Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Megha Rajasekhar

Data-Driven Virtual Flow Metering using Neural Networks

Master's thesis in Chemical Process Engineering
Supervisor: Johannes Jäschke
August 2022

NTNU
Norwegian University of
Science and Technology

Megha Rajasekhar

Data-Driven Virtual Flow Metering using Neural Networks

Master's thesis in Chemical Process Engineering
Supervisor: Johannes Jäschke
August 2022

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

,

NTNU
Kunnskap for en bedre verden

Department of Chemical Engineering

TKP 4900 - Chemical Process Technology Masters

Data-Driven Virtual Flow Metering using Neural Networks

Author:
Megha Rajasekhar

August, 2022

Table of Contents

List of Figures

List of Tables

Preface

This master thesis was written in the Spring of 2022 in the course TPK4900 - Chemical Process Technology Master's Thesis, and completes my M.Sc. in Engineering (IKP) at NTNU. I have had an interest in fluid modelling and am eager to apply concepts from data science in this field to develop simpler methods to replace conventional calculations. This master thesis is an effort towards towards applying machine learning constructs in the field of fluid dynamics.

Acknowledgments

I would like to thank my supervisor Johannes Jäschke, who has given me the opportunity to work on the topic of Virutal Flow Metering for my specialisation project and master thesis. I am grateful for his immense patience and understanding as he has guided me through this journey with my specialisation and master project. I also want to thank co-student Aksel Madslien and Md Rizwan for valuable discussions during the projects and guidance with statistical subjects that were new to me.

1 Introduction

1.1 Background and Motivation

This masters thesis is based on applying machine learning concepts to the field of flow metering. Thus the motivation is two-fold: the first part covers background on virtual flow metering and the second states motivation over the inclusion of data science principles.

1.1.1 Need for Virtual Flow Metering

Flow measurement in oil and gas production is necessary, especially relative flow rates of oil and gas, as they dictate well management and are important for optimal production from a well or even network of wells.

Physical methods of measuring relative flow rates include well testing, conventional and deductive, where multiphase flows are sent into a separator and single-phase flows are measured. This could be done directly by sampling production fluid from each well, known as conventional well testing. Another way to achieve the purpose would be to turn off the well in the network, measure the single-phase flow rates of the rest of the network via a separator and back-calculate the flow rates of the desired well. This is known as deduction well testing. These methods require a separate flowlines and separators adding to costs and can cause production losses [13]. Moreover, these operations do not perform well if continuous information is required about the flow rates.

Multiphase flow meters (MPFMs) constitute another physical method, where flow rates of all phases are measured without separation and at real time. Such meters can be installed at well-heads and use indirect measurements such as phase velocities and phase fractions to estimate flow rates [2]. Since a typical well production would contain oil, water, gas, sand, wax and other solid impurities, these can block and even damage the MPFMs leading to their failure to estimate flow rates accurately. This is further detrimental as these meters are expensive to procure and maintain.

The unreliable and round-about nature of physical methods of flow metering call for indirect methods of measurement where available field data can be utilised to estimate flow rates. Virtual Flow Metering (VFM) achieves this purpose by using easily available data such as bottom-hole pressures and temperature, downstream pressures and temperatures and choke openings to estimate multiphase flow rates periodically or in real-time [1]. Since VFM only requires readily available data, it does not call for the installation of additional physical systems and can thus be implemented even on existing fields at any phase in their lifetime. In essence, VFM is a computational model that processes available data to produce flow rates.

Based on the types of models used VFM can be divided into two approaches: First Principles VFM and Statistical/Data-driven VFM. First principles VFM is based on physical or mechanistic models where entire networks or sub-sections of it are modelled to replicate the field in flow behaviour. One is required to solve conservation equations for momentum, mass and heat which are dynamic in nature [5]. This form of modelling can be complex as it requires knowledge of empirical correlations that determine relationships between pressures and flow rates as well as constraints that accompany them. These methods however, produce robust models due to their rigorous understanding of the flow behaviour in a network at the expense of computational power. On the other hand, data-driven models seek to statistically alter input to produce desired flow rates. Machine learning models examine trends in the relationship between already available input and output data and fit a mathematical model on them which is used to predict outputs for new input data [16]. Such models can provide accurate real-time data but are heavily dependant on the availability of quality data that is used to train the model.

1.1.2 Machine Learning Models

Machine learning (ML) models aim to 'learn' trends and relationships between different parameters set as inputs and outputs to predict outcomes for similar set of data. All of this is carried out without knowing the actual physical behaviour of the system. In the field of VFM, it is a simpler way of achieving the purpose of flow rate prediction without needing to have extensive knowledge about flow modelling which is more complicated in the instance of multiphase flow.

The process of building a machine learning model begins with collecting data and processing. This entails differentiating input data from output data and selecting certain 'features' which are indirectly related to the output and can increase the chances of the ML model towards fitting a better model on the data. It is also useful to have a sufficient amount of noisy data, i.e. data with outliers so that model can cope with irregular data and prevent over-fitting. In some cases, this is induced artificially if the original data is not sufficiently noisy. The input data is then divided into training and testing sets where the training set is used to optimize the model and the testing set is to test the accuracy of the trained model. Once test, the model can be re-calibrated to better predict outputs. After segregating the data, the model is trained by optimizing some of its parameters to minimise a certain cost function. Usually a cost function is designed so that the error in prediction of output is minimized. A trained ML model is essentially a mathematical model that is fitted on the inputs so as to give its desired outputs. This trained model is used to predict outputs for a new set of inputs [16].

Neural networks (NNs) are one such way of creating an ML model. Feed-forward multi-layer NNs include an input and an output layer and several hidden intermediate layers which produce a highly non-linear relationship between input and output [4]. Although one does not need to know the exact relationship between the inputs and outputs, specific knowledge about the the field of application is always helpful to nudge the machine learning model towards replicating similar functions. Such NN models, have been observed to work well for steady state flow scenarios as compared to transient flow [10].

Other methods like, linear regression can be employed for linear relationships. In the case of there existing a non-linear relationship between the suggested input and outputs, the inputs are modified such that a linear relationship is exploited. Several improvements on the NN have been discussed in detail in a literature review [16] such as the combination of NNs with ensemble learning methods like regression tree method, adding more features as inputs using principle component analysis, using recurrent neural networks and space vector models.

1.2 Problem Formulation

In this project, we aim to predict phase flow rates for oil, water and gas for several flow scenarios in a single horizontal pipe. Four different types of neural network models are built with variations in the number of inputs and hidden layers. Pressures at different positions in the pipe are considered as inputs and the flow rate of each individual phase are considered as the outputs. The data-sets vary in increasing complexity with changes in the fluid composition (Gas-Oil Ratio and Watercut) as well as boundary conditions (outlet pressure and operating temperature).

The model is then trained and its hyperparamters also optimized for different data-sets. K-fold cross-validation is used to set the regularization co-efficient by further dividing the training set into a number of folds. The various terms related to a neural network and how it is built is described in Section 2. The methodology used to train the NN is described in Section 4.

The data is generated from an OLGA model and a general methodology is created for generating simulation files and extracting data from the OLGA simulations. The OLGA models and supporting methodologies are described in Section 3.

1.3 Related Work

Much work has been carried out towards prediction flow rates for transient flow scenarios. The usage of NNs is adapted for this application in the form of recurrent NNs which use input data from previous time-steps to predict output at the current time-step. Long-Short Term Memory is a commonly used recurrent NN model as seen in [3]. Apart from NNs, industrially established processes such as the FieldWare Production Universe created by Shell are also in application [8]. This system includes well testing of all individual wells in a network, with additional step-wise changes in the well parameters to generate a data that may apply to a plethora of flow scenarios. Such data from each of the wells is combined to generate a total network flow rates. This total network flow rate can also be generated by multiphase flow modelling carried out in the same manner. The data thus generated can be used to train ML models and then used for future prediction [11]. Many instances also combine linear regression techniques along with NNs to increase the efficiency of the NN models. Once adapted for transient flow conditions, VFM models can be used to set up online or real-time prediction systems as seen in [11].

Another development in VFM systems is the appearance of hybrid models or grey box models which combine first principles models ensemble learning methods. Since first principles models are well studied in the oil and gas field, they can leveraged to provide additional data to statistical VFM systems. Physical models replicating fields and networks can be used to generate data in the event of unavailability, especially for transient flow scenarios. Such a hybrid model would be able to provide data for different stages of field life and fill gaps in the data which may not be available due to uncertainty in measurements or inability to halt production for the sake of data gathering [9].

2 Neural Networks (NNs)

Deep feed-forward networks or feed-forward neural networks or multi-layer perceptrons(MLPs), are the most popular ML models. A NN aims to approximate a function so as to produce the same output the function would produce for a given set of inputs within some margin of error.

Due to the nature of NN, it can approximate any form of relationship between its inputs and outputs from one dimensional space to another within a desirable level of accuracy. Thus it can be considered to be a universal approximator ([7]).

2.1 Principle of an NN

The goal of any ML model is to learn from a set of data to be able to predict certain outcomes for an unknown set of data. The process of learning is constituted into three parts: learning from an *experience*, to *perform* a *task* and upto a certain level of accuracy. *Experience* constitutes the data used to train the model known as the training set. The data in a training set is divided into inputs and outputs. Inputs are the known data and outputs are the data-sets that are to be predicted by the model. If inputs are x and outputs are y, the goal of the NN is to generate a model f such that $f(x) = y$. The *task* is the manner in which the learnt data is processed. The task of this NN is regression type. In this, the model f is mapped from $\mathbb{R} \to \mathbb{R}$. The *performance* of the model is checked using a cost function. Most cost functions aim to minimise the error between predicted outputs and actual outputs for the training data. The NN is then parameterised so as to minimise this cost function. The trained (optimised) model is then tested by feeding a set of data, separate from the training set, called the test data. The accuracy of the model is evaluated by comparing the predicted outputs of the test data to its original outputs [4].

2.2 Structure of a NN

The main components of an NN are activations. An activation is can be any function that transforms the input it receives. Activations are arranged into layers. A neural network can have several layers with varying numbers of activations. The first and last are the input and output layers respectively as the input layer receives the input data of the NN and the activations of the output layer deliver the final output of the NN. Hence the input layer has the same number of activations as the number of inputs and the output layer has same number of activations as the number of outputs. The rest of the intermediate layers are known as the hidden layers. The number of layers in an NN determines its *depth*. The number of activations determine the *width* of the model. T The resulting functions from the output layer activations are used in cost function. The number of layers and the number of activations in the hidden layers form the architecture of the NN [4].

In a NN input data is received at the input layer and the input activations transform the input using the activation function. This transformed data is then passed on to the next layer. The data from each of the input activations are each weighted by some factor θ and added. This sum is passed on to every activation of the next layer. However, the weights corresponding to each activation in the subsequent layer are different. To train a neural network to a given set of data, the θ parameters are optimised so as to minimise the cost function. To further tune the model, to make it more robust, the architecture of the NN i.e. the number of layers and number of activations in the layers can also be altered. However, this is more complicated as there are few established procedures for this.

The cost function for a neural network often includes a regularization co-efficient. Its purpose is to limit the value of the θ parameters to prevent it from becoming too large. It can also be used to weight the minimisation of the prediction error over the values of the θ parameters. Such weighting coefficients, known as hyperparameters, are optimized outside of the learning process and require additional steps to be finalised.

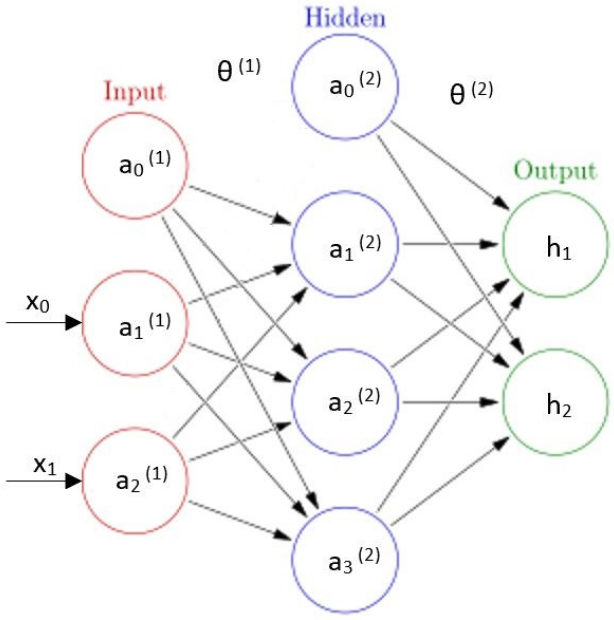


Figure 1: Structure of a Neural Network

2.3 Forward and Back-propagation

2.3.1 Forward Propagation

The forward movement of information from layer to layer in a NN is called forward propagation.In the example shown in 1, the inputs of one training sample are received at nodes $a_1^{(1)}$ and $a_2^{(1)}$. The nodes $a_0^{(1)}$ and $a_0^{(2)}$ are bias nodes. The bias nodes are not counted while expressing the width of the layer i.e in the example, the input layer is considered to have 2 nodes/activations and the hidden layer is considered to have 3 nodes/activations. The inputs are transformed by the activation function of the input nodes, weighted by some parameter θ, summed with the value of the other nodes and sent forward to the next layer. For example, at the second layer of the NN we have

$$a_1^{(2)} = z(a_0^{(1)}\theta_{1,1}^{(1)} + a_1^{(1)}\theta_{1,2}^{(1)} + a_2^{(1)}\theta_{1,3}^{(1)}) \tag{1}$$

$$a_2^{(2)} = z(a_0^{(1)}\theta_{2,1}^{(1)} + a_1^{(1)}\theta_{2,2}^{(1)} + a_2^{(1)}\theta_{2,3}^{(1)}) \tag{2}$$

$$a_3^{(2)} = z(a_0^{(1)}\theta_{3,1}^{(1)} + a_1^{(1)}\theta_{3,2}^{(1)} + a_2^{(1)}\theta_{3,3}^{(1)}) \tag{3}$$

This can also be compactly represented for the hidden layer as

$$a^{(2)} = z(\Theta^1 \cdot a^{(1)}) \tag{4}$$

and for the output layer as

$$h = z(\Theta^2 \cdot a^{(2)}) \tag{5}$$

Here, $z()$ is the activation function, $a^{(1)}$, $a^{(2)}$ and $a^{(3)}$ are column vectors containing each of the activations of the respective layer and $\theta^{(1)}$ and $\theta^{(2)}$ are matrices that contain the θ parameters. For this example, the size of $a^{(1)}$ is 2x1, of $a^{(2)}$ is 3x1 and of h is 2x1. This is because the bias nodes are not included as they have a fixed pre-defined value unaffected by the input data. The sizes of $\theta^{(1)}$ and $\theta^{(2)}$ are 3x3 and 2x4 respectively. This is determined by the sizes of the layers that the θ parameters connect. The size of a Θ matrix is defined as $(l^s x l^p + 1)$ where l^s is the number of activations in the subsequent layer and l^p is the number of activations in the preceding layer.

2.3.2 Jacobian of Cost Function

An essential step of the learning process is to optimize the θ parameters with in order to minimize the cost function. There are a plethora of optimization algorithms that can be applied to achieve this purpose. However, most methods require the Jacobian of the cost function with respect to the optimization parameters, which in this case would be $\frac{\delta J(\Theta)}{\delta \Theta_{i,j}}$ where J is the cost function and Θ is each of the θ parameters.

For NNs a back-propagation algorithm can be used which simplifies the process of calculating the Jacobian of the cost function. For the example in 1 with 3 layers, the back-propagation algorithm is given as

$$\delta^{(3)} = h \tag{6}$$

$$\delta^{(2)} = (\Theta^{(2)T}\delta^{(3)}) \cdot z'(\Theta^{(1)}a^{(1)}) \tag{7}$$

where $z'()$ is the differential of the activation function with respect to θ. For the first training example

$$\Delta = a^{(2)}\delta^{(2)} + a^{(3)}\delta^{(3)} \tag{8}$$

This is repeatedly summed for all the training examples.

3 Data Generation Using OLGA

This section describes how the flow modelling software OLGA was used to generate the training and testing example sets for this study. This is in continuation to the Specialization Project done as a preamble to the Master Thesis. A smaller section of the entire model, used in the Specialization Project, has been focused on and been amended in various ways to generate different types of data-sets.

3.1 OLGA Flow Modelling

The software OLGA (Version 2019) from Schlumberger was chosen to carry out the multiphase flow modelling of the oil and gas network. OLGA is an industry standard multiphase flow simulator which uses one dimensional modelling. It uses a three-phase model, that is it applies the conservation equations to the three phases separately.

The solver includes a steady-state pre-processor which generates its own initial conditions. The solver starts with a fully filled flowline based on these initial conditions and then converges to the required steady-state.

Each flowline is divided into discrete pipe sections and segments. The model solves the mass conservation equations for each phase separately as well as for oil and water droplets in the gas phase. Similarly, it solves momentum conservation equations for each discrete phase. In addition, the phases are linked by mass transfer (gas-condensate equilibrium). An energy balance equation is also solved assuming that the temperature is constant across the phases [15].

The Black-oil composition feature has been used in this project where the specific gravity of each phase is input to OLGA and the rest of the PVT properties are generated by the solver itself using a set of in-built Black-oil correlations. Based on the chosen values of oil specific gravity, the Lasater correlation was selected [6].

3.2 Horizontal Pipeline Model

The model consists of a single horizontal pipeline with two boundaries specified: an inlet and an outlet. The boundaries are modelled as nodes in OLGA. The inlet node is a closed node whereas the outlet node is a pressure node where pressure can be specified. Additionally a mass source is added at the first section of the pipeline which introduces flow in to the line.

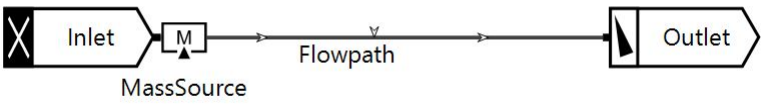


Figure 2: OLGA Pipeline Model

3.2.1 Assumptions

Temperature calculations were modified in order to ensure an isothermal model. Due to flow friction, pressures usually decrease along the length of a horizontal flowline in the direction of flow. This decreasing pressure may cause some of the oil to enter the gas phase resulting in lowering of temperature. This was offset by setting the temperature calculations to isothermal mode.

The steady-state pre-processor was used to ensure that the model reached steady state, avoiding any transient state data.

Additionally, different flow rates were tried to ensure that all flow regimes were incorporated. The presence of less favourable flow regimes such as annular flow and slug flow introduce noise into the system as they cause large periodic fluctuations in boundary pressures as well as flow rates and are calculated differently from the standard stratified flow regime [15].

3.2.2 Model Parameters

All the different flowline and fluid parameters used in the model are listed in Tables 1 and 2. These properties are listed as "Base" properties as they are the default properties in some data sets. In data sets where this parameter is a variable, randomly generated values, within a reasonable range, replace these "base" values.

Parameter	**Value**
Outlet Pressure	10 bara
Inlet Temperature	25 deg C
Length	1000 m
Diameter	0.2 m
Roughness	$3e-5$ m

Table 1: Base Pipeline Parameters

Parameter	**Value**
Oil Specific Gravity	0.867
Water Specific Gravity	1.020
Gas Specific Gravity	0.814
GOR	50 sm^3/sm^3
Watercut	0.3

Table 2: Fluid Properties

3.3 Data Generation

An OLGA simulation can be run using an .inp file where all specifications of the pipeline geometry, time-step sizes, fluid properties, boundary conditions and requirements of outputs can be listed. The .inp file contains all the details of the simulation.

To generate a large of amount data with one or two variables, a python script was coded to replicate a base .inp file with different parameters such as different flow rates or different GORs. In this manner 5 different data sets were generated:

- Constant GOR
- Variable GOR
- Variable GOR and Water cut (WC)
- Variable Outlet Pressure
- Variable GOR, WC and Outlet Pressure

Four different parameters were changed across the data sets namely, Flow Rate, GOR, Water cut and Outlet Pressure. The values for these parameters were randomly generated and directly plugged into the input files. Changing any of these four parameters would be reflected in the change in the pressure at the inlet of the flowline and hence, the pressure differential across the line.

A large number of simulation files could be run together one after the other by calling a .bat file which sequences the files and ensure that they have run and their results are stored in the required location. The results of an OLGA simulation, are stored in .ppl and .tpl files. These files have been read and required results have been extracted using another python script dedicated to result extraction. The results extracted from each simulation were:

- Inlet Pressure
- Outlet Pressure
- Oil Flow Rate
- Gas Flow Rate
- Water Flow Rate

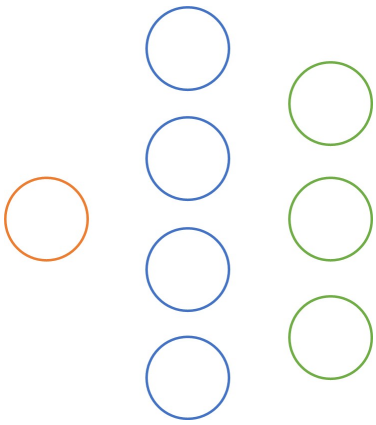
These results have been divided into inputs: Inlet and Outlet Pressure and outputs: Flow Rats. All files and generated data can be found in the link in the Appendix.

4 Methodology

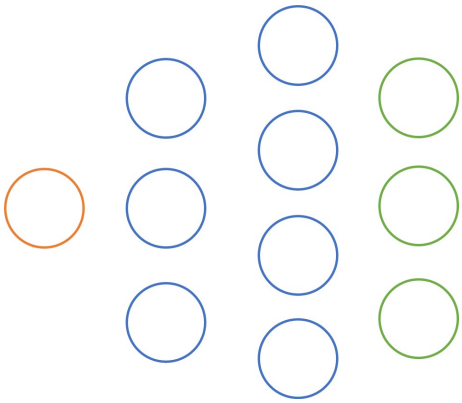
For this study 4 different types of neural network models were generated which can be classified on two basis: number of inputs taken and number of hidden layers. The models can be described as follows:

- Model 1: Accepts 1 input, provides 3 outputs and has one hidden layer with 4 activations.
- Model 2: Accepts 1 input, provides 3 outputs and has two hidden layers with 3 and 4 activations each.
- Model 3: Accepts 2 inputs, provides 3 outputs and has one hidden layer with 4 activations.
- Model 4: Accepts 2 inputs, provides 3 outputs and has two hidden layers with 3 and 4 activations each.

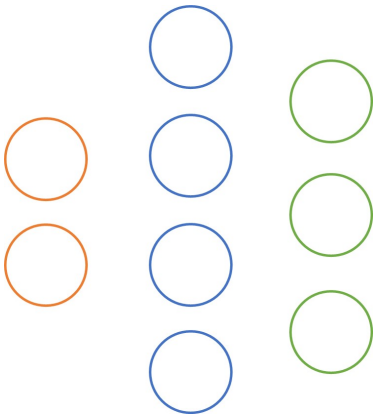
The aim of this virtual flow metering study is to use pressures across a horizontal pipeline to predict the multiphase flow rate of fluids flowing through that line. Two forms of pressure measurement are used in this study: one is a single input the overall loss of pressure across the line (difference between inlet and outlet pressure) and the other is two inputs with the inlet and outlet pressures separately.Therefore the models take either one or two inputs to produce three outputs which are the flow rates of oil, gas and water respectively.



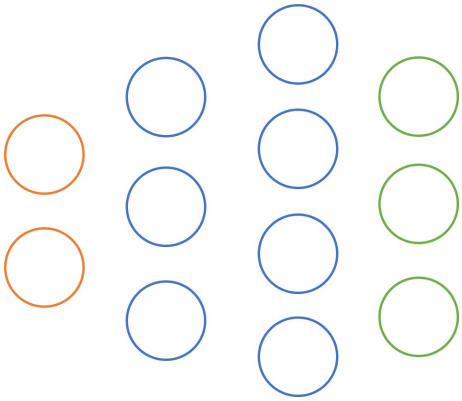
((a)) Model 1



((b)) Model 2



((c)) Model 3



((d)) Model 4

Figure 3: Structures of the different NN models

4.1 Specifications of the NN model

4.1.1 Activation Function

The activations function used in a NN determines the function that it carried out. For example, a sigmoid activation function is used for the function of data classification as it limits the mapped value of any input between 0 and 1. However, since predicting flow rates from pressure data is a form of regression, the NN models created in this study use the ReLU activation function. The Rectified Linear Unit (ReLU) is a piece-wise linear function which aims to remove the occurrence of negative values during the forward propagation of the NN. It directly outputs the input value if it is greater than 0 and outputs 0 is the input is lesser than or equal to 0 [12].

4.1.2 Cost Function

For the purpose of error minimization, a least-squares error function is used. This cost function aims to minimize the square of the error between the actual required output and the NN's predicted output. It also includes a regularization parameter, λ. The function is as follows

$$J = \sum_{i=1}^{3}(h_i - y_i)^2 + \sum_{l=1}^{L-1}\sum_{i}\sum_{j}\left(\Theta_{i,j}^{(l)}\right)^2 \tag{9}$$

where L is the number of layers in the NN. ‘ The regularization co-efficient λ is used to limit the values of the θ parameters. The value of λ can be adjusted to shift the weight of cost optimization from the error minimisation to reducing the values of the θ parameters. In this study, the optimum value of λ was chosen on the basis of the least prediction error for the training set. Selecting λ on this basis is known as *cross-validation*. There exist several techniques to carry out cross-validation, the method chosen for this study is explained in later sections.

4.1.3 Normalization

Since the data used for training and testing has been extracted from OLGA simulation results, the units of the pressure and flow rate values are set to the default of OLGA. The pressure values are output in Pascal, Pa and therefore are of the order 10^6. The flow rate values are output in cubic meters per second, m^3/s and are of the order 10^0 to 10^1. There is a large difference in the scale of the inputs and outputs which may lead to difficulties in optimizing the θ parameters. To counteract this, the inputs are *normalized* to match the order of the outputs. All pressure measurements are converted from Pa to bar by factoring them by 10^{-5}.

4.2 Cross Validation

Cross validation is a procedure carried out on a NN model to validate and fine-tune it before it is implemented on a testing set. It can also simultaneously be used to optimize the regularization parameter λ. In other cases, it can also be used to optimize or check other hyperparameters such as the number of hidden layers, number of activations, etc.

K-Fold is a widely used cross validation method and is applied in this study as well. It is carried out by dividing the training set further into training and validating sets. The training set is first divided into k number of *folds* and the cross-validation algorithm is run in a loop for k number of iterations. In this study, the training set was divided into k folds such that each fold has between 20 to 30 examples in it. In each iteration i, the ith fold of the data becomes the validating fold and all other folds constitute the training folds. In each iteration, the model is trained by the training folds and then used to predict the outputs of the validating fold. The error between the actual outputs and predicted outputs of the validating folds of each iteration are averaged to produce one error value for the NN model. This process can be repeated with different values of λ to finalize

a value which has the least validation error. The chosen λ is then plugged into the cost function and the NN model is trained over the entire training set.

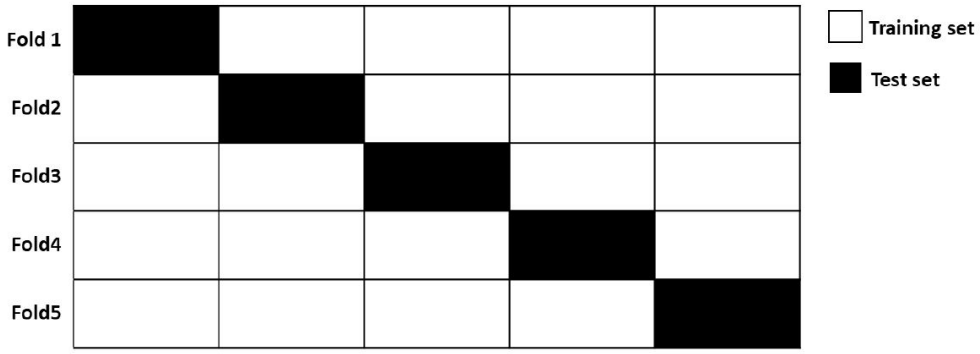


Figure 4: Example of Cross Validation with 5 folds [14]

The main assumption to be satisfied when carrying out cross-validation by the K-Fold method is that each example in the training set is independent of each other. This criteria is fulfilled for this study as flow rate prediction is being carried out for isolated instances of steady flow in a pipeline. One drawback of this method is that the λ values are limited to only those values which are tested and hence this process may not even check more optimal values of λ.

4.3 Training and Testing Procedure

The following paragraph outlines a step by step procedure followed on MATLAB through which the NN models built are trained, validated and tested. Each model has been trained with different types of OLGA generated data to produce one error value. This has been done to evaluate the differences in prediction among the four models. Also, different types of training and testing sets are used to find the change in error for increasing complex changes in the data.

Firstly, the data is imported into the MATLAB workspace and divided into a training and a testing set. The inputs of both sets are then normalized. The cost function is set up and the values of λ and the θ parameters are initialised with random values. Cross validation is then carried out as explained in the previous section. In the cross validation procedure indices ranging from 1 to k are randomly assinged to each example in the training set. This is done using the MATLAB function *crossvalind*. Then all examples having the same index are collected into a fold. The indices are assigned such that each fold has roughly a similar number of examples in it. The cross validation procedure is carried out for different ranges of λ and are gradually narrowed down to obtain an integer value. The chosen value of λ is then plugged into the cost function and it is optimized over the entire training set to minimize the cost function and obtain a set of θ parameters. The NN model along with these optimum θ values constitute the trained model. The inputs of the testing set are fed to the trained model (Boundary pressures or pressure differential) are fed to the NN so as to predict their outputs (multiphase flow rates). The predicted outputs are compared with the actual outputs of the testing set and their relative errors are calculated. The errors are averaged over the number of examples in the testing set to produce one error value which is considered to the be the *result* of each NN model for the given type of data set.

5 Results and Conclusions

5.1 Performance of NN Models

As mentioned in the previous section, each model was tested with different data sets to check their relative performance as well as their ability to handle more and more non-uniformity in the data. The relative error between the predicted outputs and the actual outputs of the testing data, averaged over the number of testing examples has been reported as the result of each model for each instance of use. All these results have been tabulated in Table 3.

Data Set	Model Type	Error
Constant GOR	Model 1	0.0512
	Model 2	0.9904
	Model 3	2.0891
	Model 4	5.4473
Variable GOR	Model 1	0.2147
	Model 2	0.2506
	Model 3	0.6202
	Model 4	1.0416
Variable GOR & WC	Model 1	4.9013
	Model 2	2.6594
	Model 3	1.5205
	Model 4	2.6184
Variable Outlet Pressure	Model 1	0.3632
	Model 2	0.3591
	Model 3	0.3504
	Model 4	0.7907
Variable GOR, WC and Outlet Pressure	Model 1	1.8805
	Model 2	1.9506
	Model 3	1.8999
	Model 4	1.7737

Table 3: Results for each Model using each Data Set

From the results, the general trend is that Model 1 performs better for lesser complex data. As the complexity of the data increases, all models show a similar range of error indicating that the models are too simple to grasp the varying trends. Also, in general Models 1 and 2 perform better than Models 3 and 4 suggesting that the model is not able to create a relation between the inlet and outlet pressures, but are able to directly map differential pressure to the flow rates.

5.2 Discussion and Conclusion

From the results, the main conclusion is that it is better to use differential pressure as an input to the models instead of providing the inlet and outlet pressure separately. In the instance that there are more inputs than simply the boundary pressures, such as temperature, fluid properties, etc, pre-processing the data to combine two inputs into one would be sensible and avoid the model to have to make further connections between the inputs.

The significant amount of error in all models can be attributed to two reasons: insufficient number of training examples and over-fitting of model to the training data. The first reason is straightforward in that using more training examples will ensure that the training data is more diverse and would make the model more robust.

Over-fitting of the NN model can arise due to lack of sufficient noise in the training data. Since the data is generated in OLGA, it lacks the noise that is part of general field measurements which may be induced due to discrepancies in sensor or general human error. This can contribute towards

the model being highly specific towards the training data and unable to predict trends in new data it has not encountered before. Another more probable reason for over-fitting can be due to the inefficient estimation of the λ co-efficient. A highly specific λ can cause the NN model to also become highly specific towards the training data and be unable to fit in new data. This was also observed when the cross-validation errors while optimizing λ were much lower than the error produced while actually testing the model. The cross-validation errors can be found on the individual .mlx files for each model which are linked in the Appendix.

It is also observed from the results that Model 2 and 4 which have an extra hidden layer and more activations as compared to the other two models do not present any extra advantage in terms of reduction of error. This may be due to the fact that increased number of layers and activations cause an increase in the number of θ values. An increased number of θ would add more constraints to the cost function minimization and may even increase the value of the cost function due to the regularization term making the optimization process complicated and inefficient.

In conclusion, several improvements can be made to the model to improve its performance and then the model can be tested with more and more complicated data sets. Firstly, in the instance of more than two inputs, the model would require some more of feature engineering to reduce the number of inputs and combine redundant inputs. Additionally, usage of more realistic data such as lab generated data or data from active fields would provide sufficient spread or variance preventing over-fitting of the model. Along with this, an increase in the number of training examples would also pose a benefit. A more efficient way to calculate λ would also ease the problem of over-fitting. In this instance reducing the number of folds in the K-Fold method might have prevented the issue. Further, more studies need to be done on improving the architecture of the model i.e. optimizing the other hyperparameters such as number of layers and activations.

For the examples of constant and variable GOR the models, especially, 1 and 2 function fairly well even with the limited number of training sets provided. With the addition of more examples and noise, these models can be used in fields over a certain period of production where the GOR and Water cut are within a predictable range. As the field life progresses, the models can be re-tuned to the logged data and be kept in use for the rest of the field life.

6 Future Work

In the vein of statistical VFM models, different machine learning techniques can be explored. A neural network has several hyperparameters that need to be optimized which can be quite taxing. Using different approaches such as SVMs might make this part of the process simpler. Furthermore, a hybrid VFM model can be explored to increase the accuracy of predictions by constraining them with physics-based correlations. This would focus the fine-tuning of parameters to the physical parameters of the model rather than strictly statistical parameters. It would also be easier to exercise physical intuition in the constraint of these parameters, increasing their reliability. In this study, flow rates are predicted using boundary pressures with the assumption that fluid properties such as GOR and Water cut are not monitored. However, multiphase flow rates are directly correlated to these fluid properties and prediction models can benefit from the knowledge of these correlations. As a practice, in active oil and gas fields, well fluids are regularly tested to estimate these properties not just for the current production but also to forecast future production. These forecasted values can also be included in the statistical or even hybrid models to bolster their ability to predict flow rates.

Bibliography

[1] Rasmussen Å. 'Field applications of model-based multiphase flow computing'. In: North sea flow measurement workshop (2004).

[2] Amin A. 'Evaluation of Commercially Available Virtual Flow Meters (VFMs)'. In: OTC Offshore Technology Conference (2015).

[3] Nikolai Andrianov. 'A Machine Learning Approach for Virtual Flow Metering and Forecasting'. In: *IFAC-PapersOnLine* 51.8 (2018). 3rd IFAC Workshop on Automatic Control in Offshore Oil and Gas Production OOGP 2018, pp. 191–196. ISSN: 2405-8963.

[4] Goodfellow I., Bengio Y. and Courville A. *Deep learning.* MIT press, 2016.

[5] Andreolli Ivanilto, Zortea Maciel and Baliño Jorge Luis. 'Modeling offshore steady flow field data using drift-flux and black-oil models'. In: *Journal of Petroleum Science and Engineering* 157 (2017), pp. 14–26.

[6] Lasater J.A. 'Bubble Point Pressure Correlation'. In: *Journal of Petroleum Technology* 10.05 (1958), pp. 65–67. DOI: 10.2118/957-G.

[7] Hornik Kurt, Maxwell Stinchcombe and Halbert White. 'Multilayer feedforward networks are universal approximators'. In: *Neural networks* 2(5) (1989), pp. 359–366.

[8] Bogaert P. M. et al. 'Improving oil production using smart fields technology in the sf30 satellite oil development offshore malaysia'. In: Offshore Technology Conference (2004).

[9] Hotvedt Mathilde et al. 'On gray-box modeling for virtual flow metering'. In: *Control Engineering Practice* 118 (2022), p. 104974.

[10] Shoeibi Omrani P. et al. 'Improving the accuracy of virtual flow metering and back-allocation through machine learning'. In: Abu Dhabi International Petroleum Exhibition Conference (2018).

[11] H. Poulisse et al. 'Continuous well production flow monitoring and surveillance'. In: Intelligent energy conference and exhibition (2006).

[12] Ramachandran Prajit, Zoph Barret and Le Quoc V. 'Searching for activation functions'. In: *arXiv preprint arXiv:1710.05941* (2017).

[13] Corneliussen S. et al. *Handbook of Multiphase Flow Metering.* Norwegian Society for Oil and Gas Measurement, 2005.

[14] Apiwat Sangphukieo, Teeraphan Laomettachit and Marasri Ruengjitchatchawalya. 'Demonstration of nested 5x3 fold-cross validation.' In: (2021).

[15] Schlumberger. *OLGA 2015: User Manual.* 2015.

[16] Bikmukhametov Timur and Jäschke Johannes. 'First principles and machine learning Virtual Flow Metering: A literature review'. In: *Journal of Petroleum Science and Engineering* 184 (2020), p. 106487.

Appendix

A GitHub Source

https://mgha2511.github.io/Megha-Rajasekhar/

The GitHub repository linked above contains the following:

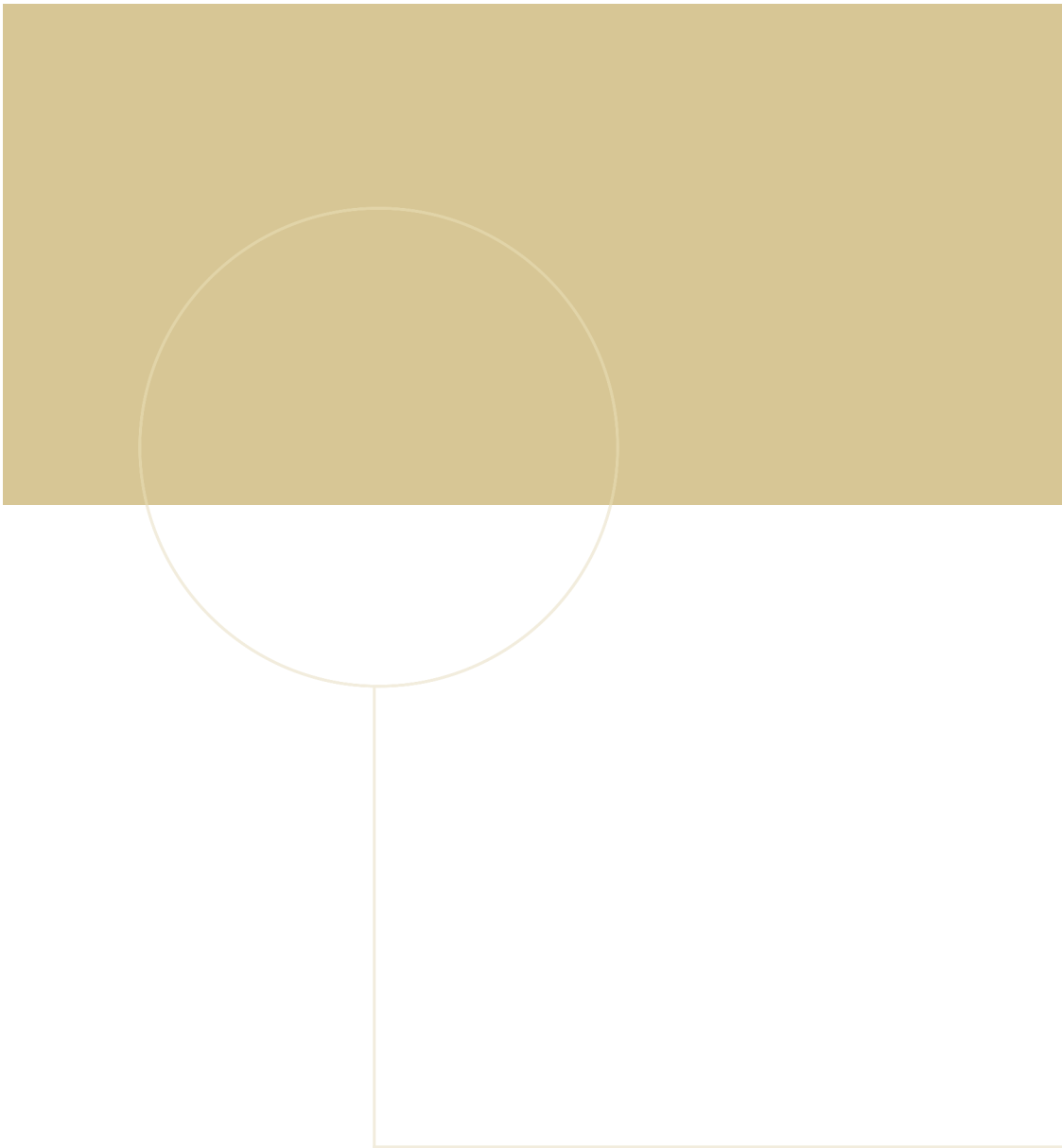
- .mlx files for each model type along with the MATLAB functions required to run the main file
- Python script for generating .inp files from a Template
- All the template .inp files used for data generation
- An example .bat file that is used to sequence the .inp files and run large batches of simulations
- An example .ppl and .tpl file for information on how resutls are output
- Python script for extracting specific results from a .ppl or a .tpl file which can be modified to extract any type of results.
- A text guide explaining the function of each python file, OLGA file and also a guide to the MATLAB model folders.

B Breakdown of Data Sets

Data Set	**Training**	**Test**
Constant GOR	180	14
Variable GOR	190	10
Variable GOR & WC	380	20
Outlet Pressure Variable	480	20
Variable GOR, WC & Pressure	480	20

s

Table 4: Number of Training Examples



NTNU

Norwegian University of
Science and Technology