks import TensorBoard
from tensorflow.python.keras.optimizers import Adam
from tensorflow.python.keras.models import load_model
from skopt import gp_minimize, forest_minimize
from skopt.space import Real, Categorical, Integer
from skopt.plots import plot_convergence
from skopt.plots import plot_objective, plot_evaluations
from skopt.utils import use_named_args
import random
import os
from decimal import Decimal



"""
Read data
"""
```

```
df_train = pd.read_csv('/Users/eier/Documents/Project2019/ProjectTrain2.csv')
df_val_well1 = pd.read_csv('/Users/eier/Documents/Project2019/ProjectValidation1.2.csv
    ↪ ')
df_val_well2 = pd.read_csv('/Users/eier/Documents/Project2019/ProjectValidation2.2.csv
    ↪ ')



"""
Input and output features
"""
Features_x = ['C_op',
              'TemperatureWellbore[K]_logpos25',
              'PressueWellbore[Pa]_logpos25',
              'TemperatureWellbore[K]_logpos990',
              'PressueWellbore[Pa]_logpos990',
              'TemperatureFlowline[K]_logpos10',
              'PressueFlowline[Pa]_logpos10',
              'WellboreAngle[Degrees]']

Features_y = ['VFRoil[m3/s]',
              'VFRgas[m3/s]']




"""
Set constant seed to obtain reproducible results
"""

seed_value = 0

os.environ['PYTHONHASHSEED']=str(seed_value)

random.seed(seed_value)

np.random.seed(seed_value)
```

```
tf.random.set_seed(seed_value)


"""
Set range for search space
"""

dim_learning_rate = Real(low=1e-15, high=1e-2, prior='log-uniform',
                         name='learning_rate')

dim_num_dense_layers = Integer(low=2, high=4, name='num_dense_layers')

dim_num_dense_nodes = Integer(low=3, high=10, name='num_dense_nodes')

dim_regularization = Real(low = 1e-3, high =1e-1, prior = 'log-uniform', name = '
    ↪ regularization')

default_parameters = [1e-10, 2, 5, 0.01] #starting values


dimensions = [dim_learning_rate,
              dim_num_dense_layers,
              dim_num_dense_nodes,
              dim_regularization
              ]


"""
Initialization of mse
"""
best_mse = math.inf

def log_dir_name(learning_rate, num_dense_layers,
                 num_dense_nodes, regularization):
```

```
    # The dir-name for the TensorBoard log-dir.
    s = "./19_logs/lr_{0:.0e}_layers_{1}_nodes_{2}_regularization_{3}/"

    # Insert all the hyper-parameters in the dir-name.
    log_dir = s.format(learning_rate,
                       num_dense_layers,
                       num_dense_nodes,
                       regularization
                      )

    return log_dir


def create_model(learning_rate, num_dense_layers,
                 num_dense_nodes, regularization):
    """
    Hyper-parameters:
    learning_rate: Learning-rate for the optimizer.
    num_dense_layers: Number of dense layers.
    num_dense_nodes: Number of nodes in each dense layer.
    Regularization: Regularization
    """
    # Reset the network graphs for results reproducability in Keras
    N = 17 #Seed number
    np.random.seed(N)
    random.seed(N)
    #session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
    # inter_op_parallelism_threads=1)
    tf.random.set_seed(N)
    K.clear_session()
    #sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
    #K.set_session(sess)
```

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape = [input_shape]))
tf.get_logger().setLevel('WARNING')

# Add fully-connected / dense layers.
# The number of layers is a hyper-parameter we want to optimize.
for i in range(num_dense_layers):
    # Name of the layer. This is not really necessary
    # because Keras should give them unique names.
    name = 'layer_dense_{0}'.format(i+1)

    # Add the dense / fully-connected layer to the model.
    # This has two hyper-parameters we want to optimize:
    # The number of nodes and the activation function.
    model.add(Dense(num_dense_nodes,
                    kernel_regularizer=tf.keras.regularizers.l2(regularization), #
                        ↪ Should I optimize the regularization rate?
                    activation=tf.nn.relu,
                    kernel_initializer=keras.initializers.glorot_uniform(seed=16),
                    name=name))

# Last fully-connected

model.add(tf.keras.layers.Dense(1))

# Use the Adam method for training the network.
# We want to find the best learning-rate for the Adam method.
optimizer = tf.keras.optimizers.RMSprop(lr = learning_rate, momentum=0.9)
# In Keras we need to compile the model so it can be trained.
model.compile(optimizer=optimizer,
              loss='mse',
              metrics=['mse'])

return model
```

```
def add_noise(np_array):
    """
    Add noise from standard distribution
    """
    np.random.seed(seed_value)
    error = 0.01
    std = error/2
    return np_array+np_array*np.random.normal(0,1,len(np_array))*std



def add_noise_dataset(x,y):
    """
    Add noise to each column
    """
    for feature in x.columns:
        x[feature] = add_noise(x[feature])
    for feature in y.columns:
        y[feature] = add_noise(y[feature])
    return x,y



def scale_inverse(pred, scaler):
    """
    Inverse transform
    """
    inverse = scaler.inverse_transform(pred)
    return inverse

def hyperparameter_optimization(x_train, y_train, validation_data, path):

    @use_named_args(dimensions=dimensions)
    def fitness(learning_rate, num_dense_layers,
                num_dense_nodes, regularization):
```

```
"""
Hyper-parameters:
learning_rate: Learning-rate for the optimizer.
num_dense_layers: Number of dense layers.
num_dense_nodes: Number of nodes in each dense layer.
activation: Activation function for all layers.

"""

#Print the hyper-parameters.
print('learning␣rate:␣{0:.1e}'.format(learning_rate))
print('num_dense_layers:', num_dense_layers)
print('num_dense_nodes:', num_dense_nodes)
print('regularization:', regularization)
print()


model = create_model(learning_rate=learning_rate,
                     num_dense_layers=num_dense_layers,
                     num_dense_nodes=num_dense_nodes,
                     regularization=regularization)

#weights = model.get_weights()

log_dir = log_dir_name(learning_rate, num_dense_layers,
                       num_dense_nodes, regularization)


callback_log = TensorBoard(
    log_dir=log_dir,
    histogram_freq=0,
    write_graph=True,
    write_grads=False,
    write_images=False)
```

```
    history = model.fit(x=x_train,
                        y=y_train,
                        epochs=1,
                        validation_data=validation_data,
                        callbacks=[callback_log],
                        verbose = 0
                        )

    global best_mse

    mse = history.history['mse'][-1]

    print('MSE: ' + str(mse))


    if mse < best_mse: #update is mse decreases

        model.save(path)
        best_mse = mse
        print('best_mse update: ' + str(best_mse))
        print()
        print()
        print()
        print()

    del model

    K.clear_session()


    return mse


search_result = gp_minimize(func=fitness,
                            dimensions=dimensions,
```

```
                            acq_func='EI', # Expected Improvement.
                            n_calls=30,
                            x0=default_parameters,
                            random_state = 17

                           )
    global best_mse #Reset initialization
    best_mse = math.inf

    return search_result


"""
Initialize scalers
"""
scaler_oil = StandardScaler(copy=True, with_mean=True, with_std=True)
scaler_gas = StandardScaler(copy=True, with_mean=True, with_std=True)
scaler_x = StandardScaler(copy=True, with_mean=True, with_std=True)

class Train:

    """
    Class for initiailization of traning set
    Fit scaler
    """

    def __init__(self,df):

        df = df.copy()

        x = df.filter(Features_x, axis=1)
        y = df.filter(Features_y, axis=1)
        x, y = add_noise_dataset(x,y)

        self.x = x.values
```

```
        self.y_oil = y.values[:,0]
        self.y_gas = y.values[:,1]

        global scaler_oil
        global scaler_gas
        global scaler_x

        scaler_oil.fit(self.y_oil.reshape(-1, 1))
        scaler_gas.fit(self.y_gas.reshape(-1, 1))
        scaler_x.fit(self.x)

        self.y_oil_scaled = scaler_oil.transform(self.y_oil.reshape(-1,1))
        self.y_gas_scaled = scaler_gas.transform(self.y_gas.reshape(-1,1))
        self.x_scaled = scaler_x.transform(self.x)


class Validation:

    """
    Class for initiailization of validation set

    """
    def __init__(self, df, path_model_oil, path_model_gas):

        df = df.copy()

        self.path_model_oil = path_model_oil
        self.path_model_gas = path_model_gas

        x = df.filter(Features_x, axis=1)
        y = df.filter(Features_y, axis=1)
        x, y = add_noise_dataset(x,y)

        self.x = x.values
        self.y_oil = y.values[:,0]
```

```
        self.y_gas = y.values[:,1]

        self.y_oil_scaled = scaler_oil.transform(self.y_oil.reshape(-1,1))
        self.y_gas_scaled = scaler_gas.transform(self.y_gas.reshape(-1,1))
        self.x_scaled = scaler_x.transform(self.x)


        self.validation_data_gas = (self.x_scaled, self.y_gas_scaled)
        self.validation_data_oil = (self.x_scaled, self.y_oil_scaled)

        self.search_result_oil = None
        self.search_result_gas = None


train = Train(df_train)

well1 = Validation(df_val_well1, 'Baseline_well1_oil.h5', 'Baseline_well1_gas.h5')

well2 = Validation(df_val_well2, 'Baseline_well2_oil.h5', 'Baseline_well2_gas.h5')


input_shape = len(train.x[0])


"""
Tune four neural networks with bayesian hyperparameter optimization
"""


well1.search_result_oil = hyperparameter_optimization(x_train = train.x_scaled,
                                                      y_train = train.y_oil_scaled,
                                                      validation_data = well1.
                                                          ↪ validation_data_oil,
                                                      path = well1.path_model_oil)
```

```
well1.search_result_gas = hyperparameter_optimization(x_train = train.x_scaled,
                                                      y_train = train.y_gas_scaled,
                                                      validation_data = well1.
                                                          ↪ validation_data_gas,
                                                      path = well1.path_model_gas)


well2.search_result_oil = hyperparameter_optimization(x_train = train.x_scaled,
                                                      y_train = train.y_oil_scaled,
                                                      validation_data = well2.
                                                          ↪ validation_data_oil,
                                                      path = well2.path_model_oil)


well2.search_result_gas = hyperparameter_optimization(x_train = train.x_scaled,
                                                      y_train = train.y_gas_scaled,
                                                      validation_data = well2.
                                                          ↪ validation_data_gas,
                                                      path = well2.path_model_gas)


class CustomCallback(keras.callbacks.Callback):

    """
    Custom Callback for early stopping
    Saves model if accuracy improves
    """

    def __init__(self, validation_data, path):

        self.x_val, self.y_val = validation_data
        self.path = path
```

```
    def on_train_begin(self, logs=None):
        predictions = self.model.predict(self.x_val)
        self.best_mse = ((np.array(self.y_val)-np.array(predictions))**2).mean()

    def on_epoch_end(self, batch,epoch, logs={}):
        predictions = self.model.predict(self.x_val)
        mse = ((np.array(self.y_val)-np.array(predictions))**2).mean()
        if mse < self.best_mse:
            self.best_mse = mse
            self.model.save(self.path)
            print('epoch: ' + str(epoch) + 'mse improved: ' + str(self.best_mse))


"""
Initialize Custom Callbacks
"""
callback_well1_oil = CustomCallback(well1.validation_data_oil, well1.path_model_oil)
callback_well1_gas = CustomCallback(well1.validation_data_gas, well1.path_model_gas)
callback_well2_oil = CustomCallback(well2.validation_data_oil, well2.path_model_oil)
callback_well2_gas = CustomCallback(well2.validation_data_gas, well2.path_model_gas)

"""
Load saved models from Bayesian Hyperparameter tuning
"""
model_oil_well1 = load_model(well1.path_model_oil)
model_gas_well1 = load_model(well1.path_model_gas)
model_oil_well2 = load_model(well2.path_model_oil)
model_gas_well2 = load_model(well2.path_model_gas)

"""
Fit the models with early stopping
"""

model_oil_well1.fit(train.x_scaled,
                    train.y_oil_scaled,
```

```
                  epochs=20,
                  verbose=0,
                  callbacks=[callback_well1_oil])

model_gas_well1.fit(train.x_scaled,
                  train.y_gas_scaled,
                  epochs=20,
                  verbose=0,
                  callbacks=[callback_well1_gas])



model_oil_well2.fit(train.x_scaled,
                  train.y_oil_scaled,
                  epochs=20,
                  verbose=1,
                  callbacks=[callback_well2_oil])

model_gas_well2.fit(train.x_scaled,
                  train.y_gas_scaled,
                  epochs=20,
                  verbose=0,
                  callbacks=[callback_well2_gas])



model_oil_well1 = load_model(well1.path_model_oil)
model_gas_well1 = load_model(well1.path_model_gas)
model_oil_well2 = load_model(well2.path_model_oil)
model_gas_well2 = load_model(well2.path_model_gas)


"""
Predictions
"""
```

```
predictions_oil_well1 = model_oil_well1.predict(well1.x_scaled)
predictions_gas_well1 = model_gas_well1.predict(well1.x_scaled)
predictions_oil_well2 = model_oil_well2.predict(well2.x_scaled)
predictions_gas_well2 = model_gas_well2.predict(well2.x_scaled)

pred_oil_well1 = scale_inverse(predictions_oil_well1, scaler_oil)
pred_gas_well1 = scale_inverse(predictions_gas_well1, scaler_gas)
pred_oil_well2 = scale_inverse(predictions_oil_well2, scaler_oil)
pred_gas_well2 = scale_inverse(predictions_gas_well2, scaler_gas)

"""
Calculate mean squared error
"""
mse_oil_well1 = ((np.array(well1.y_oil)-np.array(pred_oil_well1.ravel()))**2).mean()
mse_gas_well1 = ((np.array(well1.y_gas)-np.array(pred_gas_well1.ravel()))**2).mean()
mse_oil_well2 = ((np.array(well2.y_oil)-np.array(pred_oil_well2.ravel()))**2).mean()
mse_gas_well2 = ((np.array(well2.y_gas)-np.array(pred_gas_well2.ravel()))**2).mean()


### Validation set: well 1

"""
Plot Results for validation set 1
"""

plt.subplot(2, 1, 1)
plt.plot(pred_oil_well1[:500], label = 'Predicted␣rate', color = 'black')
plt.plot(well1.y_oil[:500],label = 'True␣rate', color = 'g')
plt.legend(loc = 'upper␣right')
plt.ylabel("Oil␣production[m3/s]", fontsize=15)
plt.title("Oil␣predictions,␣mse:␣" + "{:.4E}".format(Decimal(mse_oil_well1)), fontsize
    ↪ =25)
plt.subplots_adjust(top=1.5, right=1.5)
plt.xticks(fontsize=20)
plt.yticks(fontsize=15)
```

```
plt.subplot(2, 1, 2)
plt.plot(pred_gas_well1[:500], label = 'Predicted rate', color = 'black')
plt.plot(well1.y_gas[:500], label = 'True rate', color = 'g')
plt.legend(loc = 'upper right')
plt.xlabel("Time [h]", fontsize = 15)
plt.ylabel("Gas production[m3/s]", fontsize = 15)
plt.title("Gas predictions, mse: " + "{:.4E}".format(Decimal(mse_gas_well1)), fontsize
    ↪  = 25)
plt.tight_layout()
plt.xticks(fontsize=20)
plt.yticks(fontsize=15)

plt.subplots_adjust(top=1.5, right=2.5)

plt.savefig('BaselineWell1.eps', format='eps', bbox_inches='tight')

"""
Plot Results for validation set 2
"""

plt.subplot(2, 1, 1)
plt.plot(pred_oil_well2[:500], label = 'Predicted rate', color = 'black')
plt.plot(well2.y_oil[:500],label = 'True rate', color = 'g')
plt.legend(loc = 'upper right')
plt.ylabel("Oil production[m3/s]", fontsize = 15)
plt.title("Oil predictions, mse: " + "{:.4E}".format(Decimal(mse_oil_well2)), fontsize
    ↪ =25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=15)

plt.subplot(2, 1, 2)
plt.plot(pred_gas_well2[:500], label = 'Predicted rate', color = 'black')
plt.plot(well2.y_gas[:500], label = 'True rate', color = 'g')
plt.legend(loc = 'upper right')
```

```
plt.xlabel("Time [h]", fontsize = 15)
plt.ylabel("Gas production[m3/s]", fontsize = 15)
plt.title("Gas predictions, mse: " + "{:.4E}".format(Decimal(mse_gas_well2)), fontsize
    ↪ =25)
plt.tight_layout()
plt.xticks(fontsize=20)
plt.yticks(fontsize=15)

plt.subplots_adjust(top=1.5, right=2)

plt.savefig('BaselineWell2.eps', format='eps',bbox_inches='tight')
```

B.2 Bernoulli choke model

The code below was used to calculate the flow rates for oil and gas with Bernoulli choke model.

```
import pandas as pd
import pyfas as fa
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
import math
from scipy.interpolate import griddata
import time

class ChokeBernoulli:

    def Qchoke(self, Pwhcu, Pwhcd, Twhcu, Twhcd, C_op):

        Pwh = (Pwhcu+Pwhcd)/2
        Twhcu = Twhcu-273.15
        Twhcd = Twhcd-273.15
        Twh = (Twhcu+Twhcd)/2
        D = 0.189
        A1 = math.pi*(D/2)**2
```

```
    rho_oil = self.rho_oil_ip(Pwh, Twh)
    rho_gas = self.rho_gas_ip(Pwh, Twh)

    rho_mix = (rho_oil + rho_gas)/2

    x_gas = self.x_gas_ip(Pwhcd, Twhcd)

    Qchokemix = C_op*A1*math.sqrt(2*(Pwhcu-Pwhcd)/(rho_mix*(1/C_op**2 - 1)))

    vf_gas = 1/(1+(rho_gas/rho_oil)*(1/x_gas-1))
    vf_oil = 1/(1+(rho_oil/rho_gas)*(1/(1-x_gas)-1))

    QchokeGas = float(Qchokemix*vf_gas)
    QchokeOil = float(Qchokemix*vf_oil)

    return QchokeOil, QchokeGas


def __init__(self, PVTtable):

    PVTtable.export_all()
    Temperature_raw = PVTtable.data.T['TM'].values[0]
    Pressure_raw = PVTtable.data.T['PT'].values[0]

    x_gas_raw = PVTtable.data.T['RS'].values[0]
    x_gas_values = np.array(x_gas_raw).flatten()
    self.x_gas_ip = interp2d(Pressure_raw, Temperature_raw, x_gas_values)

    rho_oil_raw = PVTtable.data.T['ROHL'].values[0] #Density liquid
    rho_gas_raw = PVTtable.data.T['ROG'].values[0] #Density gas
    rho_oil_values = np.array(rho_oil_raw).flatten()
    rho_gas_values = np.array(rho_gas_raw).flatten()
    self.rho_oil_ip = interp2d(Pressure_raw, Temperature_raw, rho_oil_values)
    self.rho_gas_ip = interp2d(Pressure_raw, Temperature_raw, rho_gas_values)
```

B.3 Al-Safran choke model

The code below was used to calculate the flow rates for oil and gas with Al-Safran choke model.

```
import pandas as pd
import pyfas as fa
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
import math
from scipy.interpolate import griddata
from scipy.optimize import root


PVTtable = fa.Tab('/Users/eier/Documents/Project2019/PVTGOR100IP.tab')
df_train = pd.read_csv('/Users/eier/Documents/Project2019/ProjectTrain2.csv')


class AlSafran:


    def properties(self, Pwhcd, Twhcd, Pwhcu, Twhcu, C_op): #[Pa], [K], [Pa], [K], [-]
        #Convert from K to C
        Twhcd = Twhcd - 273.15
        Twhcu = Twhcu - 273.15
        #Define average wellhead pressure and temperature[Pa],[C]
        Pwh = (Pwhcu+Pwhcd)/2
        Twh = (Twhcu+Twhcd)/2
        #Average densities over choke[kg/m3]
        self.rho_oil = self.rho_oil_ip(Pwh, Twh)
        self.rho_gas = self.rho_gas_ip(Pwh, Twh)

        #Average viscosity over choke[Pa s]
```

```
        self.viscosity_oil = self.viscosity_oil_ip(Pwh, Twh)
        self.viscosity_gas = self.viscosity_gas_ip(Pwh, Twh)

        #Average heat capacities
        self.Cp_gas = self.Cp_gas_ip(Pwh, Twh)
        self.Cp_oil = self.Cp_oil_ip(Pwh, Twh)
        self.Cv_gas = self.cv_from_cp(self.Cp_gas)

        #heat capacity ratio
        self.k = self.Cp_gas/self.Cv_gas

        #specific volume gas upstream
        rho_gas_up = self.rho_gas_ip(Pwhcu, Twh)
        self.specific_volume_gas = 1/rho_gas_up

        #specific volume oil
        self.specific_volume_oil = 1/self.rho_oil

        #static mass fraction gas
        self.x_gas = self.x_gas_ip(Pwh, Twh)

        #used as initial guess for gas quality
        self.x_gas_quality = self.x_gas


    #Calculating the choke flow based pressure, temperature and choke opening
    def Qchoke_it(self, Pwhcd, Twhcd, Pwhcu, Twhcu, C_op):

        self.n = self.polytropic_exponent()

        self.R_subcritical = self.slip_ratio_subcritical()
        self.R_critical = self.slip_ratio_critical()
        self.alpha = self.alpha_ratio()
        self.rc = self.rcritical()
```

```
        Cd = 0.75
        C = 2000*Cd**2
        A1 = math.pi*(0.189/2)**2
        A2 = A1*C_op

        m2 = C*A2**2*Pwhcu*(self.alpha*(1-self.rc) +
        self.n/(self.n-1)*(1-self.rc**((self.n-1)/self.n)))/(self.x_gas_quality*self.
            ↪ spe
        cific_volume_gas*(self.rc**(-1/self.n) + self.alpha)**2*(self.x_gas_quality +
        1/self.R_critical*(1-self.x_gas_quality)))



        m = math.sqrt(m2)

        m_gas = m*self.x_gas_quality
        m_oil = m*(1-self.x_gas_quality)

        Q_gas = m_gas/self.rho_gas
        Q_oil = m_oil/self.rho_oil

        vgas = Q_gas/A2
        voil = Q_oil/A2

        return Q_oil, Q_gas, voil, vgas

    def Qchoke(self, Pwhcd, Twhcd, Pwhcu, Twhcu, C_op):

        #initialize the parameters
        self.properties(Pwhcd, Twhcd, Pwhcu, Twhcu, C_op)

        #initialize Qgas, Qoil, and velocities
        Q_oil, Q_gas, voil, vgas = self.Qchoke_it(Pwhcd, Twhcd, Pwhcu, Twhcu, C_op)

        #calculate gas quality from velocity
        self.x_gas_quality = vgas*self.x_gas_quality/(vgas*self.x_gas_quality + voil
```

```
        ↪ *(1-self.x_gas_quality))

    #initialize mass fraction
    x_gas_prev = self.x_gas

    error = math.inf

    while(error > 0.00001):

        Q_oil, Q_gas, voil, vgas = self.Qchoke_it(Pwhcd, Twhcd, Pwhcu, Twhcu, C_op)

        self.x_gas_quality = vgas*self.x_gas/(vgas*self.x_gas + voil*(1-self.x_gas)
            ↪ )

        #until gas quality has converged
        error = abs(self.x_gas-x_gas_prev)

        x_gas_prev = self.x_gas_quality

    return Q_oil,Q_gas


def slip_ratio_subcritical(self):
    a0 = 1
    a1 = 1
    a2 = -0.83
    a3 = 0
    R = a0*((1-self.x_gas_quality)/self.x_gas_quality)**(a1-1)*(self.rho_oil/self.
        ↪ rho_gas)**(a2+1)*(self.viscosity_oil/self.viscosity_gas)**(a3)
    return R

def slip_ratio_critical(self):
    R = math.sqrt(1+self.x_gas_quality*(self.rho_oil/self.rho_gas -1))*(1+0.6*math.
        ↪ exp(-5*self.x_gas_quality))
    return R
```

```
def alpha_ratio(self):
    alpha = self.R_critical*(1-self.x_gas_quality)*self.specific_volume_oil/(self.
        ↪ x_gas_quality*self.specific_volume_gas)
    return alpha

def cv_from_cp(self,Cp): #For gas
    R = 8.314472 #[J/(mol*K)]
    Mm = 16.042/1000 #[kg/mol] Assume we have most methane, molar mass = 16.024
    R_m = R*(1/Mm) #[J/(kg*K)]
    Cv = Cp - R_m
    return Cv

#calculate polytropic exponent
def polytropic_exponent(self):

    self.Cp_gas = self.Cp_gas/1000
    self.Cp_oil = self.Cp_oil/1000
    self.Cv_gas = self.Cv_gas/1000
    n = (self.x_gas_quality*self.k*self.Cv_gas + (1-self.x_gas_quality)*self.Cp_oil
        ↪ )/(self.x_gas_quality*self.Cv_gas + (1-self.x_gas_quality)*self.Cp_oil)
    return n

#calculate critical pressure ratio
def f_r(self,x):
    return ((self.alpha*(1-x) + self.n/(self.n-1))/(self.n/(self.n-1) + self.n
        ↪ /2*(1+self.alpha*x**(1/self.n))**2))**(self.n/(self.n-1))-x

def rcritical(self):
    sol = root(self.f_r, 0.5)
    self.rc = float(sol.x)
    return self.rc

def __init__(self,PVTtable):
```

```
PVTtable.export_all()

Temperature_raw = PVTtable.data.T['TM'].values[0]
Pressure_raw = PVTtable.data.T['PT'].values[0]



x_gas_raw = PVTtable.data.T['RS'].values[0]
x_gas_raw = np.array(x_gas_raw)
x_gas_values = x_gas_raw.flatten()
self.x_gas_ip = interp2d(Pressure_raw, Temperature_raw, x_gas_values)

rho_oil_raw = PVTtable.data.T['ROHL'].values[0] #Density liquid
rho_gas_raw = PVTtable.data.T['ROG'].values[0] #Density gas
rho_oil_values = np.array(rho_oil_raw).flatten()
rho_gas_values = np.array(rho_gas_raw).flatten()
self.rho_oil_ip = interp2d(Pressure_raw, Temperature_raw, rho_oil_values)
self.rho_gas_ip = interp2d(Pressure_raw, Temperature_raw, rho_gas_values)

viscosity_oil_raw = PVTtable.data.T['VISHL'].values[0]
viscosity_gas_raw = PVTtable.data.T['VISG'].values[0]
viscosity_oil_values = np.array(viscosity_oil_raw).flatten()
viscosity_gas_values = np.array(viscosity_gas_raw).flatten()
self.viscosity_oil_ip = interp2d(Pressure_raw, Temperature_raw,
    ↪ viscosity_oil_values)
self.viscosity_gas_ip = interp2d(Pressure_raw, Temperature_raw,
    ↪ viscosity_gas_values)

Cp_gas_raw = PVTtable.data.T['CPG'].values[0]
Cp_oil_raw = PVTtable.data.T['CPHL'].values[0]
Cp_gas_values = np.array(Cp_gas_raw).flatten()
Cp_oil_values = np.array(Cp_oil_raw).flatten()
self.Cp_gas_ip = interp2d(Pressure_raw, Temperature_raw, Cp_gas_values)
self.Cp_oil_ip = interp2d(Pressure_raw, Temperature_raw, Cp_oil_values)
```

B.4 ”No pressure wave” tubing equation

The code below was used to calculate the flow rates for oil and gas with ”No-pressure-wave” tubing equation.

```
import pandas as pd
import pyfas as fa
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
import math
from scipy.interpolate import griddata
from scipy.optimize import root


PVTtable = fa.Tab('/Users/eier/Documents/Project2019/PVTGOR100IP.tab')

PVTtable.export_all()


class QtubingNPW:

    def Qtubing(self, Pbh, Tbh, Pwh, Twh, length, theta):

        rho_oil_wh = self.rho_oil_ip(Pwh, Twh)
        rho_gas_wh = self.rho_gas_ip(Pwh, Twh)

        rho_oil_bh = self.rho_oil_ip(Pbh, Tbh)
        rho_gas_bh = self.rho_gas_ip(Pbh, Tbh)

        rho_mix_wh = (rho_oil_wh + rho_gas_wh)/2
        rho_mix_bh = (rho_oil_bh + rho_gas_bh)/2

        self.rho_mix = (rho_mix_wh + rho_mix_bh)/2
```

```
    viscosity_oil_wh = self.viscosity_oil_ip(Pwh, Twh)
    viscosity_gas_wh = self.viscosity_gas_ip(Pwh, Twh)

    viscosity_oil_bh = self.viscosity_oil_ip(Pbh, Tbh)
    viscosity_gas_bh = self.viscosity_gas_ip(Pbh, Tbh)

    viscosity_mix_wh = (viscosity_oil_wh + viscosity_gas_wh)/2
    viscosity_mix_bh = (viscosity_oil_bh + viscosity_gas_bh)/2

    self.viscosity_mix = (viscosity_mix_wh + viscosity_mix_bh)/2

    self.D = 0.189
    self.A = math.pi*(self.D/2)**2
    self.length = length
    self.H = math.cos(theta*math.pi/180)*length

    Q_guess = 1
    self.Q_curr = Q_guess
    f = self.friction_factor()

    error = math.inf

    while (error > 0.00001):
        Q = self.Q_calc(Pbh, Twh)
        error = abs(Q-self.Q_curr)
        self.Q_curr = Q
        f = self.friction_factor()

    return Q


def ff(self,f):
    Re = self.rho_mix*self.Q_curr*self.D/(self.viscosity_mix*self.A)
    roughness = 0.045
```

```
    return (1/math.sqrt(f) + 2*math.log10(roughness/(3.7*self.D) + 2.51/(Re*math.
        ↪ sqrt(f))))

def friction_factor(self):
    sol_f = root(self.ff, 0.1)
    self.f = float(sol_f.x)
    return self.f

def Q_calc(self, Pbh, Pwh):
    g = 9.81
    Qtm = math.sqrt((Pbh - Pwh - self.rho_mix*g*self.H)*math.pi**2*self.D**5/(8*
        ↪ self.f*self.length*self.rho_mix))
    return Qtm


def __init__(self, PVTtable):


    PVTtable.export_all()

    Temperature_raw = PVTtable.data.T['TM'].values[0]
    Pressure_raw = PVTtable.data.T['PT'].values[0]

    x_gas_raw = PVTtable.data.T['RS'].values[0]
    x_gas_raw = np.array(x_gas_raw)
    x_gas_values = x_gas_raw.flatten()
    self.x_gas_ip = interp2d(Pressure_raw, Temperature_raw, x_gas_values)

    rho_oil_raw = PVTtable.data.T['ROHL'].values[0] #Density liquid
    rho_gas_raw = PVTtable.data.T['ROG'].values[0] #Density gas
    rho_oil_values = np.array(rho_oil_raw).flatten()
    rho_gas_values = np.array(rho_gas_raw).flatten()
    self.rho_oil_ip = interp2d(Pressure_raw, Temperature_raw, rho_oil_values)
    self.rho_gas_ip = interp2d(Pressure_raw, Temperature_raw, rho_gas_values)
```

```
        viscosity_oil_raw = PVTtable.data.T['VISHL'].values[0]
        viscosity_gas_raw = PVTtable.data.T['VISG'].values[0]
        viscosity_oil_values = np.array(viscosity_oil_raw).flatten()
        viscosity_gas_values = np.array(viscosity_gas_raw).flatten()
        self.viscosity_oil_ip = interp2d(Pressure_raw, Temperature_raw,
            ↪ viscosity_oil_values)
        self.viscosity_gas_ip = interp2d(Pressure_raw, Temperature_raw,
            ↪ viscosity_gas_values)
```