ACSE-9: MSc Individual Research Project 2019/2020

# Deep Learning in Virtual Flow Metering

**Nicolas Trinephi**

**CID: 01212102**

nicolas.trinephi@imperial.ac.uk

GitHub: acse-nt719

Repository link: https://dev.azure.com/wintershalldea/Data%20Science/_git/IC-VFM

Supervised by:

**Eva Avbelj** eva.avbelj@wintershalldea.com

**Meindert Dillen** meindert.dillen@wintershalldea.com

Imperial College London
Exhibition Rd
London SW7 2BU
United Kingdom

Wintershall Norge AS
Jåttåflaten 27
4020 Stavanger
Norway

## Acknowledgements

I am not an early bird, but during this project, I was eager to wake up early every morning for the small time overlap I had with my European supervisor'. The project was filled with new libraries to learn and obstacles to overcome with something new at every corner.

I would like to thank Eva Avbelj for giving me the opportunity to work with her on a project as full of surprises as this one. As my supervisor, she shared her ideas and thoughts on the investigation and spent extra time with me to accommodate for time differences.  A great thank you to my second supervisor Meindert Dillen who helped me get accustomed to PySpark with a sprinkle of Scala. He was there to answer any programming questions I had and offered advice for solving problems, many problems. I was new to data science but thanks to Vitaly Elichev and explanations, I now understand standard data science procedures and what to look out for.

During these dark times, I would like to thank my family, my friends and my loving girlfriend for keeping me company and maintaining the strong moral. Without them, it would have been more difficult to find the courage to keep going.

## Abstract

In oil and gas production, an oil reservoir is typically extracted with many production wells with production licenses shared by different companies. Economic conflict may arise when different players use the same storage tanks. Currently, proper back allocation relies heavily on surface and subsea sensor measurements obtained by instruments which develop bias as they age. These inaccuracies can be a source of error when considering well management and decision making for the reservoir's future.

Virtual Flow Metering (VFM) is a process whereby flow is calculated from the data measured by physical sensors. It is used to back allocate the production to each well and owner. Classical VFM techniques which rely on the physical properties of the collected oil, have been extensively used in industry and studied in the literature for over half a century. Recently, data driven VFM techniques have appeared and are gaining popularity. They do not require oil property expertise and are also less computationally expensive: the efforts can be focused on the computer algorithm.

The aim of this study is to investigate the performance of such data driven VFM techniques namely the long short-term memory (LSTM) algorithm. The experiments are run both on a local machine as well as Databricks Azure cloud service. LSTM networks have been studied before in VFM literature however this paper compares many different hyperparameters as well as networks such as bidirectional LSTM and stacked LSTM. It is found these latter more sophisticated networks generally perform better than the classic ones, but surprisingly not by much. The networks using different hyperparameters are compared using the Keras library in the Python programming language. Data was collected from a North Sea field and provided by Wintershall DEA Norway. Data from different wells are also used with varying frequency and quantity.

Contents

# Chapter 1

## 1.1 Introduction

Petroleum products are paramount to many industries and its production, transportation and refining make up its very prosperous and dynamic industry. Oil and gas are the main resources produced at an oil reservoir and are used for energy through a variety of fuels and the manufacturing of other products like plastics and lubricants. In oil and gas production, it is very important to keep track of the quantity of hydrocarbons extracted in order to allocate resources to each operating company: back allocation. This has been achieved in several ways throughout the industry's history starting with phase separators installed on site. Recently, methods using data and computer networks have emerged which are mainly used to calibrate the physical sensors.

With the collection of large amounts of data, data driven computer algorithms have seen an increase in prosperity in recent years, causing a surge in virtual flow metering research. This paper proposes the use of a Long Short-Term Memory (LSTM) deep learning model to predict gas and oil flow rate based on past pressure and temperature data. Bilateral LSTM and stacked LSTM are not robustly studied in the literature; this study proposes a start on those two methods with comparison to classic LSTM which, while still novel in VFM, have been studied before. The data will be provided by Wintershall DEA and taken from an offshore satellite oil field. The available dataset contains figures on different wells and many different parameters. However, pressure, temperature, oil flow rate, gas flow rate and choke size will be the principle focus as these have been demonstrated to be correlated (Toskey, 2011).

The main field was discovered in the '80s with development and operation approved a decade later. Production started in the '90s with the construction of an offshore floating platform and 24.5 million NOK invested over the years. There have been a dozen wells drilled at the field with the primary resource retrieved being oil pre-2007 with gas exportation starting in 2007. Multiple satellite fields exist but only one is currently tied back since 2016, the data used in this paper comes from this field. The field is still dynamic and there are plans for over a dozen new wells to be drilled and multiple new satellite fields to be tied back to the platform in the future. Oil is collected via gas lift from the subsea oil centers and loaded onto storage tankers to be transported to shore.

## 1.2 Aims and Objectives

This project aims to process data from the offshore oil field and satellite field and develop a deep learning model which will learn from the data and predict flow rates. The main points of focus are:

- Data loading, analysis and filtering prior to insertion in the deep learning algorithm.
- Trends learned by the algorithm and experimentation with hyperparameters to find the best model for the application.

This can be directly applied in the future to back allocate the tie-back wells, new and old. It is hoped that the model can accurately predict rates given past data.

## 1.3 Literature review

Every human has used an oil product and yet oil production, despite being a large industry, is often over simplified and poorly understood by the average consumer. Once an oil reservoir has been found, wells can be drilled in order to extract hydrocarbon filled fluid marking the beginning of oil production. This is only possible after obtaining licenses which can take several years. These wells are flowrate controlled by choke valves at the wellhead which then allow extracted fluid to flow through the flow line. The extracted contents are a multiphase conglomerate of a variety of fluids, gases and solids (Falcone et al., 2009). Many different treatments are then conducted on the fluid such as impurity removal (sand separators) or heat treatments. In different cases, the extracted fluid is changed before collection via injection of various additives into the reservoir such as biocides if the fluid is prone to bacteria or corrosion inhibitors to preserve equipment (Fink 2015). The last step consists of separating the fluid into its useful constituents: usually oil, gas and water with a multiphase separator before being sent offsite for further processing. In many petroleum production fields, there are many wells in order to maximize fluid extraction in the reservoir.

However, there may be many different companies using the same wells. Thus, determining the amount of oil that each company owns becomes an important task called product allocation. Where inaccuracies occur, imbalances in monetary flow follow which will lead to economic disputes between the oil parties. Many methods can be used in order to provide an accurate allocation including sophisticated machinery or computer algorithms.

### 1.3.1 Well Testing

Well testing is the most basic method for back allocation. It involves running the output of a well into a test separator, the main well test piece of equipment *(Figure 1)*. The flow rates are measured with single phase flow meters at the outlets of the separator. Well testing is very accurate with up to 1% uncertainty (Dahl et al. 2005) but the opportunity cost is high: the test separator needs time to adjust and stabilize. The well must be regulated which can take several days and sometimes the well stream must be split, lowering production during this time, in addition, the equipment occupies a lot of space on an offshore platform. However, it is still a primordial step in oil and gas production, providing the most accurate data allowing thorough understanding of the well through which best strategies can be adopted. Outside of flow metering, well testing is also used to identify mechanical issues and damage (Frantzen 2003).

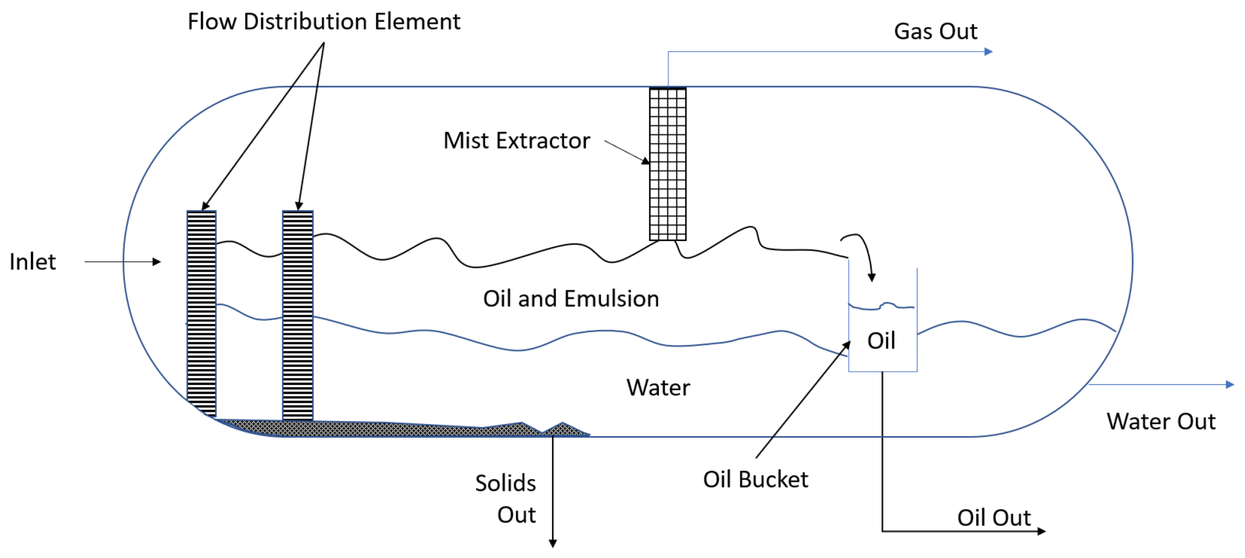


*Figure 1: Schema of Test Separator*

### 1.3.2 Multiphase Flow Metering

Multiphase Flow Metering (MPFM) is described as a technique using different gauges to track the flowrates of mixed fluids that have sand, gas and oil hence multiphase. This technique is thoroughly described in the Multiphase Metering Handbook by the Norwegian Society for Oil and Gas Measurement (Dahl et al 2015). It requires the installation of sensors at the well head or anywhere more downstream. This technique has not been used in the studied sea field but is being considered in the future for a variety of satellite projects. If used in conjunction with well testing, the producer can use both to increase testing capacity or use the well test separator for production and only test with it to calibrate the MPFM.

Any instrument, with the passing of time, can become biased and inaccurate, requiring regular recalibration. The adjustment procedures generally necessitate a representative rate to be determined at test facilities. In some subsea production architectures, verification and recalibration must be done on site. Hence, the installation of meters as well as their maintenance can be a risky and expensive task, and in subsea fields an expedition would be necessary (Berg et al, 2015).

### 1.3.3 Virtual Flow Metering

Virtual Flow Metering (VFM) makes use of the outputs of basic, readily available sensors mainly pressure and temperature in order to predict the state of the fluid in question. The flow rate of gas and oil have a mathematical relation to pressure and temperature therefore many models have been created in order to predict one from the other. VFM methods do not require extensive campaigns that MPFM methods do but are often used in conjunction with them in order to self-improve. There are two main approaches in VFM: the first principles VFM and the data-driven VFM (Bikmukhametov and Jäschke 2020), the main focus here will be on the data-driven variety.

The hydrodynamic or first principles VFM as described by Bikmukhametov, is based on matching a mechanical model to the data iteratively with an optimization algorithm. This model is a combination of fluid properties models with production models corresponding to the producer's choice. All the parameters in the model are adjusted in order to minimize the error against the data. This becomes extremely computationally expensive and time consuming. However, as it has been studied for more than half a century, it is very reliable and is the most commonly used VFM technique.

The data-driven modelling method involves using modern machine learning techniques in order to find patterns in the data and provide predictions accordingly. The mathematical analysis and physical understanding are not necessary using this technique since the algorithm will learn from the data, assuming the data is reliable. The data is required to be cleaned, processed and for the most accurate results, must be in abundance. The machine learning algorithm essentially fits the data with a complex mathematical function as closely as possible. Therefore, it is possible for the model to predict rates at future times contrary to the hydrodynamic method which can only forecast the next time step (Adrianov 2018). The two main approaches are forward neural networks (FNN) and recurrent neural networks (RNN). The former being the more broadly used technique, requiring weights to be calibrated in order to reduce error on the output. The latter is more advanced and are more suited for predictive applications. FNN have been demonstrated to provide accurate results in steady state cases; otherwise RNN are more accurate (Omrani et al. 2018). Steady state is rare in practice compared to transient cases where RNN models retain accuracy.

### 1.3.4 Data Driven Techniques:

Data is becoming more abundant and available: all industries are now collecting data which can be used to improve decision making and replace old practices. Data driven techniques use this new resource and are constantly being developed and improved. We distinguish two families of algorithms: machine learning and deep learning. Machine learning requires the features, defining characteristics of a dataset, to be explicitly presented to the algorithm through feature engineering, an extensive process involving multiple types of feature manipulation techniques: extraction, construction and manipulation (Liu et al., 1998). Deep learning networks automate this process performing feature engineering itself. The focus here will be on recurrent deep learning algorithms.

*Feedforward Neural Network*

Classical Feedforward Neural Networks (FNN), the most basic deep learning algorithm, are very well known. A basic configuration is illustrated in *Figure 2*. The goal is to produce a function $f$ which will be able to differentiate the characteristics of the dataset being used for training. The function maps *fixed* inputs $x$ to categories $y$ with $f(x;\theta) = y$ and $\theta$ are weights (Gupta T. 2017). Biases $b$ are used as a negative threshold (MacLennan 2017). Depending on the complexity of the problem hidden layers and their number of neurons can be varied (Omrani et al. 2018).

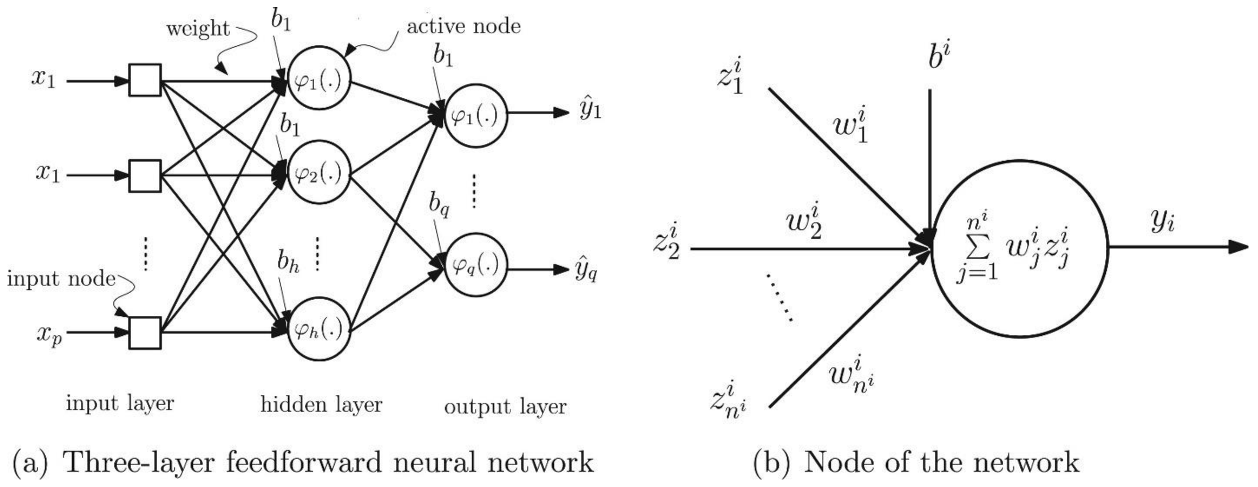


*Figure 2: a) Three-layer feedforward neural network, b) neuron of a neural network. "the neurons in a layer have connections (weights) from the neurons at its previous layer" (Ojha et al. 2017) Each neuron has a φ function calculated from the weights w and biases b, these define the whole algorithm and allow the network to be trained and function correctly.*

Each neuron has a *weighted input* $z$ and for the $i$th neuron in the $l$th layer we have

$$z_i^l = \sigma(\sum_{j=1}^{n} w_{ij}^l z_j^{l-1} + b_i^l), \qquad [1]$$

with $w_{ij}^l$ the weight to the $i$th neuron from $j$th neuron of the previous layer, $z_j^{l-1}$ the activation functions of the previous layer, the bias of the current layer $b_i^l$ and $\sigma$ is the activation function (MacLennan 2017). As such, these networks only use the current state of the input to produce an output; there are no feedback loops and are unable to recognize temporal patterns. Many researchers have found limited forecasting success in VFM with FNN (Al-Qutami et al. 2018) due to the aforementioned limitations.

### *Recurrent Neural Network*

In contrast, recurrent neural networks (RNN) operate on an input space as well as an internal state space, they retain information from the previous time steps. This allows RNNs to adapt to patterns in sequential data with great success in modelling variable length sequences (Pascanu et al., 2013) such as language modeling (Graves *et al.*, 2013). In VFM, they capture the dependency between the data and the past states of the data, the time dependency.

RNN neurons share parameters allowing retention of information from the previous time steps: patterns are not relearned with every encounter. Consider an FNN with one neuron in each hidden layer but with an input into each hidden layer corresponding to this sequential data. Since each hidden layer neuron has its own weight and bias, the layers behave differently and cannot merge. Instead, if the weights and biases are equal then these can be merged into one neuron: the RNN neuron, see *Figure 3* (Gupta 2017). The 'inner layers' are time steps with $t$ the current time step, $t-1$ and $t+1$ the previous future time steps respectively.

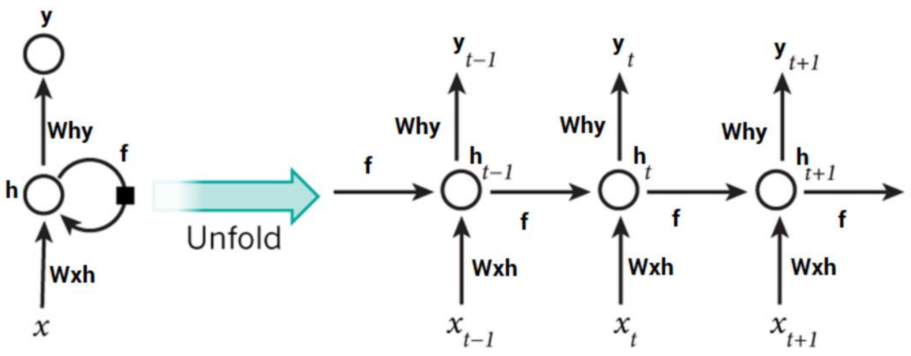


*Figure 3: RNN neuron to unfolded. The weights $W_{hy}$ and $W_{xh}$ are in matrix form as the data is sequential and are the same for each time step in a single neuron. (Gupta D. 2017)*

As a result, the current state of the neuron can be expressed as

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t), \qquad [2]$$

with σ the activation function, commonly the $tanh$ function, $W_{hh}$ the hidden layer weight matrix, $W_{xh}$ the input layer weight matrix, $h_{t-1}$ the state at the previous time step and $x_t$ the current input. The output at each state $y_t$ is of the form

$$y_t = W_{hy}h_t, \qquad [3]$$

with $W_{hy}$ the weight matrix of the output later.

RNN are limited in retaining patterns that are long term in data (Omrani 2018). This down fall is tackled by integrating a special memory cell into the network.

### LSTM

Long-short term memory (LSTM) networks take RNN a step further and are capable of capturing both long-term and short-term memory. The algorithm enforces constant error flow through its units (Hochreiter et al. 1997) called constant error carousels (CEC). The resulting architecture, *memory cells,* is generally comprised of four parts: a central recursive linear unit (the CEC), an *output gate* and input gate which control reading and writing data and a forget gate which allows memory to be reset, see Figure 4.

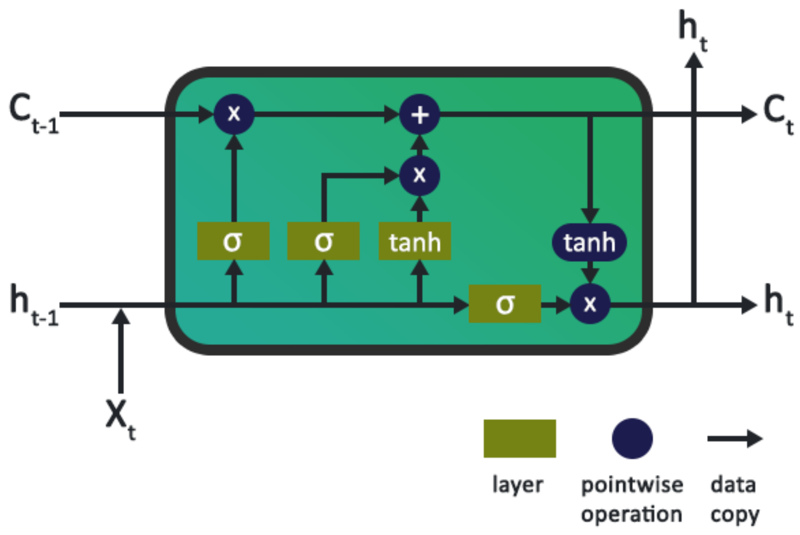


*Figure 4: LSTM network schematic, with $x_t$ the input gate, $h_t$ the output gate, A cells with inner steps for storage, forgeting and updating (Aungiers 2018). The equations which can be inferred are shown below.*

If we consider an input $x = (x_1, \dots, x_T)$, the LSTM will map it to $y = (y_1, \dots, y_T)$ by using the following equations:

$$i_t = \sigma(W_{ix}x_t + W_{im}h_{t-1} + W_{ic}c_{t-1} + b_i) \quad [4]$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}h_{t-1} + W_{fc}c_{t-1} + b_f) \quad [5]$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \mathrm{g}(\mathrm{W}_{cs}x_t + W_{cm}h_{t-1} + b_c) \quad [6]$$

$$o_t = \sigma(W_{ox}x_t + W_{om}h_{t-1} + W_{oc}c_t + b_o) \quad [7]$$

$$m_t = o_t \odot m(c_t) \quad [8]$$

$$y_t = \varphi(W_{ym}h_t + b_y) \quad [9]$$

with $\varphi$ the output activation function, $\sigma$ the sigmoid function, $i, f, o$ and $c$ the input, forget, output gates and CEC respectively, $g$ and $m$ input and output activation functions, generally $tanh$ (Sak et al. 2014). More information can be found in Sak or Hochreiter.

Bidirectional LSTM (bi-LSTM) is a method which trains two separate LSTM networks on a forward and backward version of the sequence (Graves 2005). Both LSTM are fed into the same output layer with great effectiveness (Ma et al 2016).

In VFM, LSTM is a new approach which started more actively being studied in 2018 by Adrianov. However, using other methods such as bi-LSTM could prove to be viable in reducing errors, there have been great results in natural language inferencing (Liu et al 2016).

## Set up and training

Before training, the data must be normalized or rescaled to reduce errors and time consumption. Differences in input scale can cause the appearance of large weights, making the model unstable (Bishop 1995).

When training any neural network, the weights are usually randomly set and the algorithm passes the inputs and obtains outputs. The most common technique for weight adjustment is back-propagation (BP) followed by an optimization function where the weights are updated starting from the output going 'backwards' toward the input. Weights are updated in order for the given input, to give the required output. This involves calculating partial derivatives of the cost function with respect to any weight $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ in the network in order to minimize it.

A variety of cost functions exist but the most common one is the quadratic cost function of the form

$$C = \frac{1}{2n} \sum_x \left| |y(x) - z^L(x)| \right|^2, \qquad [10]$$

where $n$ is the size of the training data set, $x$ is the individual training samples, $y$ is the desired output, $L$ is the number of layers used and $z^L(x)$ is the action vector of outputs for input $x$ (Nielsen 2019). Cost and loss functions refer to almost the same thing with the cost function being an average of the losses, hence the division by $2n$. The loss is calculated at every neuron while the cost is calculated at each epoch. At each iteration of the neural network or epoch, the BP calculates the derivatives and an optimization function seeks to use the derivatives to minimize the cost function by incrementing the weights of each neuron by a small amount.

Since the weights are the same for all of the layers, the gradients of the cost function are simply added together and the weights optimized for the RNN neuron and other layers by the optimizer.

Optimization functions, such as Adam, are used by all networks and require some parameters, like learning rate, to be set by the user. Any parameters not learned by the algorithm are called hyperparameters and must be chosen correctly, through grid search or other method. For the effect of learning rate on loss, an illustration is shown in *Figure 5*.

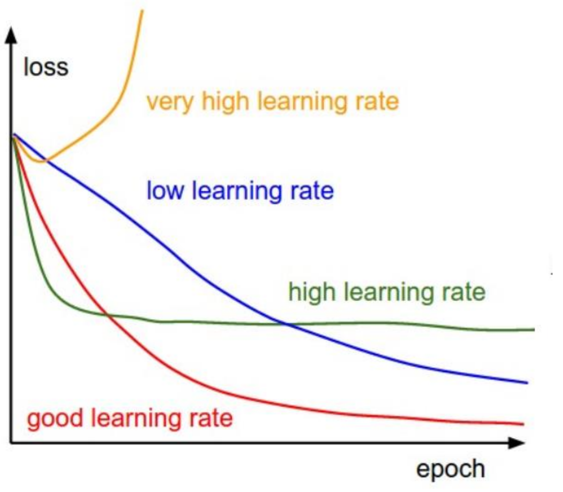


*Figure 5: loss against epoch, loss evolution with different learning rates (CS231n) If it is too low, the algorithms learn too slowly, if it is too high, the algorithm diverges.*

All networks face weight gradient-based problems. The exploding gradient problem describes the exponential rise of the gradients layer by layer when completing BP (Philipp et al. 2018). The vanishing gradient problem occurs when the gradient of the loss function is very small (Hanin 2018). Both make the learning too large or too small respectively rendering the model impractical. Some ways to reduce this problem include:

- explicitly controlling the gradients: ResNets (He et al 2016)
- precisely initializing the weights
- choosing specific non-linearities that produce stable gradients (Klambauer et al. 2017)
- LSTM CEC (Hochreiter et al 1997)
- or generally not have large amounts of hidden layers nor neurons.

# Chapter 2: Experimental Methods

## 2.1 Metadata (Code Requirements)

| OS | WINDOWS, REQUIRES DATABRICKS |
|---|---|
| **LANGUAGES** | Python 3 |
| **OPEN SOURCE LIBRARIES** | pandas, numpy, hvplot, holoviews, panel, mlflow, keras==2.2.5, sklearn, scipy, tensorflow, plotly |
| **ENTERPRISE LIBRARIES** | PySpark Apache, Cognite, in-built mlflow |
| **DISCLOSURE** | Data is undisclosed |
| **CURRENT CODE VERSION** | Most recent |
| **REPOSITORY LINK** | https://dev.azure.com/wintershalldea/Data%20Science/_git/IC-VFM |
| **INSTALLATION** | pip install vfm_tool |

*Table 1: Code Requirements*

The above requirements have been installed on the Databricks clusters. Otherwise, use requirements.txt with the following code in a cluster notebook:

```
!pip install -r requirements.txt
```

For extra information, a README.md is provided.

## 2.2 Software Description

The code was developed from scratch on Azure Databricks, a "collaborative Apache Spark™ based analytics service", and is stored on Wintershall DEA DevOps. Two functions were borrowed and slightly adapted from a colleague, clearly labeled in the code. The Databricks notebook (DN) acts as a tutorial file to the package files containing classes and methods. The methods run on DN unless specified, some plotting and Excel methods only run locally. The package uses both pandas and spark since visualization required the use of a local machine with pandas iPython widgets.

Manual testing was completed on each element of code. Sections of methods are copy pasted apart from the natural environment for testing and debugging, these can be found in the vfm_tool_tests directory. In the current code, assertions must be passed for some methods to run properly. Custom error messages are raised for invalid method inputs or types.

The utils class is used for nomenclature (`utils.__init__()` is not defined) and contains functions that rewrite data files, convert data frame lists, processes data, etc. Data files can be very messy, the utils methods takes care of these issues.

### DN Chapter 1: Data Load and Visualization

There are daily, monthly and test historic data (HD) for different wells as well as high frequency data (HFD) for one well head and its three tails. Data is loaded through instances of `pandas_data` class or its inherited `spark_data` class with file input, one object per file. For HD, `make_df` and `df_lister` are methods of `pandas_data` overridden by `spark_data`, they create the `pandas_data.df` with pandas (used locally) and `pandas_data.list_df`, the list of df for each well, analogous for `spark_data` with PySpark using overriding of parent methods; utils also works for both objects. The `spark_data` data frames are changed to pandas for deep copies to be possible. For HFD, the `spark_data.high_freq()` class method is used to **bypass** `pandas_data.__init__().` The pandas_data HFD set up exists but only runs locally. The parameters useful to this application are pressures, temperatures and rates. These are plotted (see *Figure 6*) against time using `holoviews` for HD (`holo_plot()` only local) with a drop-down menu for well display and `plotly` for HFD(`hf_plot`), situated in Visualization.py.

The visualization requires restructuring of the data in order to integrate the drop-down menu. The test data does not have well tails, only the heads so in order to plot, these had to be added by finding the well tails of each well in `list_df`. The algorithm was verified manually.

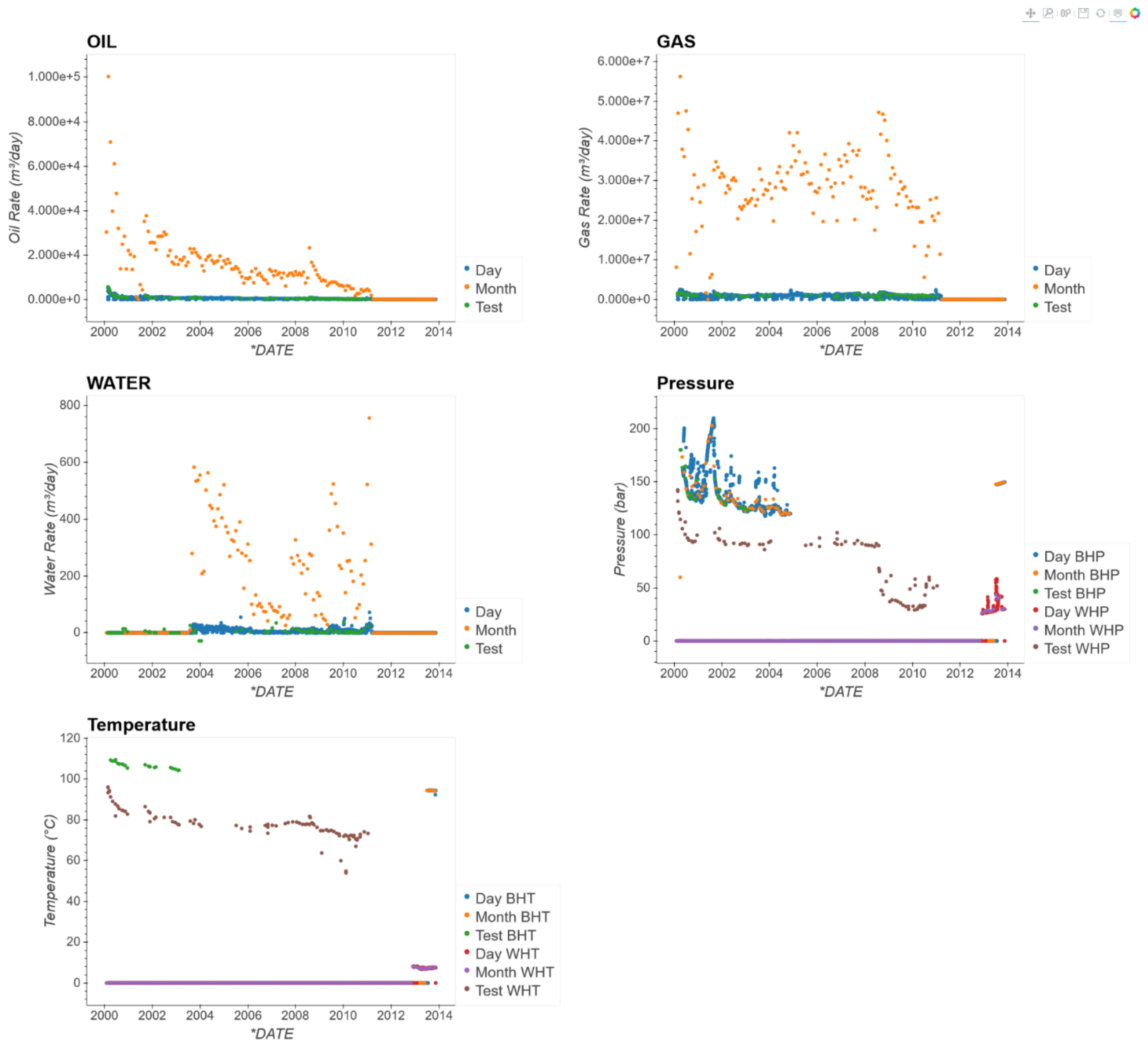


*Figure 6: Visualization of raw data from a well. The monthly rate values are extremely high and should be unaccumulated.*

## DN Chapter 2: Quality Control and Processing

Quality control involves checking the data consistency allowing us to decide which values to remove, methods are found in `q_control.py` imported globally. There are 3 steps:

1. Statistical analysis whereby ranges and percentage of values equal to 0 or `np.nan` are evaluated: `qc_data()`. Parameters that are always 0 or `np.nan` are unusable: downhole pressure (DHP) and downhole temperature (DHT) from HD. These parameters can be ignored.
2. Visually check the data for strange values. Historic monthly rates are cumulative (seen in figure 6) and require de-accumulation: `un_accum()`.
3. Scatter plot of rates against parameters to check for inconsistencies. Any points forming straight lines or outlying are turned into `np.nan`, see *Table 2*. Pressures and temperatures cannot be 0 while rates can due to shut-ins or other technical reasons.
4. The HD is noisy, sporadic and requires lots of filtering to be somewhat usable. `utils.process()` wraps the data cleaning functions together and filters the object data frames, the filtered visualization of the same well is illustrated in *Figure 7*. The HFD is clean and usable raw.

`Dynamic_filter` is an algorithm which smoothens a curve, it iterates through each point and calculates the mean of a chosen number (bin) of passed values and future values. The algorithm takes into account spikes and empty values by calculating the bin range and comparing it to a chosen limit. It also accommodates the `np.nan` values produced by processing.

| Parameter | Lower Limit | Upper Limit |
|---|---|---|
| Well head pressure (WHP) | 50 bar | 400 bar |
| Bottom hole pressure (BHP) | 20 bar | 400 bar |
| Well head Temperature (WHT) | 20 °C | 200 °C |
| Bottom hole Temperature (BHT) | 20 °C | 200 °C |

*Table 1: Table of limits found with the scatter plots from point 3 in DN Chapter 2.*

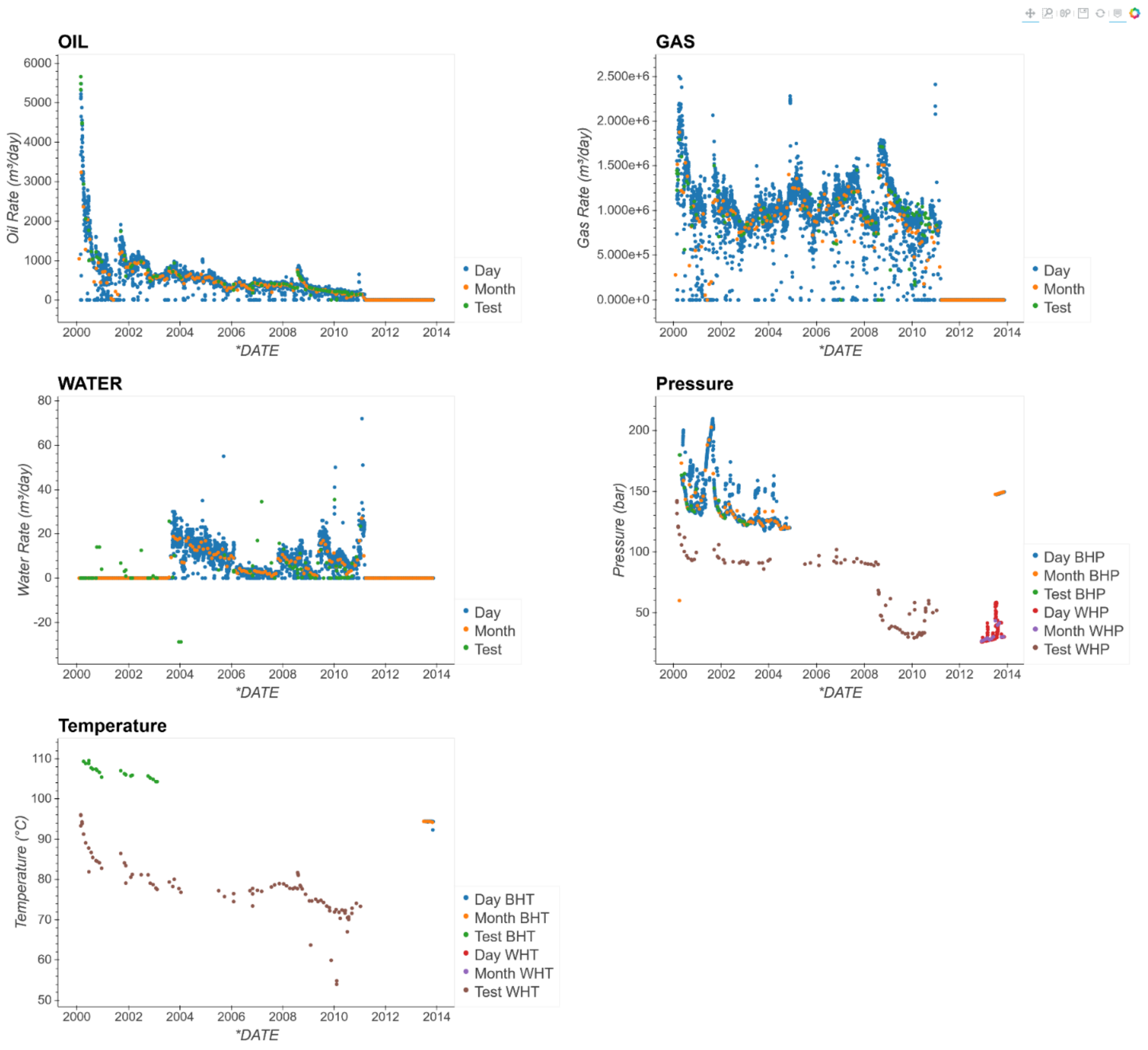


*Figure 7: Visualization of processed data, null and outlying values of pressure and temperature are removed and monthly rates are unaccumulated.*

## DN Chapter 3: LSTM

The `VFM_LSTM` class initialized with a prepared input `pd.dataframe` and a model name. It is **adaptive** to the number of labels in the input data frame: oil, gas and/or water rate. Its methods recognize which and how many labels are presented and will predict and plot the corresponding sequences. Using `VFM_LSTM` sets up an LSTM network with several steps detailed in *Figure 8*. Once the data is prepared and normalized, the user decide how many passed values are associated to each label value, the lookback. Once training is complete, the class is capable of plotting the loss as well as predicting an unseen test set. On Databricks, retaining model hdf5 files was not successful, meaning returning to a model after shut down of the cluster is impossible without retraining.

Using built in Databricks mlflow allows parameters and metrics to be logged easily into experiment files. These metrics allow models with different characteristics to be compared with each other. Three metrics are used in VFM_LSTM: root mean square error (RMSE), mean average error (MAE) and the r2 score (r2). RMSE and MAE allow the average error to be quantified with physical units, which in the HFD case are $m^3/day$. Our goal was to obtain predictions with an error smaller than $10m^3/day$. The r2 score is a general indication of the model performance: a score close to 1 is very good while a small score or negative score mean that the model prediction does not fit the actual test data. The models are accurate with r2 scores regularly above 0.990 so the main comparison metric was the RMSE.

A summary of the code workflow can be visualized with a flow chart shown in *Figure 9*.

**Dataset Preparation** Methods: `VFM_LSTM.split_data(), utils.prep_input()` (for historic), `utils.hf_prep_ input()` (for high frequency), `utils.rmv_exc()`

- Create data frame with columns of features on the left and the corresponding columns of labels on the right. Keras does not like `np.nan` values therefore these should be removed entirely.
- Initialize model object `VFM_LSTM(input, name)` with prepared dataframe and a desired name.
- Dataset is split according to the desired training split.

**Data Normalization** Method: `VFM_LSTM.normalize()`

- Rescale the data using `sklearn.preprocessing.StandardScaler()`, which transforms each column to have mean 0 and standard deviation 1. Important step to decrease training time and improve stability of the model.

**Supervised Learning** Methods: `VFM_LSTM.to_supervised(), VFM_LSTM.c_dataset()`

- Introduce lookback to the dataset whereby each label is associated to current features $x_t$ as well as past features $x_{t-1}$ and so on. The lookback refers to how many time steps we want to associate to the current label.
- Transform training and test set into the shape the keras LSTM standard input: `[samples, lookback, features]`. This process turns the problem into a supervised learning problem.

**LSTM Network** Method: `VFM_LSTM.train_model()`

- Input the hyperparameters into the training method and choose the desired LSTM algorithm style.
- `train_model` contains layers (LSTM layers, a drop out layer and dense output layer), adam optimizer and callbacks to retain the best model weights in name_wieghts.hdf5 and to stop training if there is no significant improvement in 35 epochs.
- Saving model weights for use later was not managed on Databricks.

**Plotting the Loss** Method: `VFM_LSTM.loss_plot()`

- We analyze how the validation performed against the training to check for overfitting or underfitting as discussed in Chapter 1.

**Prediction** Method: `VFM_LSTM.predict_y(), VFM_LSTM.eval_metrics()` (static method)

- Use the best weights name_weight.hdf5 file for the model to make a prediction on our unseen test set.
- Calculate metrics: sklearn library metrics modules are used to calculate root mean square error (RMSE), mean average error (MAE) and r2 score (r2), these are stored in a dictionary allows model performance to be compared.

**Plot Predictions** Method: `VFM_LSTM.predict_plot()`

- Visualize the performance of our model with time by plotting the prediction and actual against time. With the HD adaptive plotting and prediction is ensured, the model will predict any combination of the rates depending on which are present in the training set.

**Model Retention** Method: `VFM_LSTM.log_mlflow()`

- mlflow to record our parameters and metrics into a desired experiment file. mlflow proved incompatible with newer version of keras unless the source code is tweaked (this was completed locally where library source code is accessible, more in the documentation), hence the lower keras version.

*Figure 8: VFM_LSTM procedure, methods called, and description*

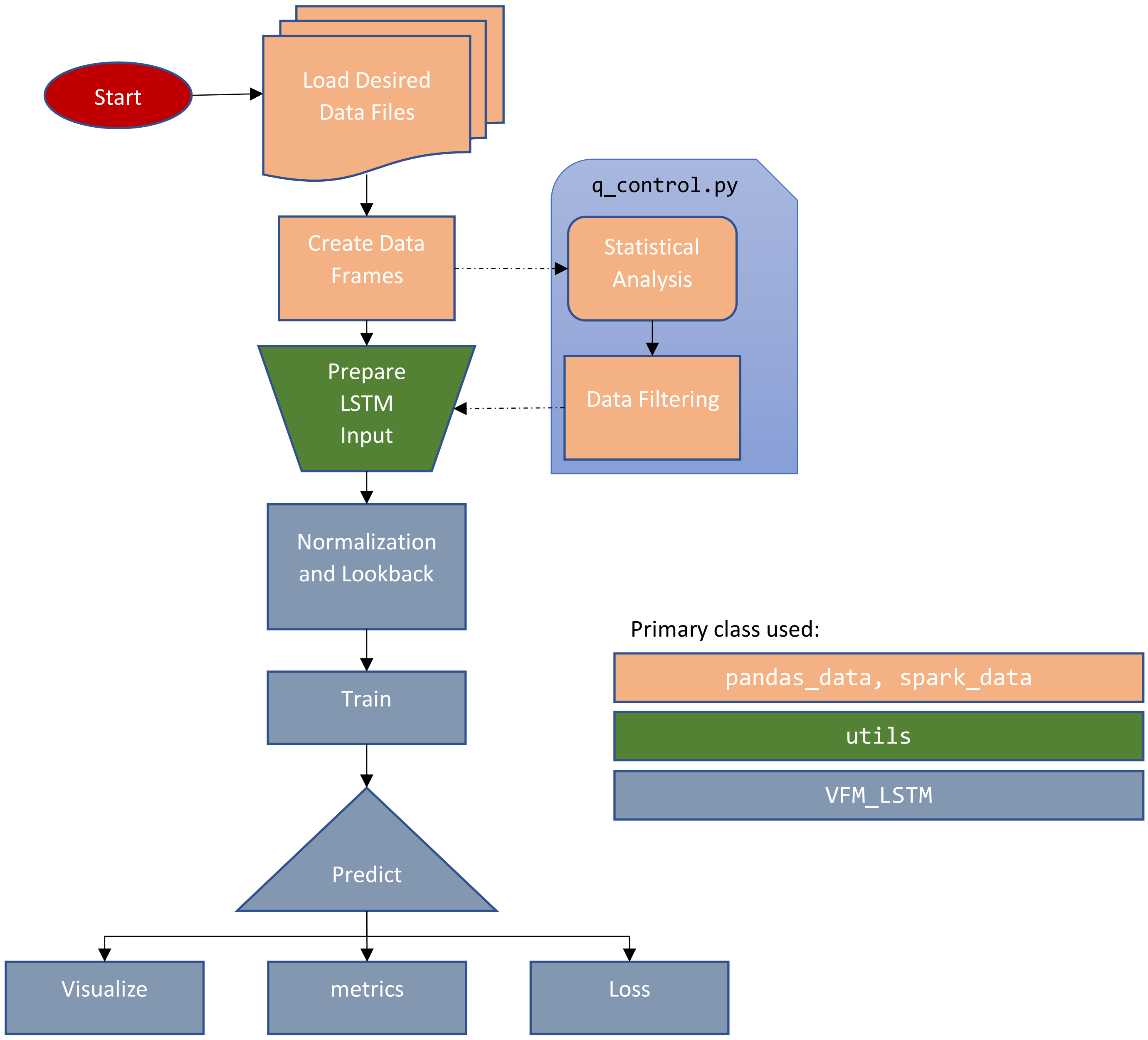


*Figure 9: Flow chart of our data driven VFM code*

# Chapter 3: Results and Discussion

All networks were trained with a batch size of 5000 or the entire dataset (the smallest of the two) over 100 epochs for HFD and 1000 epochs for HD or until early stoppage. Using the highest batch size that memory allows is advised to diminish the number of weight updates, reducing training time (Smith 2018), it was found to be around 5000. The classic LSTM and bidirectional LSTM (bi-LSTM) were the most studied, the other more complex models were quickly considered.

## 3.1. High Frequency Data

With the most samples available, HFD was the primary focus of the network. Classic and stacked LSTM behave similarly in that the latter points of the prediction stray away from the true values, see *Figure 12*, similar behavior was documented by Adrianov (2018) where predictions deviate slightly the further into the future they are made. The classic and stacked LSTM are not consistent meaning varying hyperparameters slightly shift performance greatly, see *Figure 10 and 11.*

The study investigated to compare how the hyperparameters affect performance. The drop rate serves to normalize the model and reduce overfitting. This is shown to be useful for models with low samples such as the HD. For the HFD, it does not have much affect and seems to improve or worsen performance randomly. LSTM units is another hyperparameter that would logically improve performance nonetheless the changes are very minimal past the 250 units mark, which is expected from the literature (Reimers et al. 2017). Due to the simplicity of the problem, increased units even degrade performance, see *Figure 11*.

Reducing LR however, produces very good results: the best accuracy calculated was a classic model with 128units, 1e-6 LR, 0.5 drop rate with 5.1$m^3/day$ RMSE error and >0.999 r2 score. That's 2$m^3/day$ RMSE better than the best bi-LSTM with 7.654$m^3/day$. Some tables with the best performance highlighted are accompanying networks sharing similar hyperparameters for comparison in *Table 3 and 4*.

As previously mentioned, learning rate changes the outcome greatly which is to be expected: very low learning rates yield the best classic LSTM models as seen in *Table 5*. The bi-LSTM is regularly below 15$m^3/day$ RMSE while the classic fluctuates. The classic LSTM with $10^{-9}$ learning rate is almost on par with bi-LSTM models. This is expected because of the simple problem where only one rate is being predicted. The efficiency difference should increase as more rates and more data are added to the problem. bi-LSTM varieties take roughly twice as long to train and produced only slightly better results than the single LSTM networks. However, due to the training time, classic LSTM may be the better choice on HFD. Of course, with more complex problems where multiple rates are predicted, the bi-LSTM is consistently high performing as expected from the literature.

Other complex variations were run: stacks, bi-LSTM stacked and bi-LSTM + classic. Apart from the stacked LSTM, the other two take 24hours or more to train; due to time contraints not as many were run. For the most part, they do not seem to offer any advantages to single bidirectional and are not worth the extra training time.

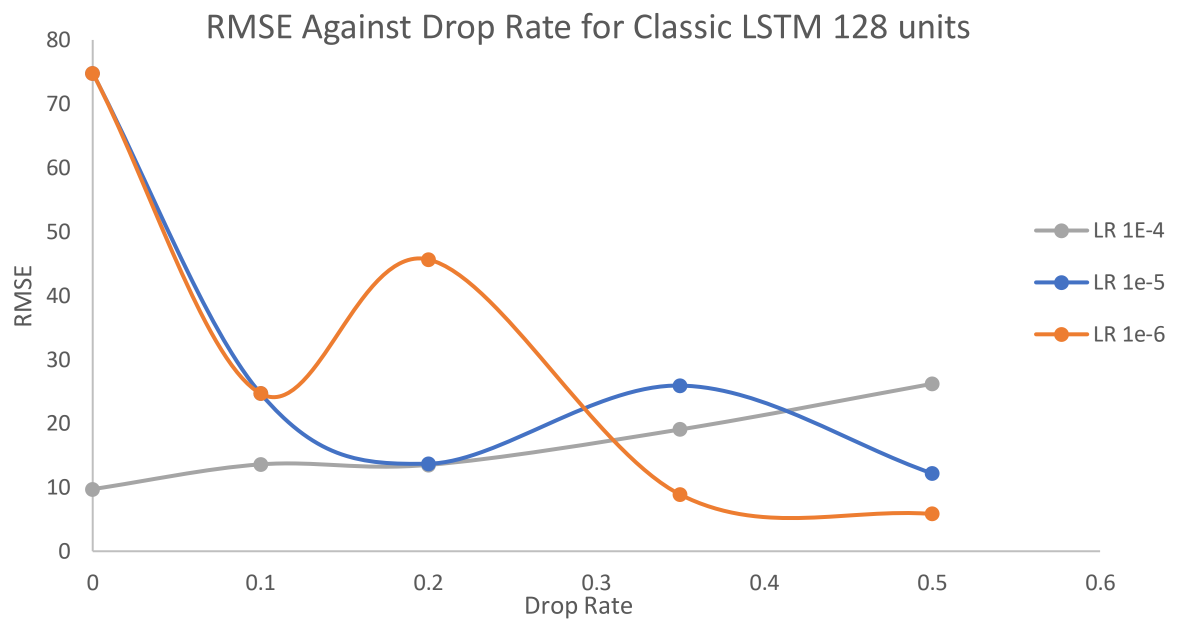


*Figure 10: RMSE against drop rate for 128-unit classic LSTM*

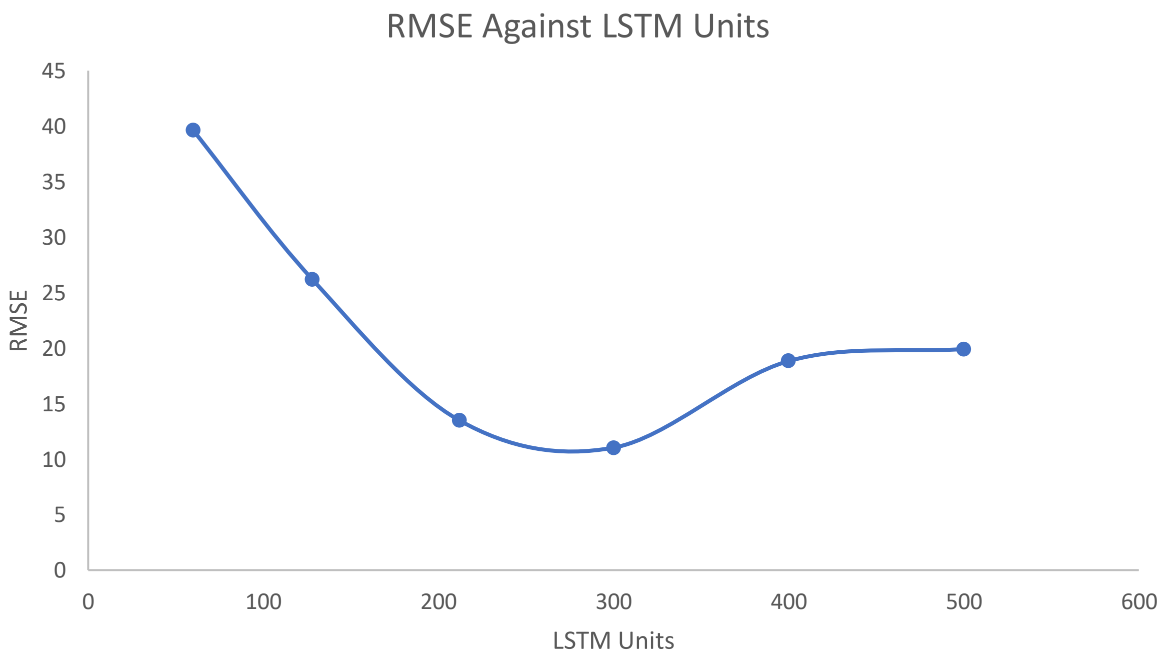
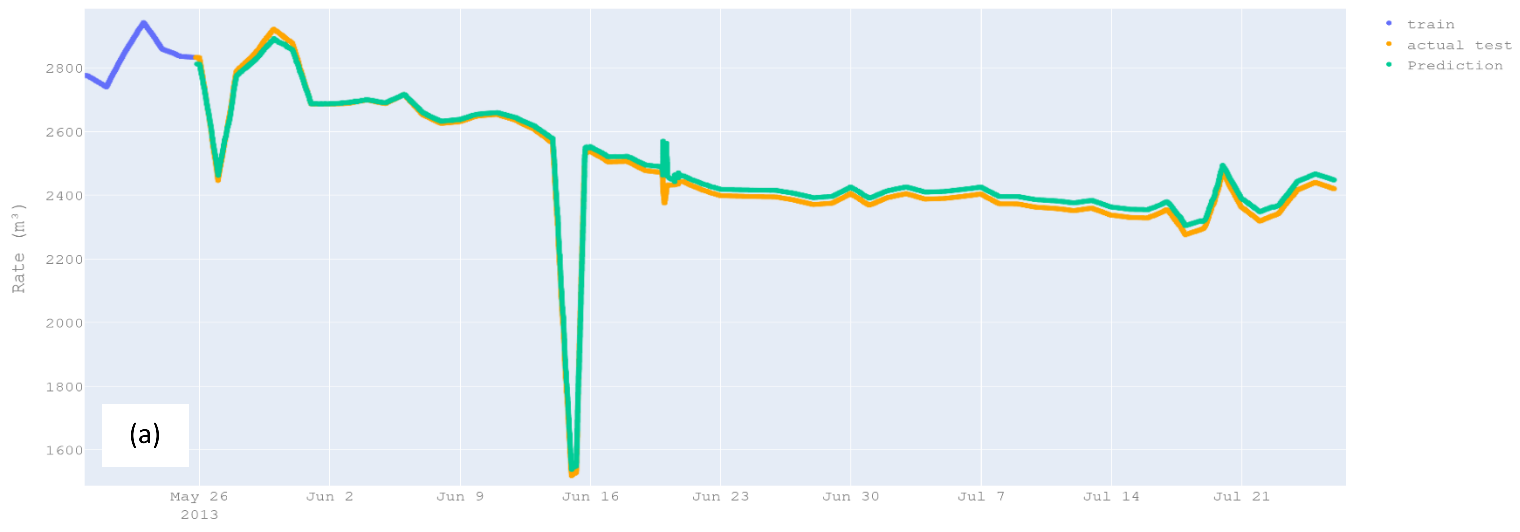


*Figure 11: RMSE Against LSTM Units. 0.5 drop rate, 1e-4 LR*



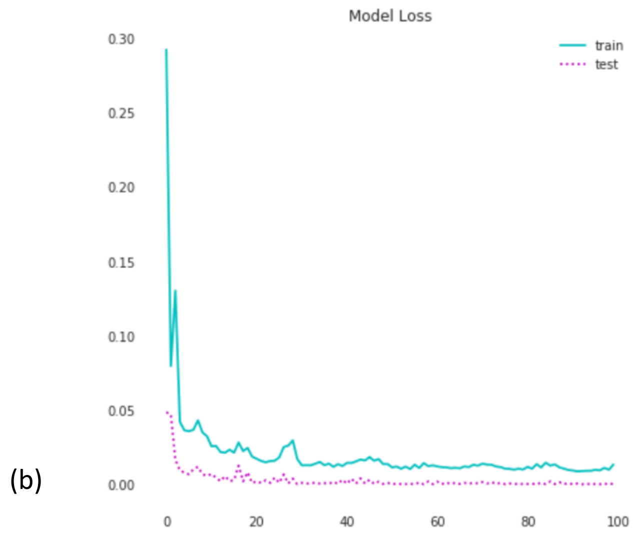
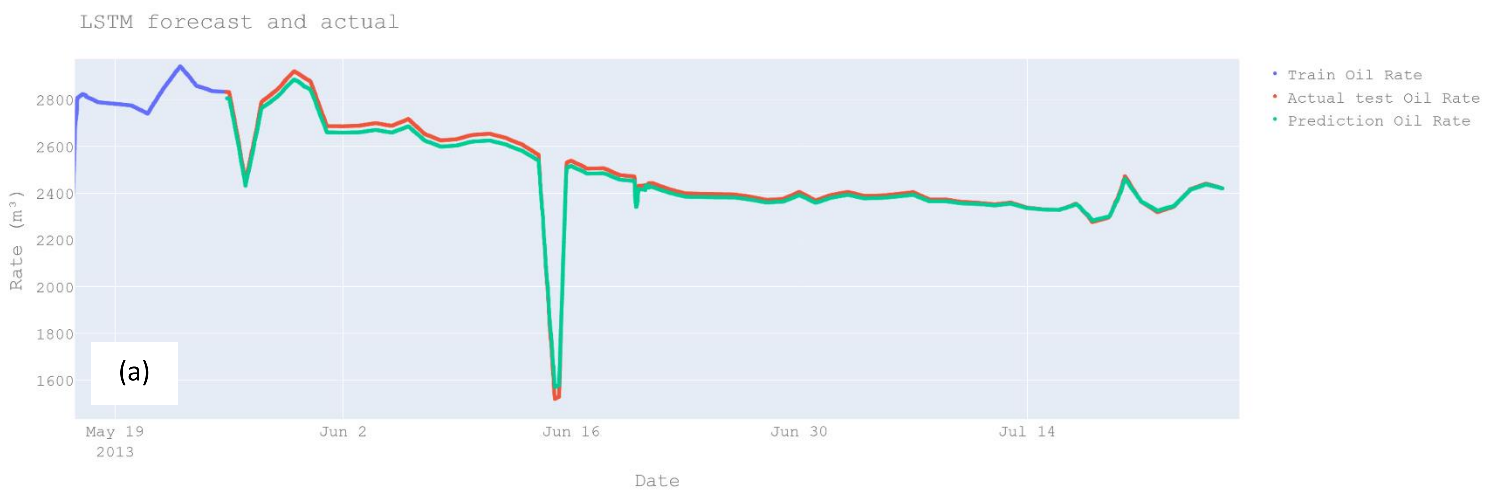


*Figure 12: Classic LSTM prediction (zoomed in on prediction) (a) and loss plot (b): 128 units, $10^{-4}$ LR, 0.5 drop rate. RMSE: 63.519*

| Network Type | units | LR | Time (h) | Drop rate | rmse | Mae | R2 |
|---|---|---|---|---|---|---|---|
| Classic | 300 | 1.E-04 | 3.5 | 0.5 | 11.032 | 9.06 | 0.998 |
| Classic | 128 | 1.E-04 | 2 | 0.5 | 26.2 | 19.289 | 0.991 |
| Bidirectional | 256 | 1.E-04 | 12 | 0.5 | 7.654 | 4.967 | 0.999 |
| Stacked | 256 | 1.E-04 | 13 | 0.5 | 22.192 | 18.134 | 0.994 |
| Bi-bi | 128 | 1.E-04 | 24 | 0.5 | 30.279 | 22.28 | 0.988 |

*Table 3: comparing best bi-LSTM to other networks of similar hyperparameters*



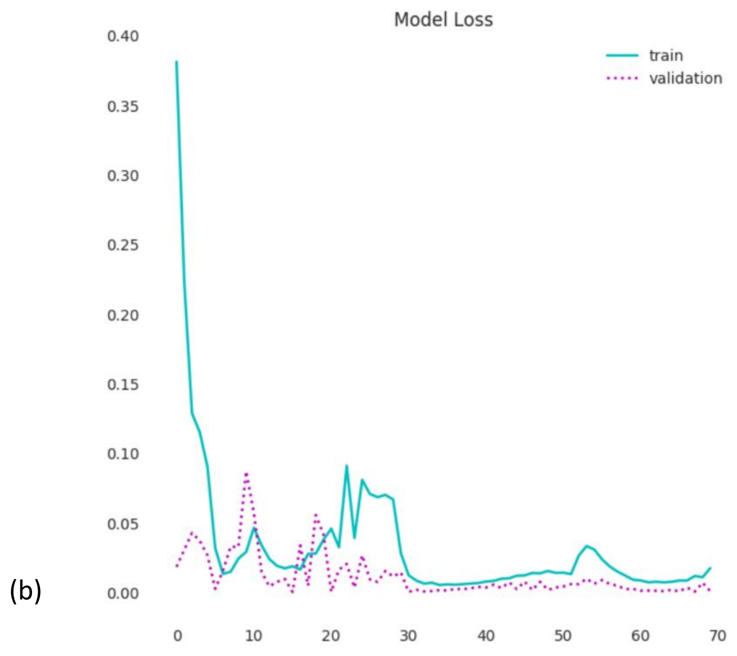
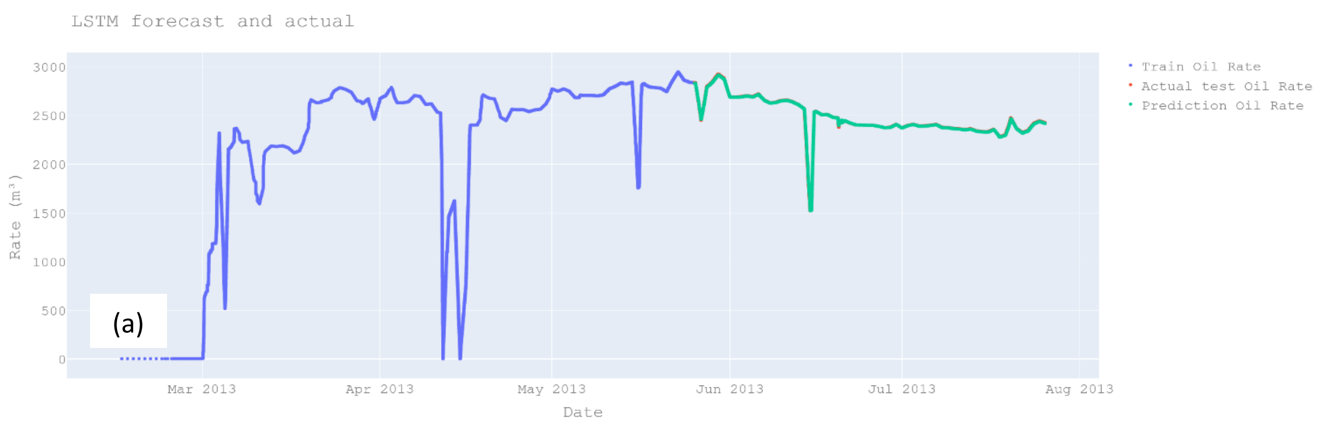


*Figure 13: Stacked LSTM prediction (a) (zoomed on prediction) and loss plot (b), 256 unit, $10^{-4}$ lr, 0.5 drop rate. RMSE: 22.2*

| Network type | Units | LR | Time(h) | Drop Rate | RMSE | MAE | R2 |
|---|---|---|---|---|---|---|---|
| bi-LSTM | 256 | 1.00E-06 | 12 | 0.5 | 7.654 | 4.967 | 0.999 |
| | 128 | 1.00E-06 | 4 | 0.5 | 8.794 | 6.93 | 0.999 |
| Classic | 256 | 1.00E-06 | 4 | 0.5 | 17.218 | 12.941 | 0.996 |
| | 128 | 1.00E-06 | 2 | 0.5 | 5.054 | 3.94 | 0.999 |

*Table 4: comparing best Classic LSTM to bi-LSTM*



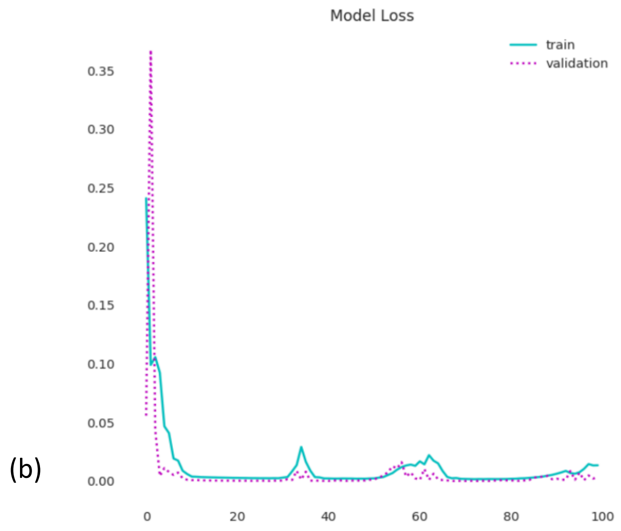


*Figure 14: Bidirectional prediction (a) and loss plot (b) for HFD: 256 units, 100 epochs, $10^{-9}$LR, 5000 batch size, 0.2 drop, 14hours, RMSE: 7.7$m^3$/day*

| Network Type | units | LR | Time(*h*) | Drop rate | rmse | Mae | R2 |
|---|---|---|---|---|---|---|---|
| Bi-LSTM | 256 | 1.E-09 | 11 | 0.2 | 7.712 | 5.842 | 0.999 |
| | 128 | | 4 | | 8.447 | 6.306 | 0.999 |
| Classic | 256 | | 4 | | 8.873 | 6.972 | 0.999 |
| | 128 | | 2 | 0.5 | 26.2 | 19.289 | 0.991 |
| | 128 | | 2 | 0.2 | 52.049 | 45.712 | 0.964 |
| Bi-LSTM | 256 | 1.E-08 | 11 | 0.2 | 13.016 | 10.582 | 0.997 |
| | 128 | | 4.3 | | 8.538 | 5.749 | 0.999 |
| Classic | 256 | | 7 | | 5.831 | 4.951 | 0.999 |
| | 128 | | 2 | | 13.511 | 11.434 | 0.998 |

*Table 5: Comparing very strong class LSTM to some bi-LSTM*

## 3.2. Historical Data

Regarding historic data, the model predicts for oil, gas and water rate. However, well data is mediocre. Only day data can have enough samples to produce decent results, over 1000 points, however much of the data is removed via processing or due to shut-ins. Hence, one well was chosen with a decent number of samples available for comparison between raw, processed and smoothened data. As we can see in *Table 5*, the classic LSTM outperforms the bi-LSTM slightly. This can be explained by the overfitting which, while present in both models, is much worse in the bi-LSTM due to the added network complexity (figure 9b and c).

| Network Type | units | LR | Time ($h$) | Drop rate | rmse | Mae | R2 |
|---|---|---|---|---|---|---|---|
| Bi-LSTM | 50 | $10^{-7}$ | 20min | 0.2 | 105.3 (oil)<br>215k (gas)<br>5.1 (water) | 83.51 (oil)<br>131k (gas)<br>3.3 (water) | -1.235 (oil)<br>0.3 (gas)<br>0.47 (water) |
| Classic | 50 | | 10min | | 93.73 (oil)<br>212k (gas)<br>5.1 (water) | 73.94 (oil)<br>126k (gas)<br>2.8 (water) | -0.771(oil)<br>0.32 (gas)<br>0.56 (water) |

*Table 6: comparison of bi-LSTM and classic with raw historic data.*

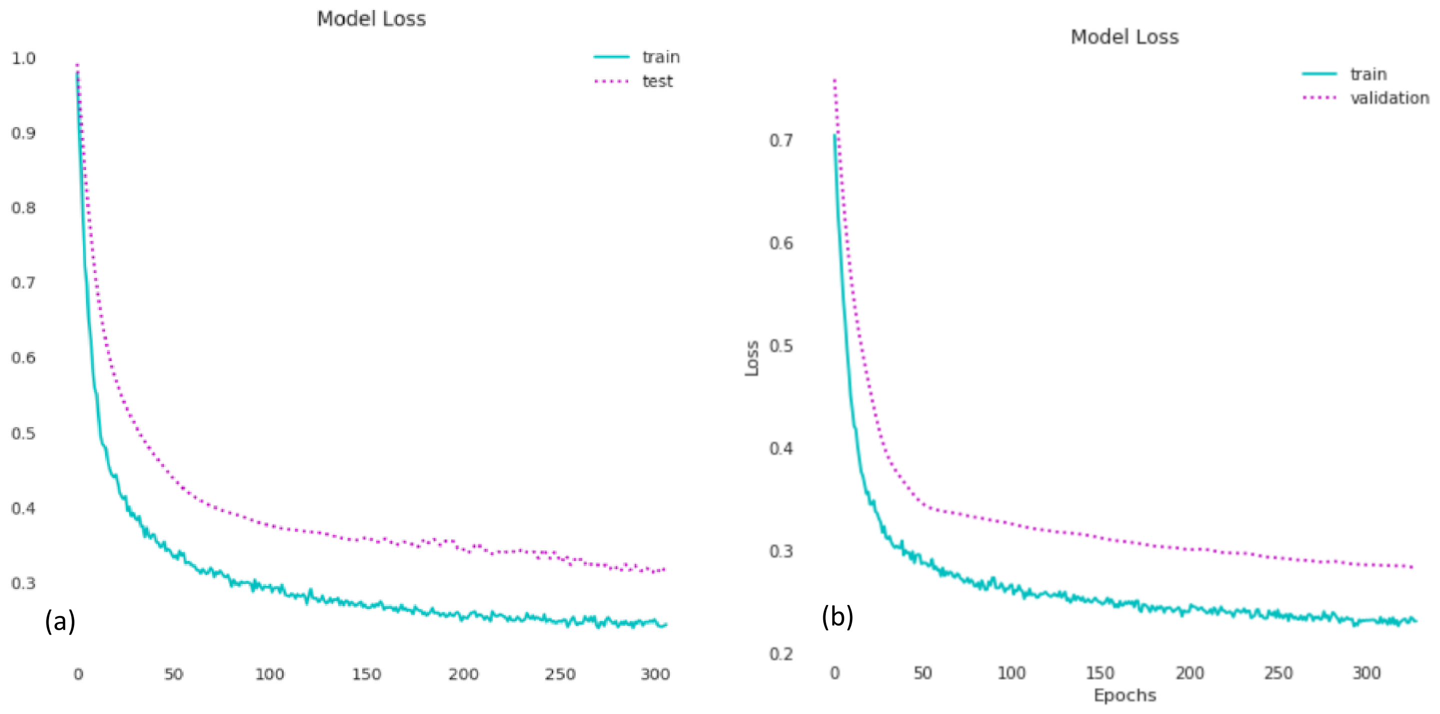
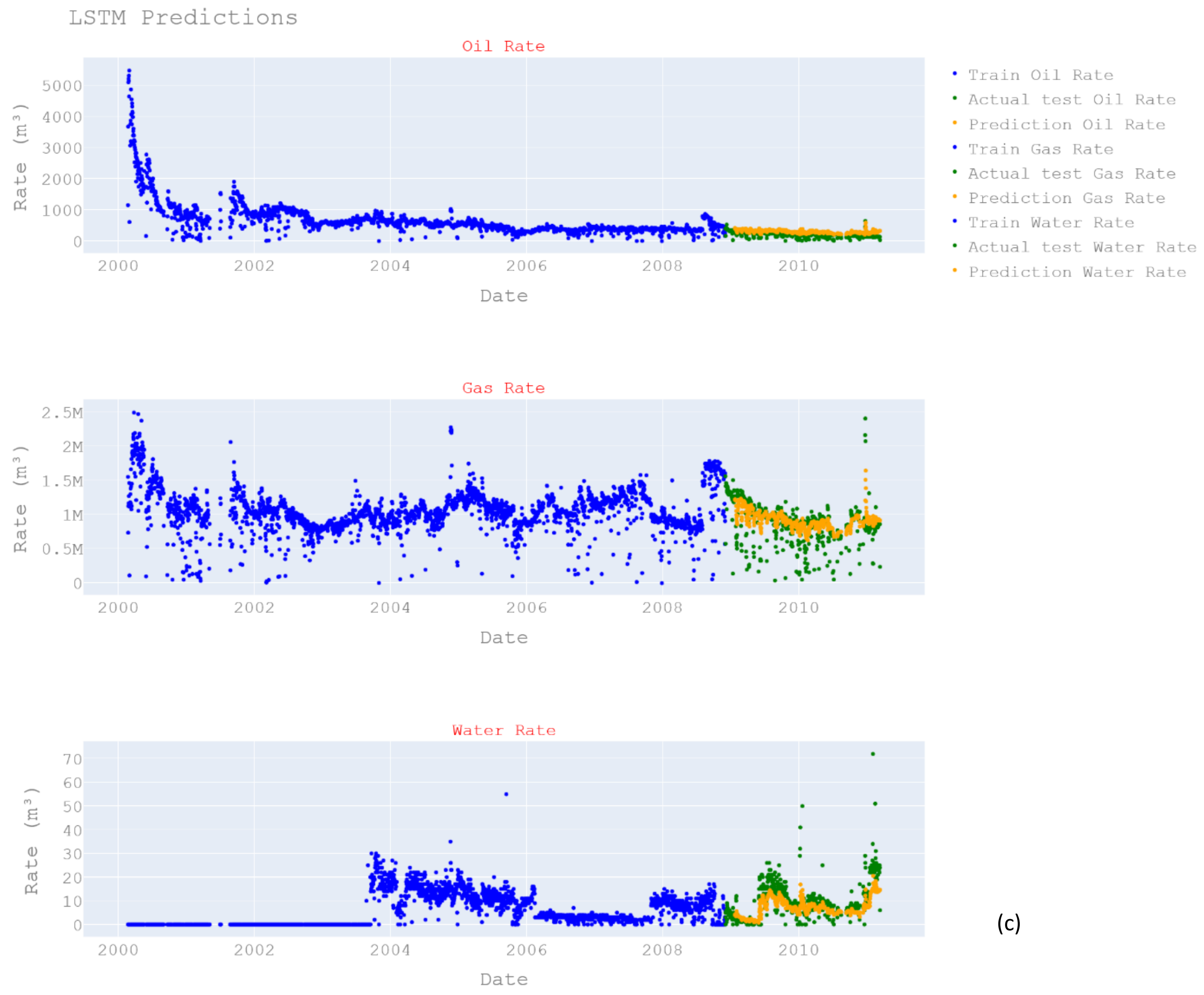


*Figure 15: Bi-LSTM predictions (c. next page) and its loss plot (b) of HD. 0.7 drop, 7 minutes and (c) the loss plot The validation set has a perpetually higher loss than the training set thus, there is overfitting. To be expected from such a small number of samples.*


LSTM Predictions
Oil Rate
Gas Rate
Water Rate
Rate (m³)
Date
2000
2002
2004
2006
2008
2010
5000
4000
3000
2000
1000
0
2.5M
2M
1.5M
1M
0.5M
70
60
50
40
30
20
10
Train Oil Rate
Actual test Oil Rate
Prediction Oil Rate
Train Gas Rate
Actual test Gas Rate
Prediction Gas Rate
Train Water Rate
Actual test Water Rate
Prediction Water Rate

(c)

# Chapter 4: Conclusion and Future Applications

From the observations, analysis and previous studies on deep learning networks, classic LSTM or bi-LSTM are the most ideal choices for back allocation applications due to low training time and high accuracy. The current data show that the classic LSTM may even be a better choice due to their unexpectedly strong results, and if tolerance for error is not extremely strict for the time efficiency. With only a two-hour training time, the best run had an R2 score over 99.9%. However, it is expected that bi-LSTM will perform better in more complex situations which we were unable to thoroughly test. It is the more stable network, performing well consistently making it more flexible than the classic LSTM. The correct choice of course depends on the exact application, error tolerance and the result of studying on more complex problems.

Regarding applications in industry, the software can be built upon for real-time production analysis to aid in production back allocation on a well by well basis and attempt to minimize sensor bias. This would allow the production metering system (multi-phase flow meters, pressure and temperature gauges) to be checked and any errors detected quickly. Furthermore, crossflow between two legs of a lateral production well could be reduced in order to maximize production efficiency.

As of now, there's no sure way to check for the best hyperparameters extremely quickly, grid search and random search are popular but all take time. However, there are new libraries and algorithms appearing everyday with intuitive ways to search for the best hyperparameters. In the future, libraries such as Elephas could be used to further utilize the Apache Spark integration of Databricks which would decrease training time further. Hyperparameters can be tested more quickly to find better trends in the flow rates. A high performing network would allow the industry to progress towards the imminent data driven future.

## Bibliography

Adrianov N. (2018). A Machine Learning Approach for Virtual Flow Metering and Forecasting, *IFAC PapersOnLine 51-8 191–196*

Aungiers J. (2018). 'Time Series Prediction Using LSTM Deep Neural Networks'. Altumintelligence.com. December 2016. Available at: https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks

Berg K., AS L., Ausenm H., Gregersen S., Bakken A. and Vannes K., Smørgrav S. (2015). Tie Backs and Partner Allocation. FMC Technoolgies Inc, Norway. *33rd International North Sea Flow Measurement Workshop*

Bikmukhametov T. and Jäschke J. (2020). First Principles and Machine Learning Virtual Flow Metering: A Literature Review, *Journal of Petroleum Science and Engineering 184 106487*

Bishop C. (1995) *Neural Networks for Pattern Recognition*. 1st edition. Oxford University Press. Cambridge, UK

CS231n., Convolutional Neural Networks for Visual Recognition., CS231n

Dahl E., Corneliussen S., Couput JP., Dykesteen E., Malde E., Moestue H., Moksnes P., Scheers L. and Tunheim H. (2005). Handbook for Multiphase Flow Metering. *Norwegian Society for Oil and Gas Measurement*

Falcone, G., Hewitt, G.F. and Alimonti, C. (2009). Multiphase Flow Metering: Principles and Applications. Elsevier. *Developments in Petroleum Science Vol 54*

Fink J. (2015). Petroleum Engineer's Guide to Oil Field Chemical and Fluids. *Gulf Professional Publishing 2nd edition*

Frantzen K. (2003). Well testing using Multiphase Meters. *The 21st International North Sea Flow Measurement Workshop*

George P., Song D., Carbonell J. (2018) GRADIENTS EXPLODE- DEEPNETWORKS ARE SHALLOW- RESNET EXPLAINED. *Workshop track – ICLR 2018 Carnegie Mellon University*

Graves A. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850
[cs.NE].

Graves A., Schmidhuber J. (2005) Framewise Phoneme Classification withBidirectional LSTM and Other Neural NetworkArchitectures. *IJCNN 2005*

Gupta D. (2017). 'Fundamentals of Deep Learning – Introduction to Recurrent Neural Networks'. Analyticsvidhya.com. December 2017 Available at:
https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/?utm_source=blog&utm_medium=cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning

Gupta T. (2017) 'Deep Learning: Feedforward Neural Network'. *towardsdatascience.com*. January 2017 Available at: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7

Hanin B. (2018) Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.*

He K., Zhang X., Ren S., Sun J. (2016) Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision andpattern recognition*, pages 770–778, 2016

Hochreiter S., Schmidhuber J. (1997) Long Short-Term Memory. *Neural Computation 9(8): 1735-1780*

Klambauer G., Unterthiner T., Mayr A., Hochreiter S. (2017) Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, pages 972–981

Krizhevsky A., Sutskever I., Hinton G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *University of Toronto*

Liu H., Motoda H. (1998). Feature Extraction, Construction and Selection: a data mining perspective, *Kluwer Academic Publishers.* 3-4

Ma X., Hovy E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *Language Technologies Institute Carnegie Mellon University arXiv:1603.01354v5*

Liu Y., Sun C., Lin L., Wang X. (2016). Learning Natural Language Inference using Bidirectional LSTM model andInner-Attention. *arXiv:1605.09090v1 [cs.CL]*

MacLennan B. (2017). Quantum Inspired Computational Intelligence. *Chapter 3: Field computation: A framework for quantum-inspired computing*

Nielsen M. (2019) Neural Networks and Deep Learning. *Determination Press. Ch2*

Ojha V., Abraham A., Snášel V., (2017) Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence Volume 60, April 2017, Pages 97-116*

Omrani P., Dobrovolschi I., Belfroid S., Kronberger P. and Munoz E. (2018). Improving the Accuracy of Virtual Flow Metering and Back-Allocation through Machine Learning. *Society of Petroleum Engineers SPE-192819-MS*

Pascanu R., Gulcehre C., Cho K., Bengio Y. (2013). How to Construct Deep Recurrent Neural Networks. *Cornell University.* In ICML'2013.

Reimers N., Gurevych I. (2017). Optimal Hyperparameters for Deep LSTM-Networks for SequenceLabeling Tasks. *Proceedings of the 2017 Conference on Empirical Methods in NaturalLanguage Processing (EMNLP),* Copenhagen, Denmark, September.

Sak H., Senior A., Beaufays F. (2014) Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*

Smith L. (2018) ADISCIPLINED APPROACH TO NEURAL NETWORKHYPER-PARAMETERS:PART1 – LEARNING RATE,BATCH SIZE,MOMENTUM,AND WEIGHT DECAY, *US Naval Research Laboratory. US Naval Research Laboratory Technical Report 5510-026*

Smith S., Kindermans. P.J., Ying C., Le Q.V. (2018) DON'TDECAY THE LEARNINGRATE, INCREASE THE BATCHSIZE. *arXiv:1711.00489v2 [cs.LG]*

Toskey E. (2011). RSPEA Evaluation of Flow Modelling Final Report. (Houston, TX)