Deep Learning Implementations on Acoustic Wave Processing with LSTM Architecture

A Dissertation submitted for the degree of

Master of Science

In

Data Analysis for Business Intelligence

by

Seckin Tataroglu

Supervisors: Prof Jeremy Levesley
Dr Said Assous
Department of Mathematics
University of Leicester

England

September 2019

To my family

Abstract

Recently, deep learning has become one of the most robust, flexible, accurate and reliable tools used in almost every field. The contribution of Google's TensorFlow machine learning platform in this new era is incontestable. By the decision of Google for making the TensorFlow platform open-source, a new perception started to spread out not only among mathematicians and computer scientists but also among entrepreneurs and large-scaled companies from all sectors.

Even though the deep learning methods are capable of handling almost any type of problem, their efficiency in finding proper solutions and the appropriateness of such methods to specific case studies should be well considered before deploying. This is due to the risk of their complicated tensor structures as well as due to the demanding calculation requirements of deep learning mechanisms. Therefore, before deciding to implement a deep learning model to business, it is advisable to investigate whether simpler solutions are available or whether deep learning support is indeed required.

This research focuses on analysing whether it is feasible to deploy deep learning for waveforms analysis and slowness prediction in the petroleum industry. The dataset used for this research was provided by Weatherford which is one of the pioneer companies in the oil and natural gas services field.

For this project, it was started by assessing the potential of getting the same results from a simplified version of the current industry standard. To do so, it was determined to work with extreme points and the global maximum and local minimum was suggested. Results showed that this approach was not appropriate for this case study due to inconsistent waveform structure.

Following this attempt, another deep learning model was built using LSTM architecture and consulting Google's TensorFlow library. This model was trained in different wells' data. The results suggested that, on one hand, the LSTM model is capable to learn and later predict the results similar to what it has already learned. On the other hand, this model results in completely different findings when requested to use data it has never encountered before.

Consequently, LSTM architecture is beneficial in terms of it is suitable for handling tensors. It results also show that as the model gets trained, it is more likely to get more accurate results.

Acknowledgements

This research has been done in requirements for the partial fulfilment of the Master's degree in Data Analysis for Business Intelligence.

First of all, I would like to express my sincere gratitude to my supervisor Professor Jeremy Levesley who helped me a lot with his enthusiasm, knowledge and amazing motivation throughout this dissertation. He inspired and guided me with his precious advice, continuous support and kind personality during the whole course of my master's studies.

I also would like to thank Dr Said Assous from Weatherford for all his support. He explained sonic logging in detail and expanded my knowledge in this new field. He somehow changed my perception of life by pushing me to do and learn more.

I must also thank all the lecturers and professors especially Professor Alexander Gorban and Dr Evgeny Mirkes who shared their experiences with me / my class during this year.

Last but not least, I would like to express all my gratefulness to my family and my friends both in Istanbul and Leicester. Special thanks must go to Eleni Samantzopoulou for being by my side all year long, Vishal Sharma for all the sleepless night study sessions, Qusay Hawari for his friendship and mental support during the long library study sessions in this summer and Yaşar Kemal Peştreli who sparked everything.

Contents

List of Figures

Chapter 1

Introduction

1.1 A Brief History

Even though there is a distinctive difference between traditional sonic logging and borehole sonic logging, the origins of the two methods are both based on surface seismic techniques which have been used since the 1920s in oil and gas exploration. [1]

In 1927, the Geophysical Research Company started to collect velocity logs in France by generating explosions on the surface and recording the arrival times of sound at specific depths determined by geophysicists in order to investigate the time-depth relationship at the borehole.

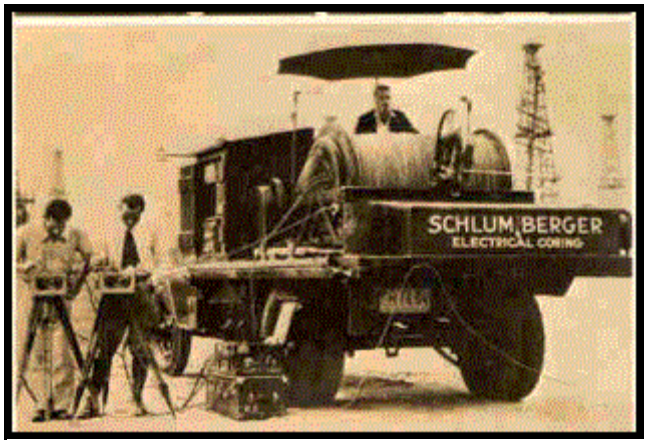

Figure 1 Schlumberger Wireline Truck

The logic behind this theory was to measure the propagation of a sound that is generated from a logging tool, travels through rock and returns to receivers of the initial logging tool. This technology allowed geophysicists to assess the underground

[1] (Close, Cho, Horn, & Edmundson, 2009)

structure by creating only a small borehole instead of a large-scale drill. The results of the sonic logging could show whether there was precious material to extract as well as how costly this process would be.

Henry Doll, (Conrad Schlumberger's son in law) was the first to attempt a well logging on 5th of September 1927 in a 500-meter-deep well in Pechelbronn, France. This logging was recorded as the first electrical resistivity well log. After this first attempt, Schlumberger Well Surveys Inc. made their wireline truck and cable (Fig. 1) commercial service available to borehole owners in order to measure the velocity of each depth level and get a simple photograph of the underground.

In 1932, [2] Conrad Schlumberger and his brother Marcel Schlumberger presented the first comprehensive paper on well logging titled "Electrical Coring: A Method of Determining Bottom-Hole Data by Electrical Measurements" to the American Institute of Mining and Metallurgical Engineers in New York. In this paper, Conrad Schlumberger explains how to use a transmitter (sound sender) and only two receivers in order to measure the speed of sound and accordingly the velocity. Modern sonic logging is based upon these principles (Fig. 2).

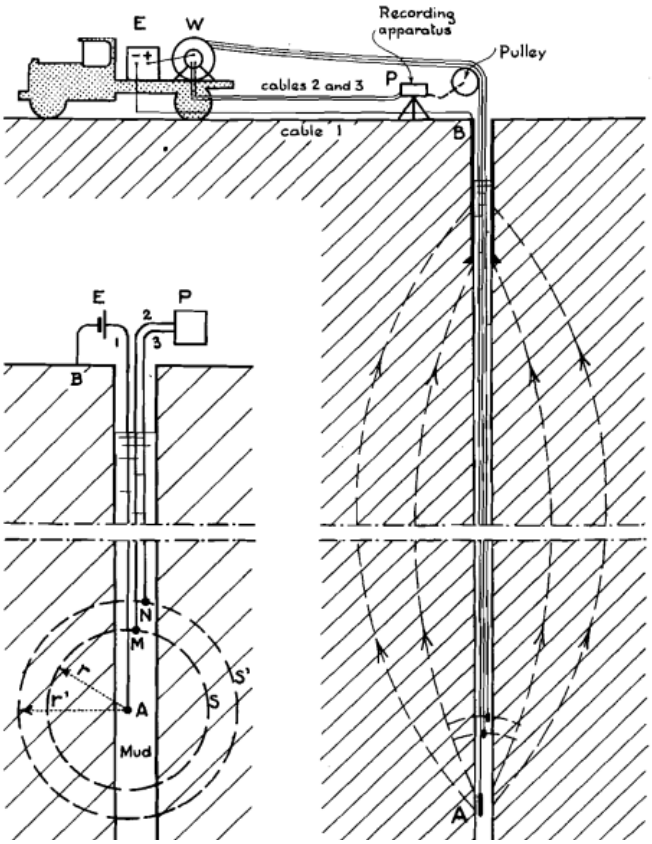

Figure 2 Scheme of Electrical Coring Device and Outline of Operation

In order to provide a flow connection between ground and underground, electrode A is connected by an insulated cable 1 to one of the poles of battery E which is located at the surface, the other pole is grounded to earth contact B which is situated to the drill hole. In order to measure the differences between M and N,

[2] (Schlumberger & Schlumberger, 1932)

these electrodes are linked to the terminals of a potentiometer P located at the surface by two cables 2 and 3. Knowing the distances r and r', the intensity i of the current passed through the ground, and the difference of potential ΔV between M and N, the average resistivity of the ground R in the region of the measuring arrangement AMN, can be calculated.[3]

There have been numerous significant findings in the sector since that time. In 1942, Gus Archie(A petro-physician of Shell)'s paper[4] brought another ground-breaking approach also known as "Archie's Law". It explains the relationship between electrical resistivity and oil bearer rocks. By 1970, the sonic logging tools changed and more accurate logs started to appear. Nowadays, due to the contribution of technology, nowadays, we can monitor the underground even during the drilling process. Through wired logging, we can get more accurate loggings which enables professionals to identify the correct velocity of a borehole.

1.2 Types of Sonic Logging Tools

In this section, the types of logging tools are described based on the source of the sound they collect and the technique they use to do so.

Sonic logging tools mainly fall into two categories. These are examined by the transmitter and receiver architectures of sonic logging tools.

1.2.1 Transmitter Architectures

A device that pulses sound in specific intervals is called "transmitter". This is why it is also described as the "Energy Source" of the tool.

Based on the transmitter type, sonic tools are divided into three classes, namely monopole, dipole and quadrupole sonic tools[5] (Fig 3).

[3] (Schlumberger & Schlumberger, 1932)
[4] (Archie, 1942)
[5] (Zemanek, May 1991)

Figure 3 Direction of pressure waves from (left to right), monopole, dipole and quadrupole sources

1.2.1.1 Monopole Sources

Monopole transmitters emit energy towards all directions radially from the tool axis. They are sometimes called axisymmetric or radially symmetric sources.[6]

One type of the earliest tools is the wireline sonic tools contain a monopole transmitter along with at least two monopole receivers. These tools are designed to generate conventional compressional sonic logs.

When the sound reaches the rock, it starts travelling along the borehole inside the rock. This sound is refracted back to the borehole and is collected by the receivers. The travel time between the two receivers is known as *slowness*. Sound velocity is the inverse of slowness.

In fast formations,[7] this design can also receive shear waves beside the compressional waves which are converted to shear energy.

Overall, monopole sources are considered relatively older in terms of technology, whilst there are some specific scenarios that monopole sources are still more effective than the other sources.

[6] (Crain, How Many Acoustic Waves Can Dance On The Head Of A Sonic Log?, 2004)
[7] Fast Formation is a formation where the velocity of the compressional wave travelling through the borehole is less than the velocity of the shear waves through the surrounding formation. (Crain, How Many Acoustic Waves Can Dance On The Head Of A Sonic Log?, 2004)

1.2.1.2 Dipole Sources

Dipole transmitters are relatively newer in terms of technology as they emit the energy towards a specific direction instead of doing so radially. This is why they are also called asymmetric or non-axisymmetric sources.

The advantage of dipole transmitters is that they can generate a compressional wave in the formation and, in addition, they can pulse a strong shear wave both in slow and fast formations. This is known as *flexural wave* and it travels on the borehole wall. [8]

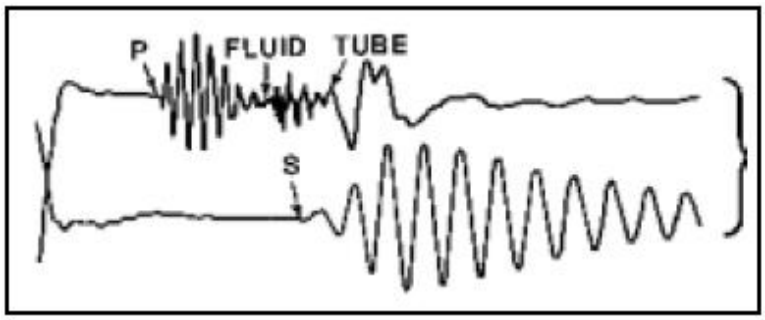

Figure 4 Monopole (upper) and dipole (lower) waveforms in a slow formation

As dipole sources direct energy towards a single direction, clearer logging can be obtained. Thus, p-wave, fluid and tube waves disappear and a higher amplitude is gathered on the Shear wave (Fig. 4). This type of waveforms makes the analysis of slowness easier as finding maxima and minima are encountered easier.

Modern sonic tools have two dipole sources placed orthogonally upon each other. These are called X- and Y- dipoles or axes. (Fig. 5). Moreover, these tools have corresponding receivers separately assigned to collect X-pole sounds and Y-pole sounds. These special tools are known as

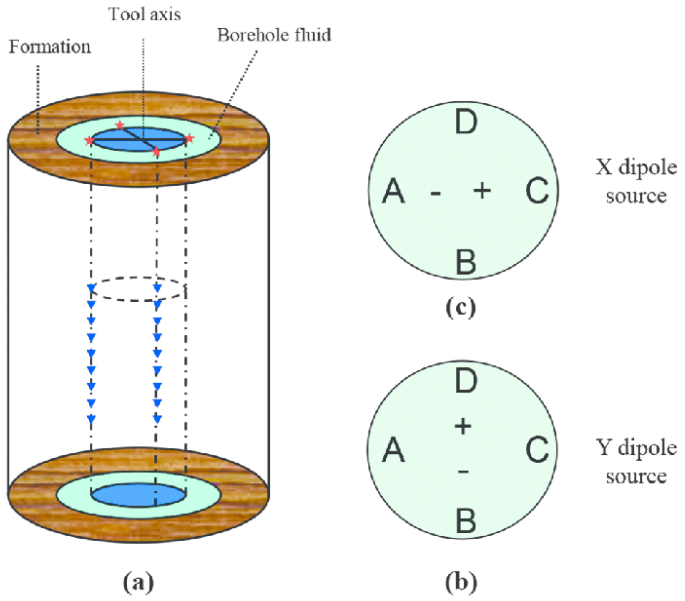

Figure 5 Tool Geometry: (a) Logging Tool Placement, (b) Y Dipole, (c) X Dipole

[8] (Crain, How Many Acoustic Waves Can Dance On The Head Of A Sonic Log?, 2004)

crossed-dipole tools. Sequentially, the X and Y pole pulse sound towards specified directions and the corresponding receivers collect sound. During the last stage of the logging, two different waveforms are obtained.

Figure 4 shows a waveform obtained from a monopole transmitter in a slow formation where there is a compressional wave (P), yet no shear arrival is found. The lower waveform was captured by a dipole tool at the same depth as the borehole. As one can notice in Figure 4, the dipole wave carries no compressional wave, but yet a clear shear (S) arrivals. Note that the shear wave arrives after the fluid wave (This is the definition of a slow formation).[9]

The data used in this dissertation was collected by a dipole source.

1.2.1.3 Quadrupole Sources

Quadrupole transmitters form a special kind of sources generating asymmetric pressure waves, named as *screw waves*. Principally, working logic is highly similar to dipole sources. However, these tools are commercially unavailable.

A different way of logging known as "logging-while-drilling" requires these special quadrupole sources. Even though it is known that this way of logging is not efficient for identifying velocity, recent technological developments have been successful in measuring shear velocity while drilling.

1.2.2 Receiver Architectures

In this section, different types of receivers are described after a brief introduction to early drilling tools. Taking into account their receiver architecture, logging tools can be divided into mono receiver tools, dual receiver tools, borehole compensated (BHC) tools and long spacing sonic (LSS) tools.

[9] (Crain, How Many Acoustic Waves Can Dance On The Head Of A Sonic Log?, 2004, p. 10)

1.2.2.1 Early Tools

Early tools consisted of only one transmitter (Tx) and one receiver (Rx) (Fig. 6). In order to provide sonic isolation, the body was covered by rubber. This rubber body had a function of stopping waves travelling through the tool to the receiver.

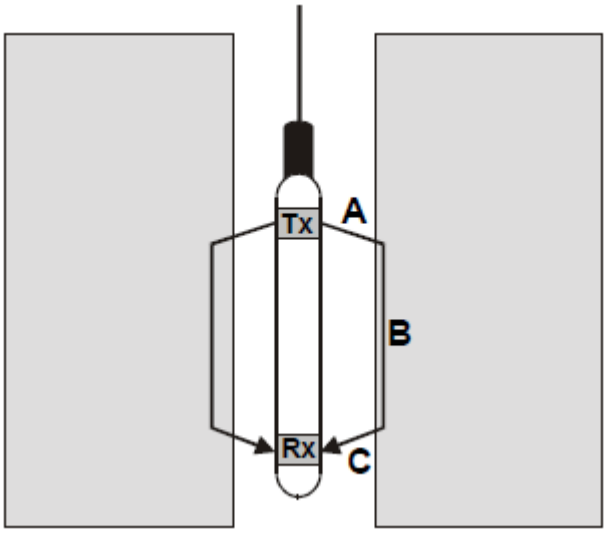

Figure 6 Early Sonic Tools

These type of tools had two main problems. The first concerned errors in the measured time occurring from the long travel time of the sound. This is because the calculation included A, B and C into the measured time rather than just B. The second problem was a miscalculation of velocity. As the formation length B was not constant, the velocity of the wave was depending upon the formation altered the refraction angle.

1.2.2.2 Dual Receiver Tools

Dual receiver tools were designed to overcome the problems of the early tools.[10] The bodies of these tools contain two receivers situated within a few feet distance from each other, in order to measure the difference between arrival times. This amount of time is called *sonic interval transit time* (Δt) and is the time of distance difference between the receivers, or interval D.

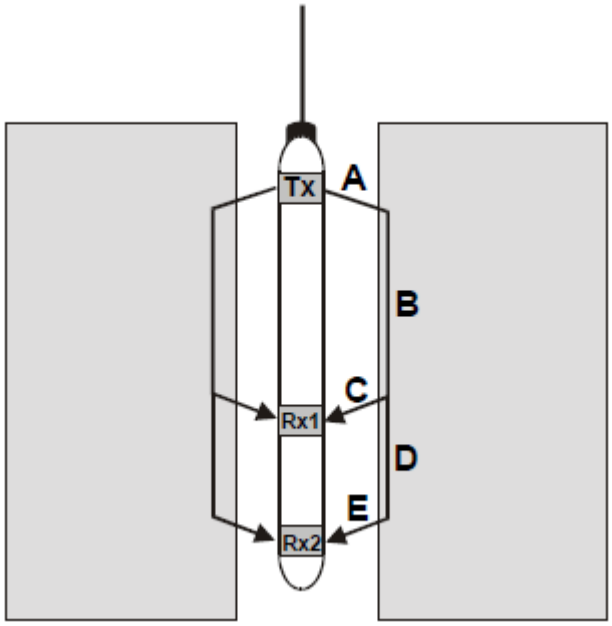

Figure 7 Dual Receiver Sonic Tools

[10] (The Sonic or Acoustic Log, p. 176)

This new tool type solved the first problem of the early tools, which was the inclusion of unnecessary travel time to the calculation. Yet, this tool type works effectively unless it is located vertically in the borehole. Once the tool is tilted, the A, C and E angles are changed and the velocity calculation to deviate.

1.2.2.3 Borehole Compensated Sonic *(BHC)* Tools

This tool type automatically overcomes problems caused by misalignment and the varying sizes of holes. These tools carry two transmitters and two sets of receivers on their body. One transmitter and a set of receivers are located invertedly on the opposite direction of the other transmitter and the set of receivers. The transmitters and the corresponding receivers work sequentially and the Δt values are measured one after the other. These two collected values of Δt are then averaged in order to offset the misalignment of the tool.

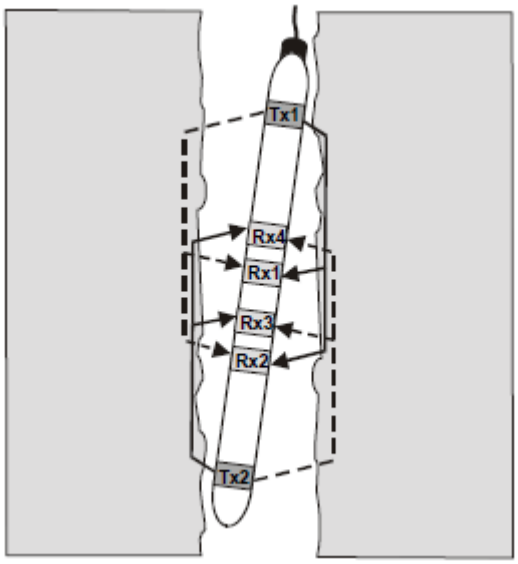

Figure 8 Borehole Compensated Sonic Tools

1.2.2.4 Long Spacing Sonic *(LSS)* Tools

Drawing on the experience of petro-physicists on sonic logging, it is noticeable that in some logging environments, a longer transmitter-receiver distance may be more helpful. Hence, Schlumberger developed the long spacing sonic (LSS) tool which consists of two transmitters and two receivers. Eventually, two different log data are gathered by this tool.

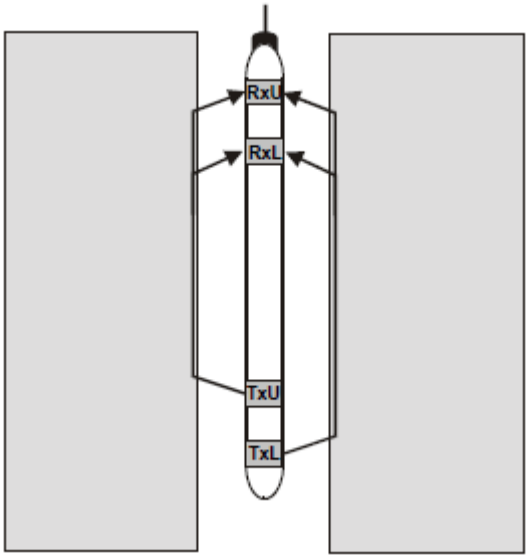

Figure 9 Long Spacing Sonic Tool

1.3 Wave Types

In wireline logging terminology, acoustic waveforms are divided into two main categories depending on whether the propagation direction of sound is originated by a monopole transmitter or a dipole transmitter. In this section, these sound waves are explained briefly.

1.3.1 Waveforms of Monopole Sources

Monopole transmitters emit sound equally towards every direction and therefore a spherical emission is generated from the source. Once the sound is transmitted, the direction of wave propagation is always perpendicular to the wave. This simple case also assumes that the formation is homogenous and isotropic and that the sonic tool itself has no other effect on wave propagation.[11]

In order to analyse the sound propagation easily, the 3D spherical emission is converted to 2D view. In this 2D simplification, when the waveform is fired from the transmitter, it meets the borehole wall, the waveform is refracted and mainly three new waveforms are generated. They are named as Compressional, Shear, and Stoneley respect to the order of travelling speed (Fig. 10).

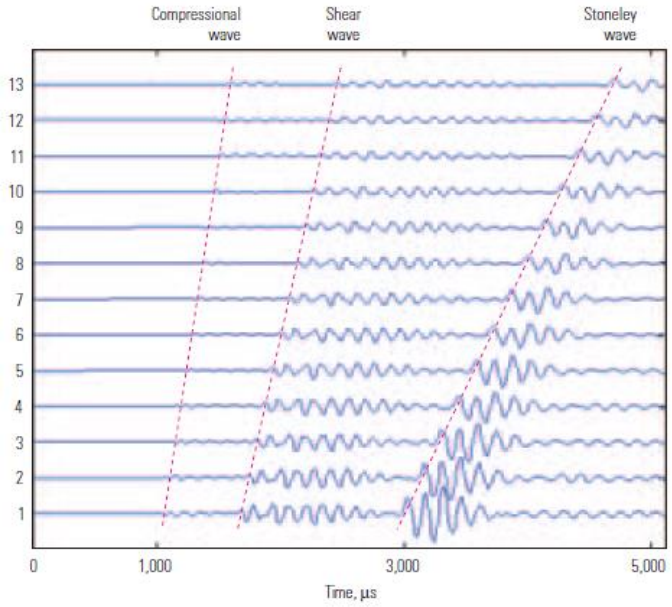

Figure 10 Compressional, Shear and Stoneley Waves

The petro-physicists who use monopole transmitter for well logging are interested in the "Stoneley Wave" as it relates to the extraction of the velocity of the specific depth. Other waves are used to understand the structure of the borehole.

[11] (Haldorsen, et al., 2006)

1.3.2 Waveforms of Dipole Sources

Where monopole transmitters are insufficient, for example, in slow formations, dipole transmitters can be effective. They produce an extra flexural wave. Thus, dipole transmitters are considered to be newer than monopole transmitters.

Typically, the tool contains two orthogonally oriented dipole sources called X- and Y- dipoles or axes. Dipole transmitters separately fire sounds. First, the X-pole fires sound and X- dipole receivers collect the waves. Then the same process is repeated by the Y axe. Then the depth is changed and the procedure continues as described above.

As dipole transmitters generate flexural and dispersive waves slowness changes along with frequency. Also, amplitude changes are noticed in different receivers. While the first receiver gets the highest amplitude, the last receiver gets a flatter waveform which has a lower amplitude comparing the first wave received.

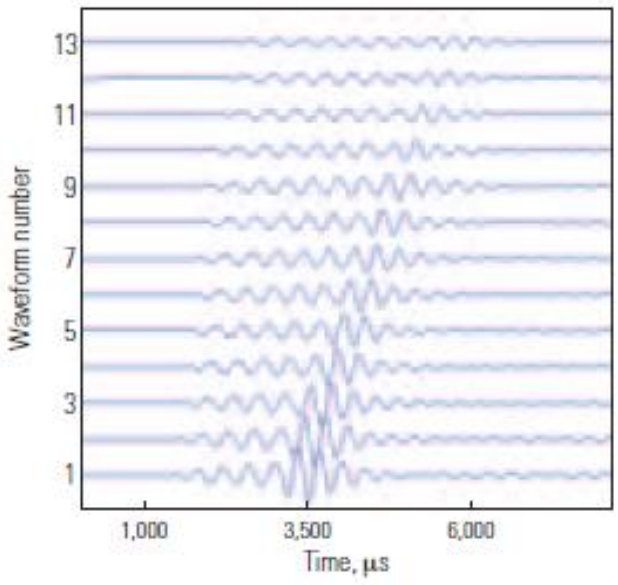

Figure 11 Flexural waveforms

1.4 Sonic Log Analysis

Once the well logging is completed, logs are sent to analysts specialised in sonic logging interpretation. Their main objective is to calculate the slowness of each depth level of the borehole in order to inform other geophysicists of the borehole structure. This is done by analysing what materials are encountered around the borehole.

Slowness can be calculated by using signal processing methods known as semblance or as coherence. This method is based on searching a set of partial

waveforms that maximizes the coherence in the given wave window. By calculating the maxima corresponding to compressional, shear and Stoneley slowness are plotted (Fig. 12) for each depth creates a slowness log.

Sometimes analysts deploy an additional method to process dipole logs. As one dipole transmitter fires sound on two opposite ways and the sound is collected by receivers located on the two opposite sides of the logging tool, geophysicists usually subtract the two waveforms and a new waveform is gathered. Thus, the compressional and shear waves give their place to a clearer of Stoneley wave which still includes the same slowness. As a result, the analysis of the borehole slowness becomes easier.

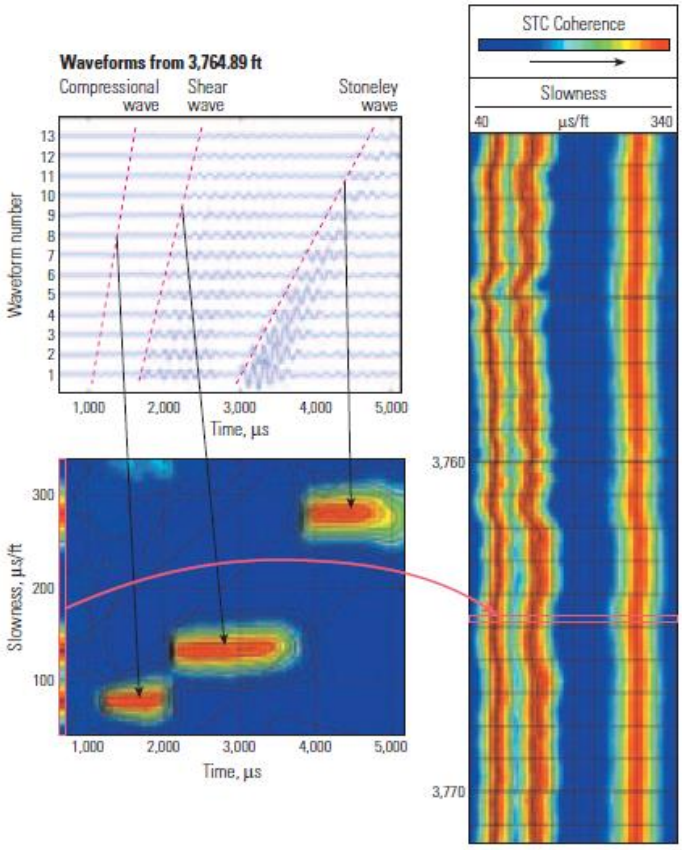

Figure 12 Slowness-time-coherence (STC)

1.5 Problem Statement and Structure of The Dissertation

For this research project, worked with Weatherford to determine whether LSTM deep learning architecture is feasible for interpreting the velocity of sonic logging data.

As the business cycle, Weatherford works with numerous log analysts to examine the sonic log outputs in order to draw a picture of the velocity of the corresponding borehole. Even experienced analysts require at least 5 to 6 hours to complete acoustic analysis of a well. Moreover, different analysts draw slightly different results as different methods are used to examine the data. Therefore, it is demonstrable that the human factor affects the accuracy of results as human individuals are easily influenced by their emotional states.

This is why an interpretation machine such as a neural network, would be helpful in terms of eliminating the operational risks mentioned above as well as accelerating the business process.

Overall, this dissertation addressed the question of whether a proper interpretation machine by using LSTM deep learning architecture based on the previously collected sonic log and interpreted velocity data can be created.

This dissertation is composed of six chapters and is organised as follows:

In Chapter 1, the brief history of sonic logging is referred and, the different types of logging tools used today are described as well as how these tools produce sound and collect waveforms. It is also explained how analysts process waveforms in order to find the slowness tomography of the borehole.

In Chapter 2, an overview of the global maxima and local minima approach is given. At the beginning of the chapter global maxima and local minima definition is introduced. Then it is mentioned about Savitzky-Golay smoothing algorithm and how effective smoothing is on the signal processing in Section 2.2. After explaining pre-processing in Section 2.3, it is presented the global maxima local minima model used for this project in detail and is analysed how it can be applied

to the data in order to calculate slowness. Finally, in Section 2.4, the problems and the solutions for this approach discussed.

In Chapter 3, the LSTM architecture was reviewed. This is one of the most popular deep learning architecture used for sequenced data nowadays. Google's TensorFlow library was used extensively in this research. In Section 3.3, the TensorFlow use, the LSTM neuron structure, layers, loss functions and optimisers are further explained. At last, the problem types that LSTM architecture can handle are listed.

Chapter 4 explain in details, how the deep learning model is built and how it produces results. In Section 4.1, it is analysed the structure of the data used for this research as well as how the input data was processed to import the neural network. Section 4.4 describes model training. Additionally, overfitting is explained as well as how overfitting can be avoided in a deep learning model.

Chapter 5 refers to the business contribution of this model. The further details are given how to calculate the slowness of a borehole and compare the calculation cost of the traditional way. Then the advantages of the model are demonstrated.

In Chapter 6, results are summarised to conclude this research. Finally, ideas for possible future research projects are discussed.

1.6 Literature Review

The use of deep learning methods in the petroleum industry spiked after 2014 when machine learning tools became publicly available. This dissertation is concerned with acoustic wave processing as well as LSTM implementations.

The studies on acoustic wave processing and LSTM implementations are divided into two parts. The first examines the correlation between acoustic waves and other measurements such as gamma-ray, x-ray and, porosity. Based on this correlation, multi-linear regression solutions are tested for slowness prediction.[12]

[12] (Akhundi, Ghafoori, & Lashkaripour, 2014)

These studies do not provide satisfactory results regarding the implementation of neural networks on acoustic waves.

The second part focuses on deep learning methods. These studies are only limited to classify rock facies[13] or basic implementations of RNNs (Recurrent Neural Networks).[14]

However, there are other studies related to LSTM architecture, but they focus on predicting missing log values.[15] Also, some focus on sonic log prediction based on other parameters, such as gamma-ray, bulk density and neutron porosity.[16]

When it comes to slowness, some researchers have previously deployed ANNs (Artificial Neural Networks) to predict slowness, but none research project used acoustic logs with LSTM architecture to do so.

Table 1 summarizes several studies for acoustic wave processing and ANN implementations.

Authors	**Target**	**Methodology**
(Chen & Zeng, 2018)	Facies Classification	Support Vector Machine
(Dahai, Jun, Qian, Yuanyuan, & Hanghang, 2019)	Facies Classification	Support Vector Machine
(Akhundi, Ghafoori, & Lashkaripour, 2014)	Slowness Prediction	Multiple Regression – RNN
(Konate, Khalid, Pan, & Li, 2015)	Facies Classification	KNN
(Zhang, Chen, & Meng, 2018)	Log Interpolation	LSTM
(Izurieta, Rocha, & Sui, 2019)	Facies Classification	K-Means
(Jianhua, Yancui, Dan, & Xiankun, 2017)	Log Prediction	ELM – KELM
(Pham & Naeini, 2019)	Log Interpolation	FCNN – RNN
(An, Yang, & Zhang, 2018)	Porosity Prediction	Non-Linear Optimisation
(Tariq, Elkatatny, Mahmoud, & Abdulraheem, 2016)	Sonic Log Prediction	ANN – Regression

[13] (Chen & Zeng, 2018)
[14] (Akhundi, Ghafoori, & Lashkaripour, 2014)
[15] (Zhang, Chen, & Meng, 2018)
[16] (Tariq, Elkatatny, Mahmoud, & Abdulraheem, 2016)

1.7 Materials and Methodology

The dataset used for this research was provided by Weatherford. This dataset contains six different well's (borehole) sonic logs and corresponding observations on slowness examined by analysts in Weatherford.

Each well's sonic log file contains 16385 columns. Only one out of these refers to the depth, while the rest refer to eight different receivers each having four different directions. Each soundwave comprises of 512 data points. The receivers were named as A, B, C and D and were symmetrically located on each other's side. Each couple of receivers is named a *dipole*. A and C constitute the X-Pole, while B and D constitute the Y-Pole. X-Pole and Y-Pole were located orthogonally.

Python programming language with Anaconda environment was used to handle data. Then, Pandas, NumPy libraries were used to complement the manipulation of data.

Prior to pre-processing the data, each dipole was examined separately. Firstly, unrelated dataset values were eliminated and the filtered data was matched with observations provided by Weatherford. As symmetric receivers mostly comprise of the same compressional and shear waveforms, C values subtracted from A values and D values subtracted from B values, in order to extract plain Stoneley waveforms. Hence, new shapes of waveforms were obtained.

Two different methods were deployed to calculate the slowness of boreholes. Firstly, the extreme points were examined. Secondly, a neural network model with LSTM architecture was trained.

The theory of extreme points approach locates the maximum absolute value on a waveform and, with respect to the global maxima (or minima), also locates the two nearest local minima (or maxima). Eventually, based on these locations and the values of the points, it creates a linear regression line. The higher R^2 value is, the better the accuracy gained on the slowness value.

For this project, an algorithm was written that initially locates the absolute maximum value on the first waveform. This point is considered to be the global maxima or global minima depending on its direction. The global maxima location

is the point where the waveform splits into two other waveforms, in order to find two local minima points. The waveform on the left is used for locating the first local minima and the waveform on the right is used for locating the second local minima. [17] After locating eight different maxima, first minima and second minima, linear regression is performed by using their values. Linear regression is used in order to find a perfect match between the eight waveforms. Hence, a higher R^2 value gives a higher level of match between the waveforms.[18]

This approach only worked on clean datasets. In cases of unstable waveforms, it became much harder to locate extreme points and, therefore, wrong estimations on slowness followed. In order to overcome this problem, a smoothing algorithm namely the Savitzky-Golay Filtering algorithm was used to handle insignificant peaks.

The Savitzky-Golay is a digital filtering algorithm that applies to any time series dataset and eliminates noises without distorting the signal tendency. 41 data points were used for the smoothing window and filtering was applied with the third order of polynomial smoothing. Filtering improved the efficiency of the algorithm, but only to a certain degree. On waveforms carrying a large number of noises, filtering was not especially successful and, as a result, the algorithm was not able to draw regression lines correctly. Due to its unstability, this approach considered inappropriate for estimating slowness.

The deep learning approach aimed to solve a regression problem as the problem was a best-fit / optimisation problem. For this purpose, a model with LSTM architecture was built through Google's TensorFlow and Keras libraries. As input, the tensor structure was used as it suits the LSTM architecture.

The LSTM architecture was essentially built for image recognition and, translation functions that require a matrix and a sample size as the third axis. Accordingly, the eight different waveforms by each 512 data points actually build an 8 by 512 matrix. This is why every matrix was introduced as an image and the

[17] Global Maximum, the firs Local minimum and the second Local Minimum are named as "Max", "Min1" and "Min2" respectively in Appendix 1.
[18] Note that: Slowness is the reverse of slope.

number of samples was set as input size which is recognised as the third dimension by LSTM architecture.

The model consisted of four hidden layers. The number of Epochs[19] was set as 150, however, TensorFlow's Early Stopping feature was used to avoid overfitting. The Batch Size[20] was set as default. The training Set – Test Set ratio was set to 50%.

The model was trained on over 25592 samples. In the first phase, the model was trained using only one borehole data. Due to the narrowness of this sample, the model was not able to predict the slowness values of the other borehole's. In the second phase, the model was trained using all six boreholes as a sequence. It was observed that the model tended to adjust itself to the last dataset it trained on. Hence, the model was not able to predict all different borehole data correctly but only the one last trained. The accuracy fluctuated between 69% and 98%. Eventually, in order to stop the model to fit only one borehole, another small algorithm was coded. It randomly generates a sample over all six boreholes by shuffling 25592 matrices. As a result, relatively more stable results were obtained and the accuracy took place with a narrower range between 73% and 93%. After this alternation, the model was capable of processing 4000 to 5000 matrices in less than 30 seconds.

[19] Epoch is one pass through all samples in the training set and updating the network weights LSTMs may be trained for tens, hundereds, or thousands of epochs. (Brownlee, 2017, p. 46)
[20] Batch is a pass through a subset of samples in the training set after which the network weights are updated. One epoch is comprised of one or more batchs. (Brownlee, 2017, p. 46)

Chapter 2

Global Maxima – Local Minima

This section is focussed on the theory of global maxima (or minima) and local maxima (or minima). It explains how this approach can analysts to calculate slowness.[21] Prior to applying this model to the case study examined, the Savitzky-Golay smoothing algorithm is described in order to clarify its role in calculating slowness.

2.1 Introduction to Global Maxima-Local Minima

In mathematics literature, the maxima and minima, or respective plurals of maximum and minimum of a function are known collectively as extrema. They are the largest and smallest value that the function can reach, either within a given range (the local or relative extrema) or within the entire domain of a function (the global or absolute extrema). [22]

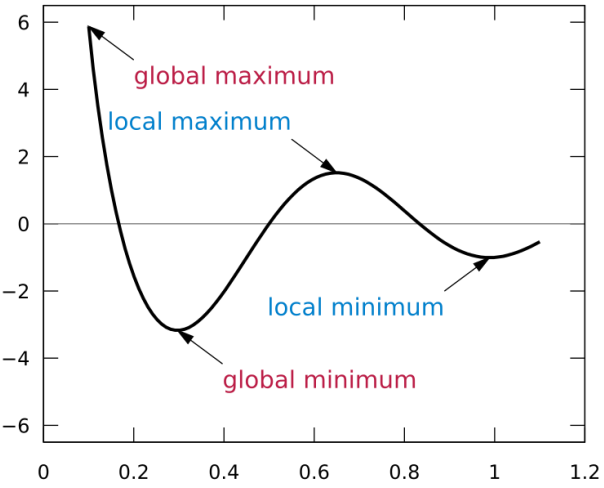

Figure 13 Local and Global Maxima and Minima

The extreme points are accepted the turning points of the functions. In order to precise, the

[21] Once the extreme points are found, the slope of the whole picture is tried to find as the slowness is the inverse of the slope.
[22] (Stewart, 2008)

location of these points, algebra and differential calculus is benefited. It should be noted that the tangent values of the turning points are always zero.

As Figure 14 demonstrates, not all the zero tangent points are turning points. At points A, B and C, $\frac{d_y}{d_x} = 0$. Although all points have a zero tangent, only A and B can be defined as turning points. Moreover, all three points are named as stationary points. It can be easily deduced that even though not all the points where $\frac{d_y}{d_x} = 0$ are turning points, all the turning points are stationary points.

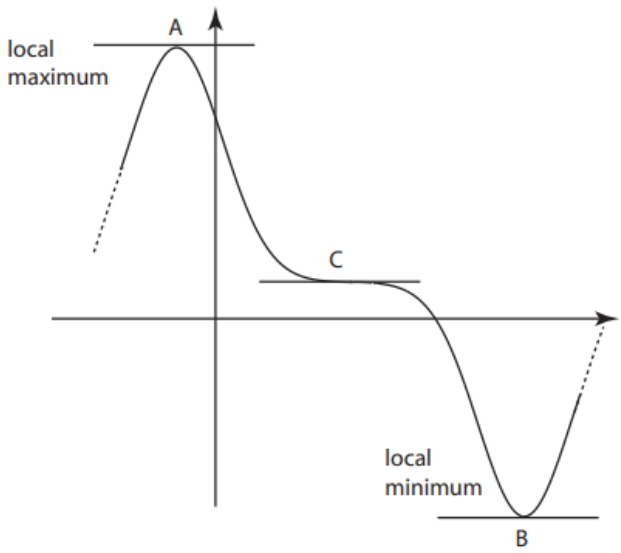

Figure 14 Gradients of Extreme Points

In order to distinguish the turning points from the stationary points, it is necessary to check the step before and step after the point where $\frac{d_y}{d_x}$ equals zero (Fig. 14). As seen in Figure 15[23], to the left of the minimum point, $\frac{d_y}{d_x}$ is negative as the gradient of the tangent is negative. At the minimum point, it is zero and to the right of the minimum point, the gradient is positive. Overall, by looking at the step before, the point itself and the step after, one can conclude the current point is the local minimum in the given window as long as the point itself has a zero gradient as well as the previous step has negative and the next step has a positive gradient.

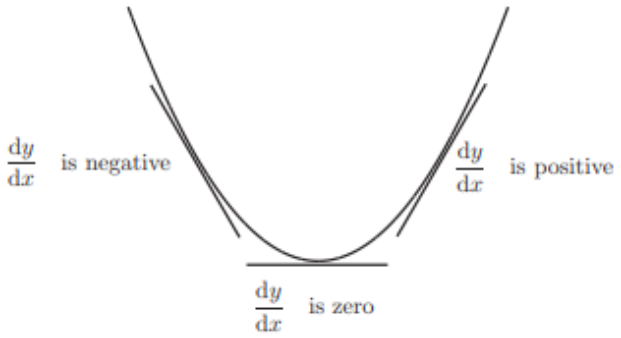

Figure 15 Tangent changing on Turning Point

[23] (MathCentre, 2009)

2.2 Savitzky-Golay Smoothing Algorithm

Abraham Savitzky and Marcel J. E. Golay created a filtering algorithm, which serves in smoothing on sequential data.

The Savitzky-Golay algorithm is based on local least-squares polynomial approximation. This allows the algorithm to avoid distorting the signal tendency and shifting the data, unlike many filtering algorithms.

In this research, the SciPy Savgol Filter was used in order to smoothen waveforms.

In this pre-defined function two parameters accompany the input itself: Window Length and Polyorder.

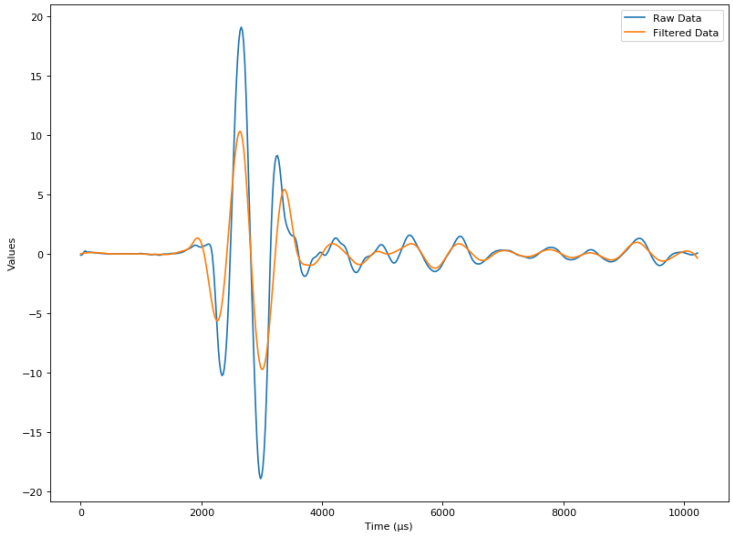

Figure 16 A Comparison of Raw and Filtered Data Smoothed by Savgol

Window Length serves in capturing how many sequential data points are involved in the smoothing process.

Polyorder determines the optimal order in terms of finding the polynomial curve for the data points because of Savitzky-Golay tries to fit polynomials of the data points.

Both in Figure 16 and generally in this research, window length and poly order were determined as 41 and 3 respectively.

2.3 Data Analysis and Application

This section explains how the global maxima local minima approach is applied and refers to the reason why it is considered to apply to this dataset.

First of all, as dipole source waveforms are examined in this dissertation in order to eliminate compressional and shear waveforms, the data diagonal receivers subtracted each other and gathered new shapes of waveforms for each depth level. The new, subtracted data is used for this research.

As mentioned previously, in order to find the extreme points (turning points), differential algebra is used for the given function. In this case study, instead of having a function, the data points were already obtained. Therefore, differential algebra was not used to find the extreme points. Instead, sorting algorithms were used.

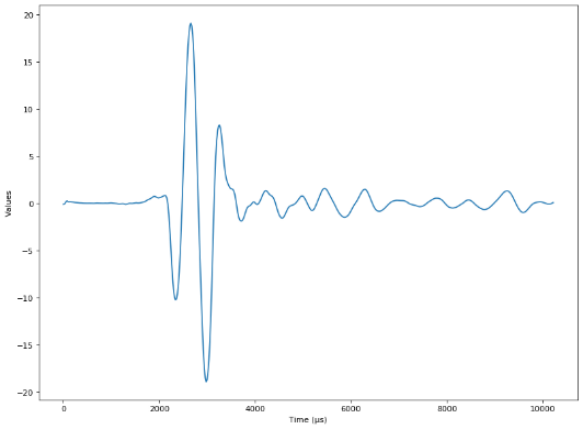

Figure 17 An Example of Single Waveform

The main idea behind this approach is that the analyst locates the extreme points for each waveform. As previously explained, (quick reminder, for each depth there are eight different waveforms.) Based on the locations and the values of the extreme points, linear regression is generated. This linear regression enables the analyst to find the perfect matching pattern between eight waveforms where the higher R^2 value is, the better the match it produces.

As one can observe in Figure 17, a single waveform has at least four or five distinctive extreme points with different amplitudes. Based on this observation the location finder algorithm was designed to encounter the maximum three absolute values, which appear to be the most peculiar points among all others.

In Figure 18, waveform samples belong to a specific depth is shown. Evidently, the amplitude of the waves decreases as the distance increases. This shows that the waveforms are sourced by a dipole transmitter as mentioned in Section 1.3.2.

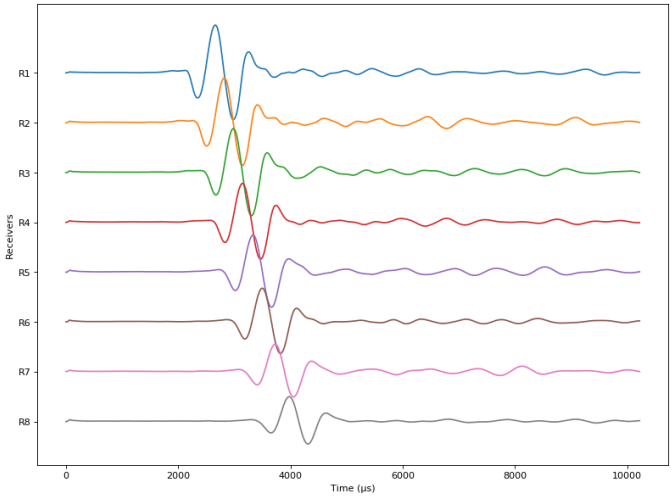

Figure 18 An Example of Waveforms at The Same Depth

In short, the algorithm finds the maximum absolute value for a single waveform. If the integer value of this value is negative, it is named as global minimum, otherwise global maximum. Whilst this point is the global maxima (or minima), it also has a splitting function for the waveform as it is desired to find the closest two maxima (or minima).

Depending on the global maxima, the waveform is divided into two waveforms in order to calculate the closest minima points (local minima). As unexpected oscillations on other sides of the waveform can occur, a sequential examination starts from the global maxima and goes towards the beginning and the ending of the waveform.

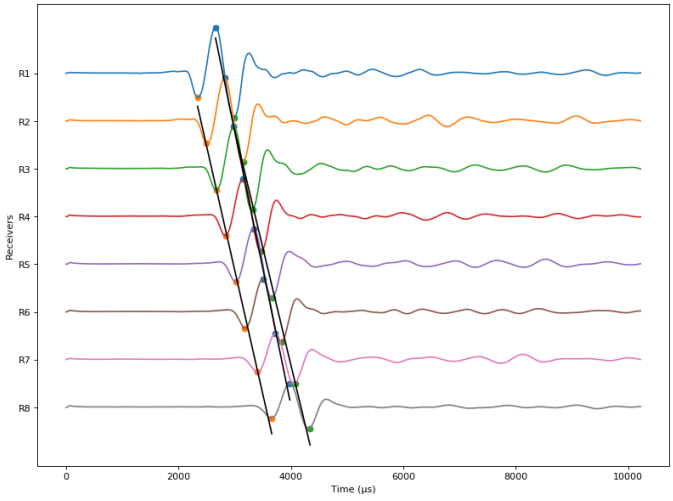

Figure 19 An Example of Waveforms after Calculating The Regression Lines

While finding local minima, the gradient of the data points is considered as mentioned in Section 2.1. The local minima are located where the direction of the gradient changes. The same process is applied to the other side of the waveform.

The location and the value information are stored in the memory for every waveform. Based on this information, it is gathered an X and Y axis information

which is eligible to create a linear regression on it. The inverse of the slope of the regression line should be equal to the desired slowness value.

Eventually, three regression lines are calculated for a single depth level (Fig. 19). In order to find the whole tomography of a borehole, this process should be repeated as many times as the number of depth levels.[24]

2.4 Problems with Global Maxima – Local Minima

Although this approach is easy to apply and effective in terms of calculation, there are associated problems. In this section, three partially related problems of this approach are discussed. Namely, the following issues have discussed the noise on waveforms, partially corrupted waveforms and completely corrupted waveforms. All three problems originate from the irregularity of waveforms. The main problem source is the shape of the waveform itself.

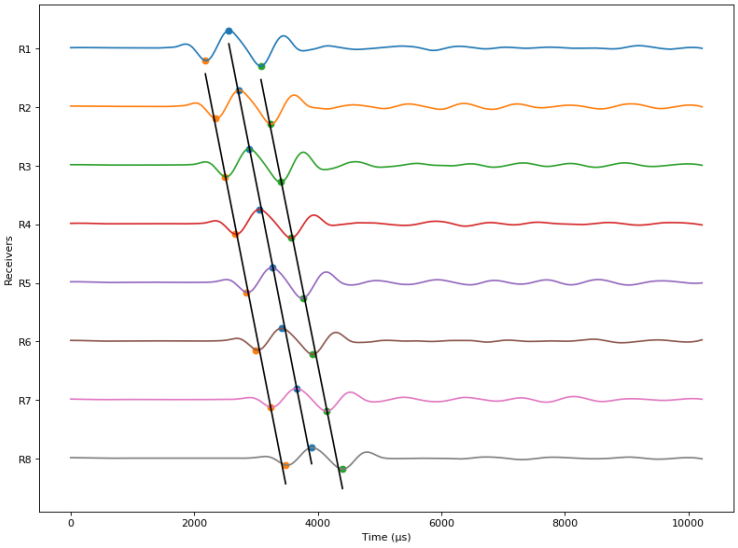

Figure 20 An Example of Filtered Waveforms

It is often observed that waveforms are noisy and that can impede the calculation of extreme points. Sometimes a peak on the outside of a Stoneley waveform may lead to miscalculations. In order to get rid of this problem, the Savitzky-Golay smoothing algorithm was used on all waveforms before running the main algorithm. As mentioned previously, window length and polyorder were assigned

[24] Please see the source code attached in APPENDIX 1: Global Maxima – Local Minima Source Code.

as 41 and 3 respectively. The Savitzky-Golay filter not only solves the noise problem of waveforms, but it also decreases the calculation time to find slowness as clearer values are sent to the central processing unit instead of distorted values. In figure 20, the same waveforms are shown with the waveforms in Figure 19 but filtered.

The second problem concerns partially corrupted waveforms. Sometimes one or two waveforms have a completely different shape than the other waveforms. As the global maxima, local minima algorithm goes over all eight waveforms to locate extreme points, having a distorted waveform leads to wrongly located extreme points for this specific waveform. Hence, this results in a wrong regression line.

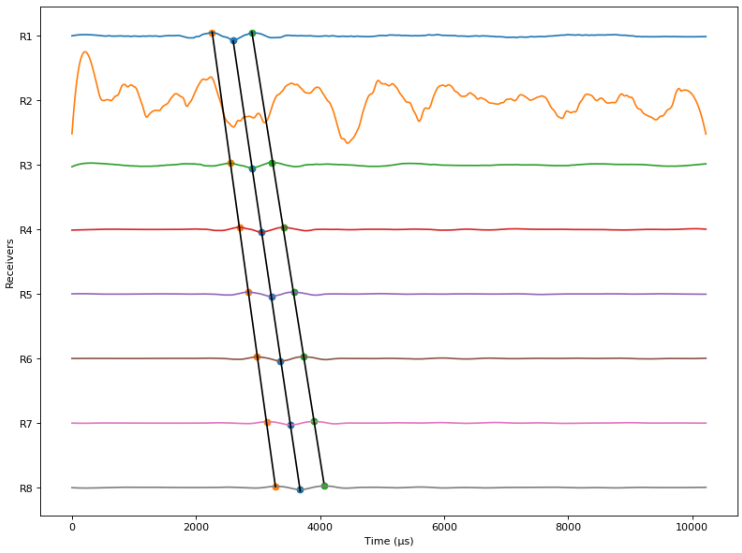

Figure 21 An Example of Partially Corrupted Waveforms

To avoid this problem, another algorithm was created to drop waveforms sequentially and check the R^2 values every time. If the R^2 value increases once a waveform drops, this waveform is signed as distorted and is excluded from the calculation. By ignoring this waveform, it is assumed that there is another waveform perfectly correlated with the others.

The last problem concerns cases where waveforms are completely corrupted. Some external forces created during the drilling process can have such effects on waveforms. Such waveforms are unnecessary for the calculation.

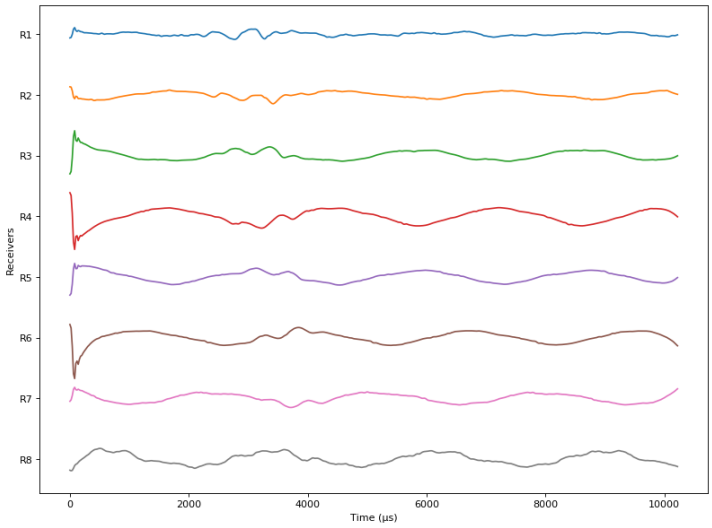

Figure 22 An Example of Completely Corrupted Waveforms

The algorithm used in this research ignores corrupted data by checking their R^2 values. Whenever a significantly low R^2 value appears during the calculation, it is printed as it is and the examination of the next depth level begins.

Chapter 3

LSTM Architecture

In this chapter, what LSTM is, its architecture, which library is used in this research and the elements of this library and eventually the problem types which LSTM can handle are explained.

3.1 Introduction to LSTM

Prior to introducing LSTM, it is useful to mention about Recurrent Neural Networks, also known as RNNs.

The development of Recurrent Neural Network architecture was based on human learning mechanisms. Early neural networks have a learning method for only the current case and whenever another case is imported they need to learn from the beginning. However, RNN architecture provides a memory strength to the network layers. For the following cases, the RNN combines the new data with past information. In Figure 23, the structure of an RNN neuron is illustrated. X is input, h is output and A is the neuron itself.

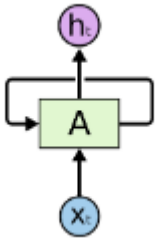

Figure 23 RNN Neuron

The shortage of RNN architecture is that it is not convenient for long term dependencies. A traditional RNN model can predict a missing value in a vector, but as the missing values increase, the RNN's become unable to learn how to

connect the information. This is why the LSTM network architecture was invented.

Long Short Term Memory networks as known as LSTM, are a special kind of RNN capable of learning long term dependencies.[25] LSTM architecture was first introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber[26]. The architecture took its last form after getting substantial updates in 1999 and 2000.

As of 2016, pioneer technology companies started to use LSTM architecture for their projects. For instance, Google deployed LSTM for speech recognition in smartphones and Google Translate.[27] Apple used the same architecture for Siri[28] and Amazon used it for Alexa.[29]

3.2 Architecture

The architecture of an LSTM neuron is completely different and more complicated than a conventional RNN neuron. Whilst an RNN neuron only consists of an activation function operator, an LSTM neuron has different paths that input information follows. These paths play a significant role in deciding which pieces of information need to be remembered and which ones need to be forgotten. An LSTM neuron contains Sigmoid and Tanh activation functions that forward significant information to the next layer appropriately. For this purpose, the neuron contains three divisions which are called gates.

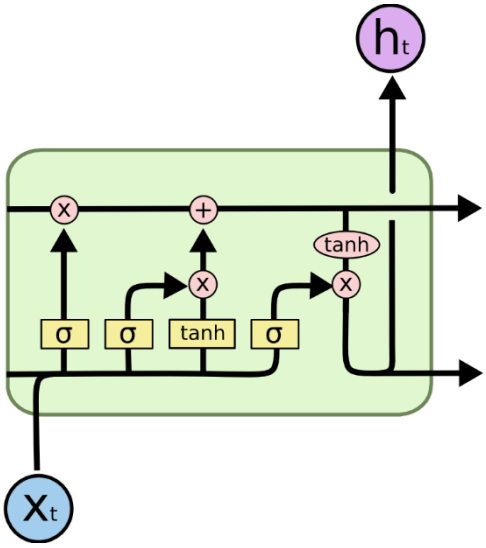

Figure 24 LSTM Neuron Architecture

[25] (Olah, 2015)
[26] (Hochreiter & Schmidhuber, 1997)
[27] (Metz, 2016)
[28] (Smith, 2016)
[29] (Vogels, 2016)

The neuron has three different gates forget gate, input gate and output gate.

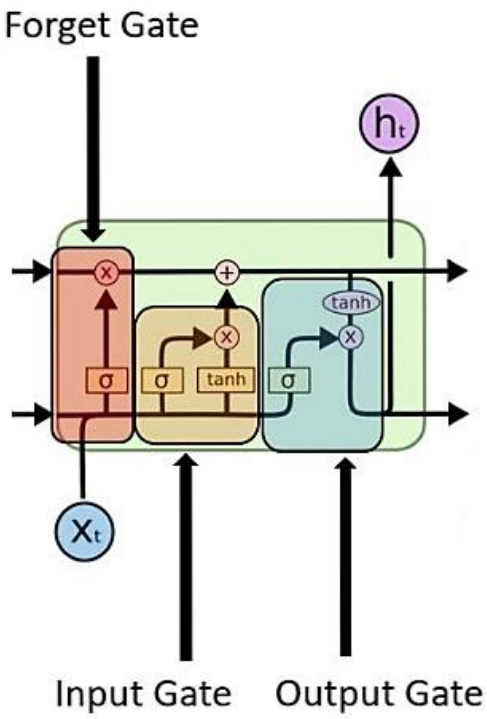

Figure 25 LSTM Gates

Forget Gate forwards the previous information towards paths one of which leads to the input gate, while the other to the memory section of the neuron.

Input Gate performs core operations. After activation functions worked, the information is copied into two paths while one goes to the memory section, the other one goes to the output gate. The information that goes to the memory section is compared with the one that comes from the forget gate and it is decided to which pieces of information will proceed to the next step.

Output Gate is where the last operation is performed. The result gained from this gate is combined with the information from the memory section and is sent to the next neuron.

Overall, information is carried to the next neuron with two copies, one generated by a logic similar to RNN and the other generated after a heavy process. The information that goes through the memory section is called long memory, while the information that goes through the input and output gate is called short memory. Once the two copies are carried to the other neuron, this neuron sends the long memory information to the memory section and begins calculating short memory operations using the short memory information just received.

3.3 TensorFlow

In this section, TensorFlow is introduced and LSTM architecture is explained with its parameters.

3.3.1 Introduction

In the early 2000s, Google began experimenting with a machine learning library for internal use. The purpose was to handle big data and interpret them more efficiently. In 2014, Google decided to turn this library into an open-source platform known as TensorFlow.

The name suggests the function of the library, which is based on deep learning algorithms. As the training process of a neural network requires an enormous amount of sample data, this sampling layer gives the third dimension of the matrices which is actually a tensor in mathematics (Fig. 26).

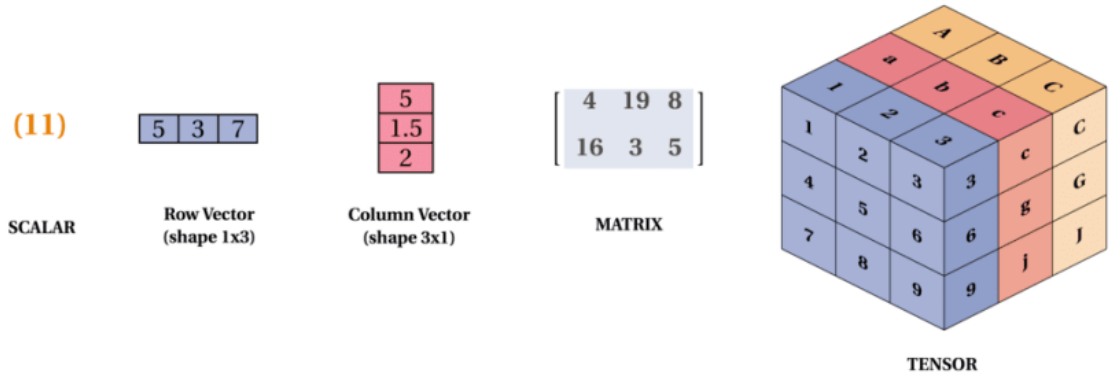

Figure 26 Scalar, Vector, Matrix, Tensor

The TensorFlow library contains a wide range of options for algorithms, optimisers, activation functions, loss functions, pooling algorithms etc. This variation brings experimental opportunities for data scientists. In this research, TensorFlow library has been used for building an LSTM model.[30]

[30] TensorFlow is used for image recognition, text mining, translation and numerous purposes by Google.

3.3.2 Layers

The TensorFlow library provides different neural network layers for different learning methods. In this section, the layers encountered in the LSTM architecture are further explained.

Two core layers are found in LSTM models. Namely, LSTM layer and Dense Layer.

The LSTM layer is the main layer that processes all tensor as explained in Section 3.3.1. By processing tensors, it creates a weight matrix and returns it.

The Dense layer is the final layer which takes the matrix comes from the previous layer and by flattering it, returns the final weight matrix. If there is no previous layer, it requires a matrix as an input shape.

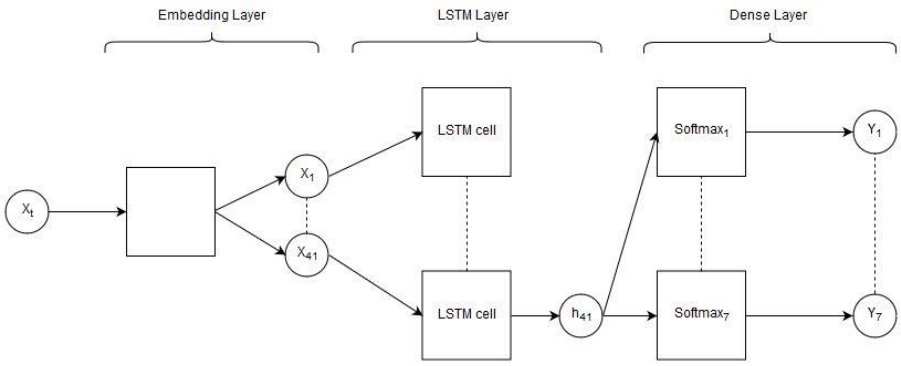

Figure 27 LSTM Layers

In Figure 27, a simple neural network with LSTM and Dense layers is illustrated.

3.3.3 Activation Functions

An activation function is an algorithm that takes place inside each neuron. It returns a decimal number, which specifies the weight of the neuron. There are many activation functions, however, the most are the following:

- Sigmoid (Logistic)
- Tanh (Hyperbolic Tangent)
- ReLu (Rectified Linear Units)

Sigmoid Activation Function: It is one of the oldest activation functions in the literature. It scales information gained between zero and one.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

If the value calculated is closer to zero, it means the neuron is less likely to get activated. Otherwise, if the value is closer to one, the neuron is more likely to get activated.

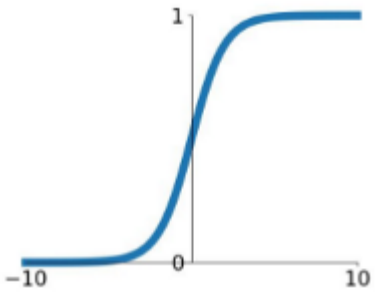

Figure 28 Sigmoid Activation Function

However, this function has weaknesses, which actually affect the learning progress of the network. In the majority of cases, it has a gradient vanishing problem. Furthermore, when the output is not zero centred, the gradient updates tend to go towards wrong directions. Finally, once it saturates, it kills the gradient as well as it converges too slowly.

Tanh Activation Function: This function attempts to correct the weaknesses of Sigmoid. Tanh squeezes the information gained between minus one and one. Hence, the optimisation becomes easier.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

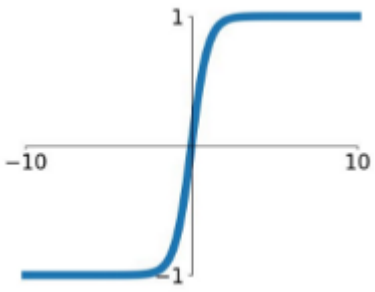

Figure 29 Tanh Activation Function

If the value calculated approaches minus one, the neuron is less likely to get activated whereas if the value approaches one, the neuron is more likely to get activated.

ReLu Activation Function: This function offers another approach to the activation functions world. Instead of squeezing the information between two numbers, it replaces negative information with zero and takes the maximum value.

$$R(x) = \max(0, x)$$

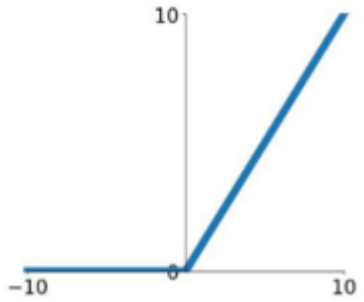

Figure 30 ReLu Activation Function

ReLu is the most popular activation function nowadays for two main reasons. The first is that it avoids and rectifies the vanishing problem encountered in Sigmoid and Tanh. The second reason is that when ReLu gives zero value for some neurons, these become passive and, as a result, the network becomes faster and more accurate. However, this function might present a significant limitation during training that can cause some of the gradients to die. Eventually, this leads to some of the neurons completely deactivated thus, the deactivated neurons lead the weight updates not to activate in the next steps.[31]

3.3.4 Loss Functions

A loss function measures the result of the algorithm training in order to compare them to the data point needed to be fitted before backpropagation. Based on the difference between the loss function result and actual observations, the model optimises itself and the process repeats.

It is worth underlining that there is no loss function that fits every algorithm in machine learning. This is why there are different types of loss functions categorized by their purposes. Mainly, loss functions can be divided into two categories: Regression Loss Functions and Classification Loss Functions. [32]

[31] (Goodfellow, Bengio, & Courville, 2016)
[32] In this research, the *Mean Squared Error* loss function was used

3.3.4.1 Regression Losses

The three following regression loss functions are the most common ones:

Mean Squared Error / Quadratic Loss / L2 Loss: This loss function measures the average of the squared difference between predictions and actual observations. As evident in the formula, it only concerns the average anomaly irrespective of its direction. Moreover, as squaring the predictions penalise the error heavily, calculating gradient descent becomes easier.

Mathematical formula:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Mean Absolute Error / L1 Loss: MAE represents as the average sum of absolute differences between the predicted values and actual observations. Just like MSE, MAE does not take into account the direction of the errors. MAE is more efficient in compute linearity and, since it does not square the values, it is more precise.

Mathematical formula:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$$

Mean Bias Error: This loss function is less common in comparison to the previous ones. It is identical to MSE with only one difference. It does not square the difference between prediction and observation. Therefore, this feature makes it sensitive to the direction of the error. As the positive and negative errors could cancel each other out, practically this method is proven to be less accurate.

Mathematical formula:

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

3.3.4.2 Classification Losses

The most common classification loss functions are as follows:

Cross-Entropy Loss / Negative Log-Likelihood: This is the most common classification loss function. The Cross-entropy loss function increases the probability of the prediction. A higher probability means that it is more likely to categorise the input correctly.

$$CrossEntropyLoss = -(y_i \, log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Hinge Loss / Multi-Class SVM Loss: This loss function is generally used for image classification. It computes the differences between class errors and eventually, a higher loss gives higher accuracy on the prediction.

Mathematical formula:

$$SVMLoss = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

3.3.5 Optimisers

An optimiser is an iterative algorithm that minimises the loss function's output on the process of neural network training. To find the local minimum, proportional steps are taken to the negative of the gradient of the function at the current point. Eventually, the global minimum (Fig. 31) is reached.

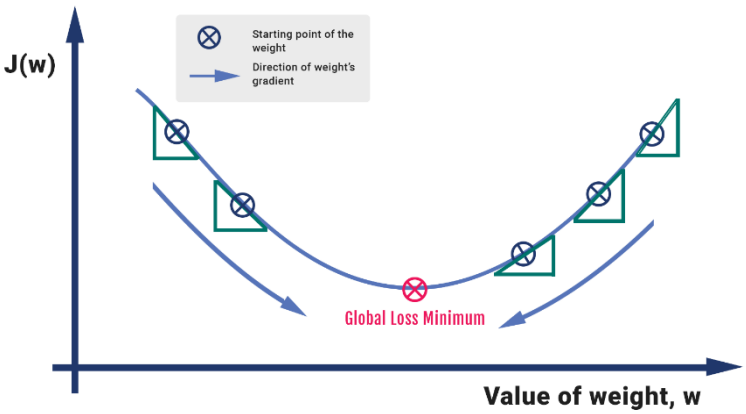

Figure 31 Gradient Descent on 2D Surface

Practically, optimisation problems are explained with three dimensions. In Figure 32, point A is the starting point of the loss function and point B refers to the global minimum which is desired to reach. By starting from A, the algorithm calculates which direction (x-y axis) brings the steepest descent in the value of the loss function. This direction is determined as the one move in and this process is repeated until the algorithm cannot find any descent in any direction which is point B.

Nowadays, there are different types of optimisers which serve for a different type of purposes. In this section, the most common optimisers are explained.

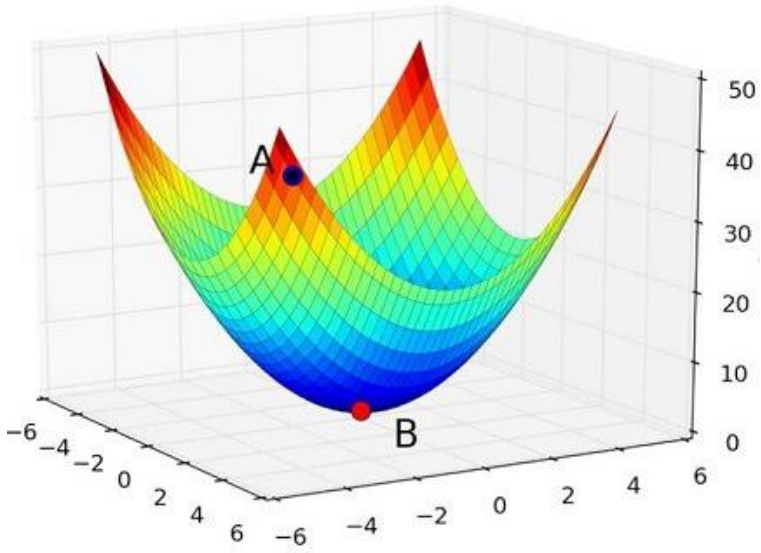

Figure 32 Gradient Descent on 3D Surface

3.3.5.1 Stochastic Gradient Descent (SGD)

It is the most popular optimisation algorithm used for neural network training. SGD is a variant of Gradient Descent and performs one update at a time so it can stay away computation redundancy.

$$\theta = \theta - \eta \cdot \nabla_\theta J\ (\theta; x^{(i)}; y^{(i)})$$

where '**η**' is the learning rate, **"∇J(θ)"** is the **Gradient** of ***Loss Function-*** **J(θ)** and $\{x^{(i)}, y^{(i)}\}$ are the training samples.

As these frequent updates, a high variance in the parameters occurs which causes the loss function to fluctuate to different intensities. This actually helps the model to discover a new and possibly better local minimum.[33]

3.3.5.2 Momentum

Momentum was developed to cover SGD's shortages. SGD has trouble navigating ravines where the surface curves much more steeply in one dimension than in another which are common around local optima.[34]

Figure 33 SGD without Momentum

Momentum is a method that accelerates SGD through a convenient direction and protects it from oscillations (Fig. 33 and Fig. 34).

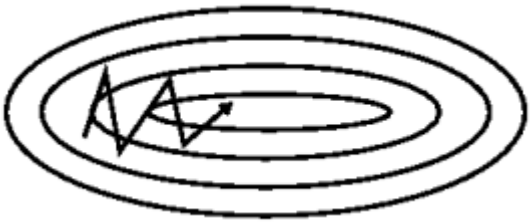

Figure 34 SGD with Momentum

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

The momentum term γ is suggested set to 0.9 or a similar value less than 1.

3.3.5.3 Nesterov Accelerated Gradient

Nesterov Accelerated Gradient as known as NAG was developed by Yurii Nesterov for the purpose of improving momentum on SGD.[35]

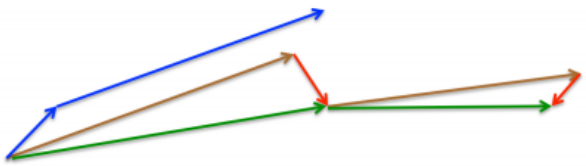

Figure 35 Nesterov Update

[33] (Walia, Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent, 2017)
[34] (Ruder, 2017)
[35] (Nesterov, 1983, pp. 372-376)

Once Momentum finds the correct direction to follow towards local minima, it only follows the gradient and sometimes passes the local minima just because of not knowing when to stop which is highly unsatisfactory. This is why Yurii Nesterov developed NAG which is a momentum that knows where to stop beforehand. Hence, a more accurate local minimum is captured.

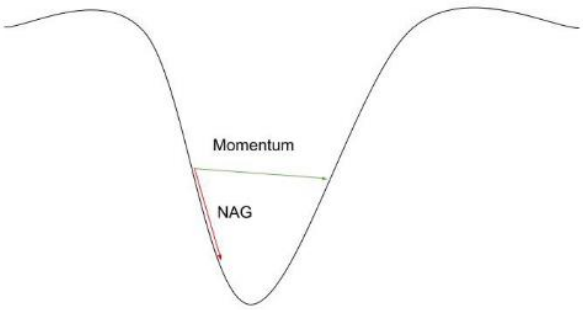

Figure 36 Momentum vs NAG

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J\,(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

3.3.5.4 Adagrad

Adagrad *(ADAptive GRADient)* is an algorithm for gradient-based optimisation that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.[36]

$$g_{t,i} = \nabla_\theta J\,(\theta_{t,i})$$
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\sum_{i=1}^{t} g_i^2 + \varepsilon}} \cdot g_{t,i}$$

where $\varepsilon = 10e - 8$

One drawback of Adagrad is learning rate becomes smaller and smaller at each step because of the accumulated gradients take place in the denominator. Hence, this may cause stopping the training earlier than it should be.

3.3.5.5 Adadelta

Adadelta is an improvement algorithm that cures the problems of Adagrad on the learning rate. Adagrad reduces the aggressive and regular decreasing on learning rate by restricting the window of accumulated gradients to a fixed size ω.

[36] (Ruder, 2017)

In Adadelta, the sum of gradients is recursively defined as a decaying average of all past squared gradients. [37]

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

where RMS stands for *Root Mean Squared error criterion of the gradient.*

3.3.5.6 RMSprop

RMSprop and Adadelta were introduced almost at the same period with the same purpose of handling the diminishing learning rate of Adagrad. While Adagrad was published, RMSprop was not published but introduced as an alternative learning rate method by Geoff Hinton.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

Momentum γ is suggested to be *0.9* while the default learning rate η is 0.001.

3.3.5.7 Adam

Adaptive Moment Estimation (Adam) is another optimiser which computes an adaptive learning rate for each parameter. Adam keeps an exponentially decaying average of past gradients $M_{(t)}$ besides storing an exponentially decaying average of past squared gradients.

$M_{(t)}$ and $V_{(t)}$ values stand for ***Mean*** *(first moment)* and ***uncentered variance*** *(second moment)* respectively.[38]

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

[37] (Ruder, 2017)
[38] (Walia, Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent, 2017)

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Lastly, the parameter update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

Where the values for β_1 is 0.9, for β_2 is 0.999 and for ε is $10e - 8$.

In this research, Adam optimiser was used.

3.4 Problem Types

Broadly, there are two problem types of deep learning methods. Classification problems and regression problems.

Classification problems are based on a number of classes; the input data is desired to be classified. After training, the model gets ready to calculate the probability of the new inputs. Based on the new input, the model measures the likelihood of the input to be one of the pre-defined classes. The output vector matches the number of classes and the highest valued element of the vector defines the probability hence, the input matches with the corresponding class.

Image recognition, hand recognition, binary classifications (spam, not spam) can be considered types of classification problems.

Regression problems are basically fitting problems. Based on the input, it is desired model to predict a continuous value between a range. The continuous output value is a real-value which can be an integer or a decimal number.

In this research, a regression problem was examined and an LSTM model was built according to this problem type.

Chapter 4

Data Analysis and LSTM Implementation

In this chapter, the problem was tried to be solved in Chapter 2, is examined but with a deep learning method, LSTM neural network. After explaining the data structure briefly, what has been done on the pre-processing stage, model building, training steps are explained and lastly, the results are illustrated.

4.1 Data Structure

As mentioned in Section 1.7, a dataset contains six well's sonic logs, was provided by Weatherford. Each well's sonic log file contains one column for depth and 16384 columns for sonic log data which refer eight different receivers with four different directions and each soundwave comprises of 512 data points. The number of rows ranged from 2000 to 4500.

Four receivers A, B, C and D and symmetrically located on the other's side. Two dipole X and Y were located orthogonally while A and C constitute the X-Pole, B and D constitute the Y-Pole.

4.2 Pre-processing

Firstly, after the data was read from the CSV file, undesired rows were cleaned and the rows in the training input were matched with training output according to the depth values.

Each dipole was processed separately. As symmetric receivers contain mostly same compressional and shear waveforms, in order to extract plain Stoneley waveforms, C values subtracted from A values and D values subtracted from B values. Hence, new waveforms were obtained (Fig. 37).

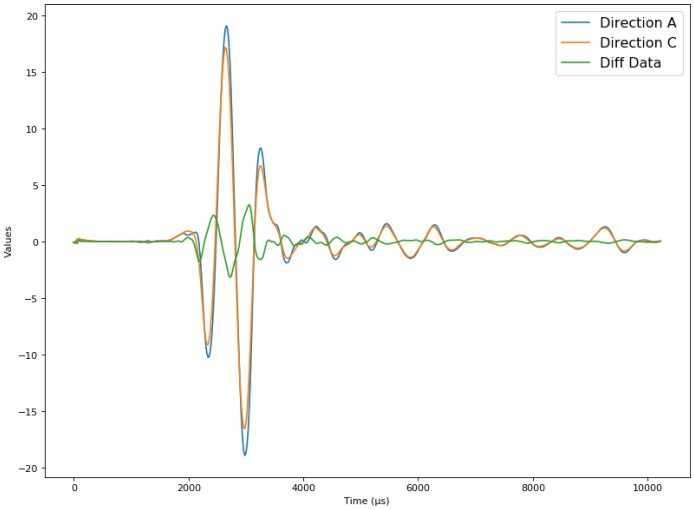

Figure 37 Subtraction of C and A Directions and New Waveform

In order to train the model all over six boreholes, the data sets were merged after subtraction.

After the cleaning process was completed, the data was imported to NumPy arrays in order to handle it easily and quickly. While importing the data, Savitzky-Golay smoothing algorithm was applied over the data.

For the purpose of increasing the efficiency of both loss function and optimiser, input data was scaled between -1 and 1 while the observation data were scaled between 0 and 1.

Last but not least, the shape of the array was changed as the LSTM architecture requires a special type of input shape which is a tensor. LSTM neural networks

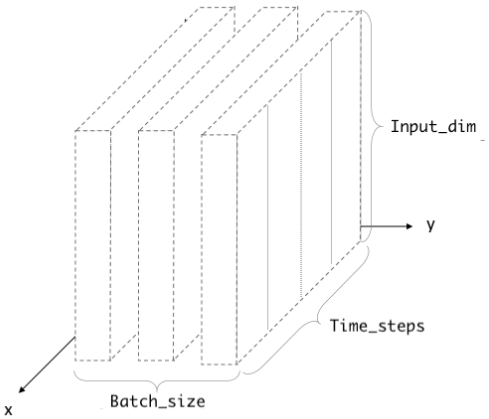

Figure 38 LSTM Input Shape

require three-dimensional input as they were generally developed for image recognition. Every image is actually a matrix and the sample size becomes the third dimension, the same procedure was applied in this research. As for every depth level, having 8 waveforms becomes an 8 by 512 matrix. Hence, while 8 by 512 matrices were the two dimension of the tensor, the size of the samples was assigned as the third dimension.

4.3 Building The Model

On the model building stage, it is benefitted from TensorFlow and Keras libraries. Following the LSTM layer which had 512 neurons, the next three dense layers have 512, 100 and 50 neurons respectively. The last dense layer has only one output.

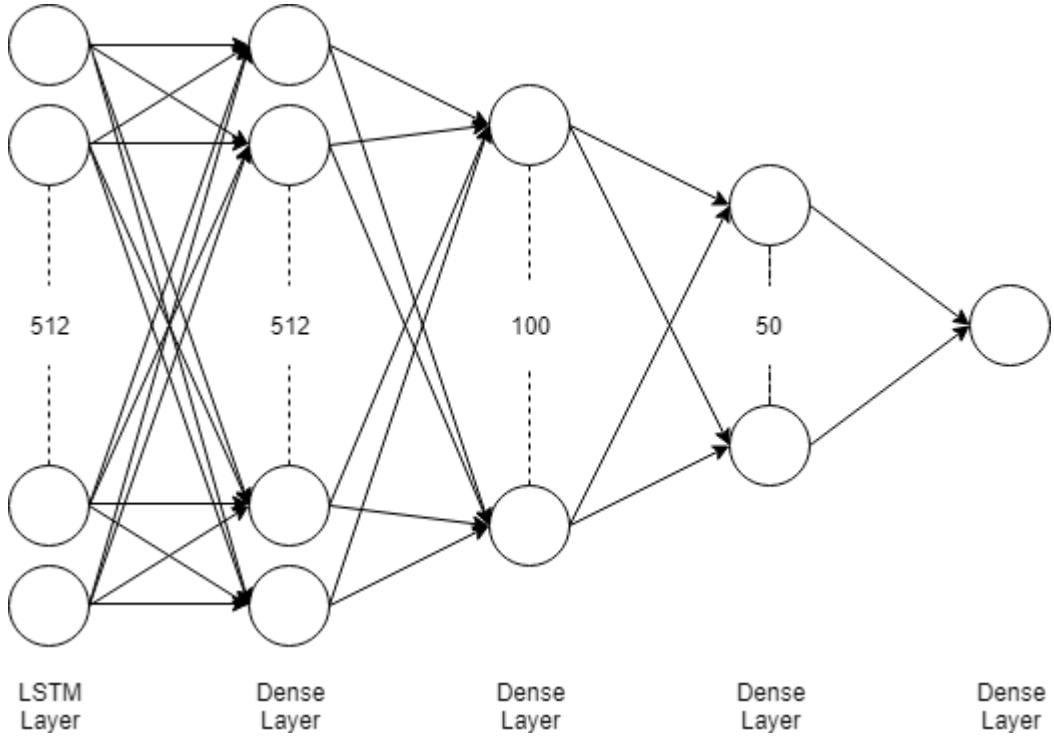

Figure 39 Model Structure

In every layer, *ReLu* is used as an activation function. *Mean Squared Error* is chosen as loss function and *Adam* as the optimiser.

Keras' EarlyStopping feature was used in order to avoid overfitting and minimum delta was set $1e-4$ and patience was set 5. These parameters set the trigger of continuing five more steps and stop training if there is no improvement greater than $1e-4$. EarlyStopping was set to monitor *Validation Loss*.

Lastly, the validation split was set as 50%.[39]

4.4 Training and Results

Three different training methods were used.

The first training of the model, trained only one model. Based on this training, it was examined that if the model can predict the other well's output. However, it was not successful. Two graphs below show the inconsistency between the borehole predictions.

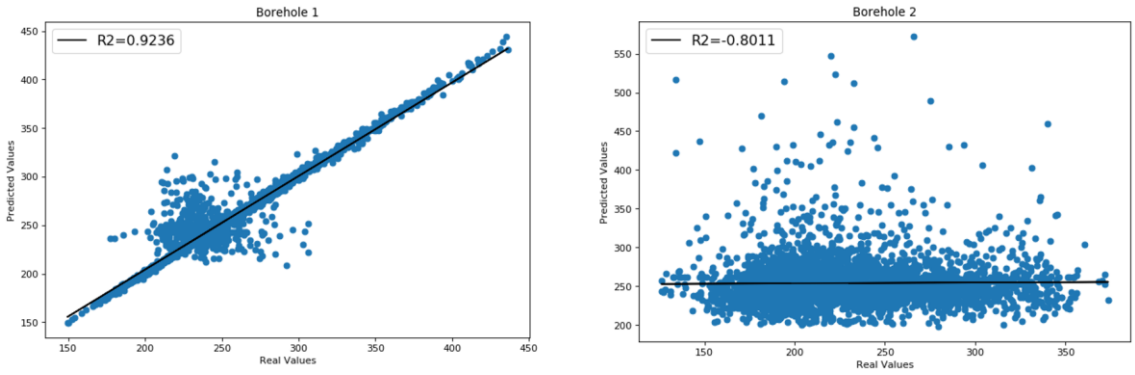

Figure 40 Training 1, Prediction Comparison

The model was trained with the data of *Borehole 1*. This is why it has a 92% explanatory power on this well whereas the same model (weight matrix) has no efficient prediction on *Borehole 2* data.

On the second training, the model trained on well's log data in order by starting with *Borehole1* and ending with *Borehole6*. It was observed that the results were not that much different than the first training way as the model was trained on *Borehole 6* as last and it was fit itself to *Borehole 6* dataset. This is why, with this

[39] Please check Appendix 2 for further detail.

training, the model was able to predict *Borehole 6* dataset perfectly but the other data sets' predictions were not that much stable.

Finally, a shuffling algorithm was written for the purpose of randomising the training data. So, the model training would not only stick on a specific well's data set. Among 25592 sequential samples, 25592 samples were selected but by randomising and the model was trained with 50% validation split. The results were significantly accurate and stable.

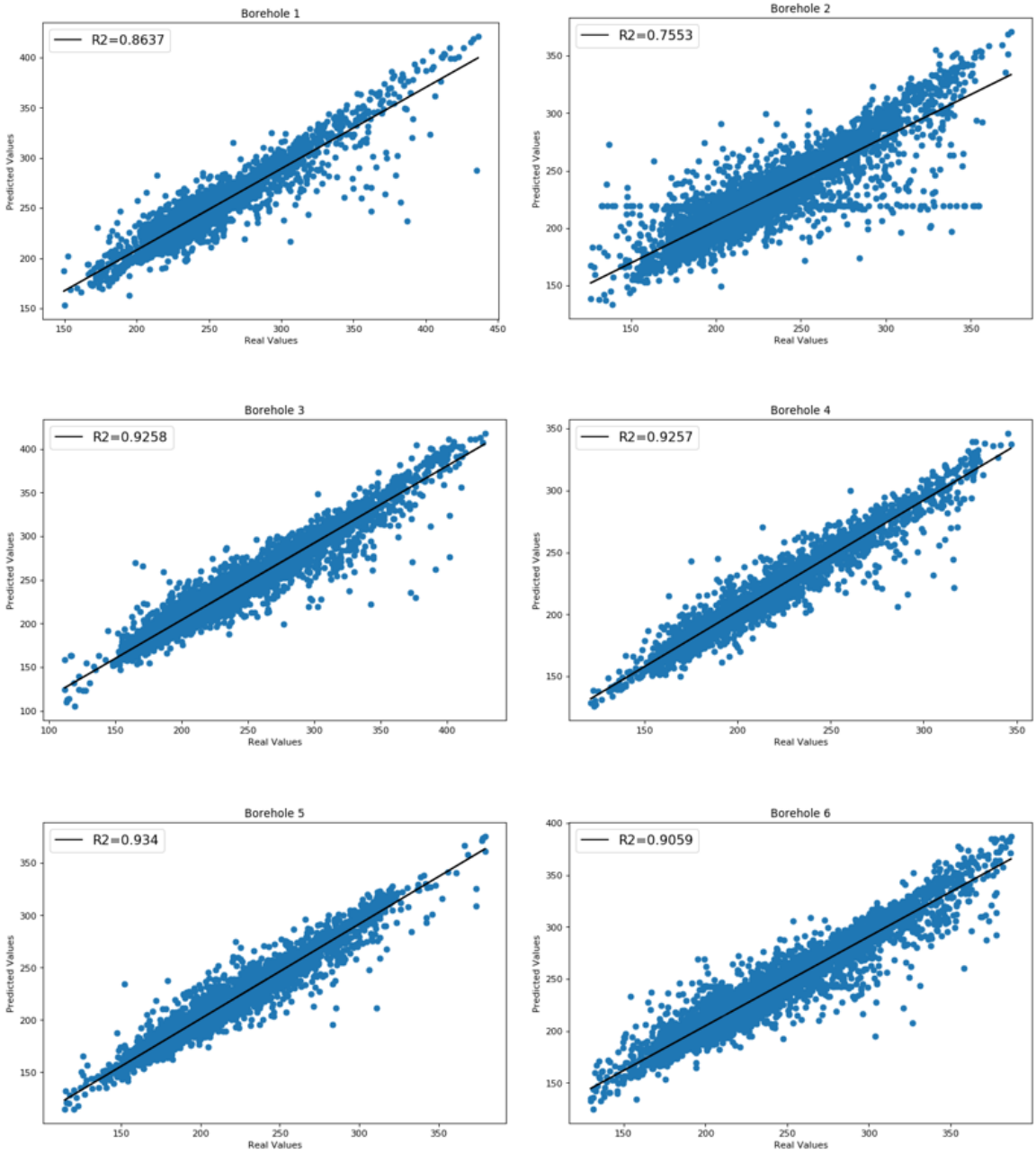

Figure 41 Accuracy of Boreholes

Additionally, the model was tested with two log data which are significantly different than the training set's output range. Figure 42 shows there is no relation between the observations and the predicted values which is expected behaviour as these sets were the sets that the model has never seen before.[40]

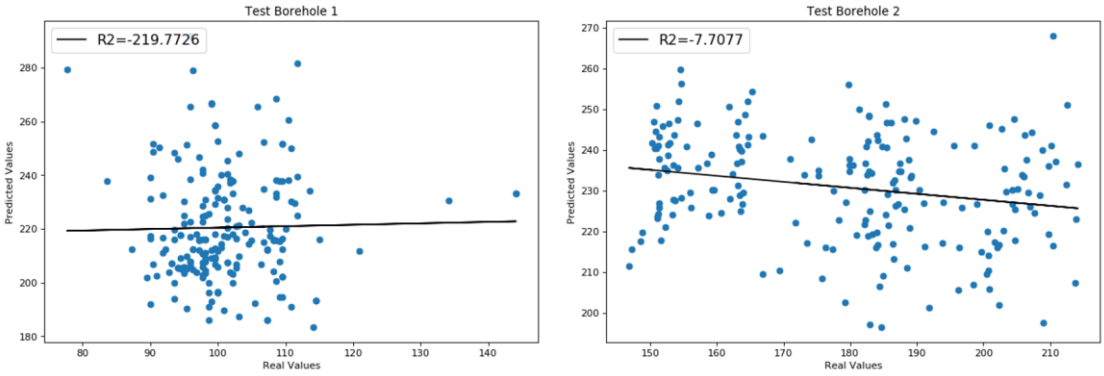

Figure 42 Model Testing Results

[40] Note that: The observation range that the model was trained on had a range from 120 to 460.

Chapter 5

Business Contribution

Weatherford International is one of the largest multinational oil and natural gas service companies. The company has a broad range of services from equipment providing to well analysis. Also, Weatherford has its own productions.

As the focal point of this research is well log analysis, the analysis part will be examined. Weatherford has branches over 90 countries, 30000 employees (by 2016) and hundreds of log analysts.

A log analyst examines the different types of measurements such as gamma-ray, x-ray, sonic wave logs and draws the tomography of the well. The field workers drill the well, collect the logs and send them to the analyst to get the final results. This conventional business cycle has been running for a long time with decent reliability. However, this business cycle comes accompanied by some problems.

Based on the talks made with Weatherford employees, a qualified and well-experienced analyst requires at least 5 to 6 hours to finish the sonic analysis of a well. This can extend through days as the experience level of the analyst decreases. Moreover, different analysts draw slightly different results as different methods are used to examine the data. Even same analyst may extract the results with different deviations. Additionally, the company puts a disclaimer for the accuracy of the results with ±10%. These facts show that human factor carries an operational risk inside the business cycle.

The model was developed during this research is able to give the same well's results in less than 30 seconds with 90% accuracy. Furthermore, centralisation of the log interpretation decreases the operational risk while increasing the overall accuracy. Considering the size of the company, these small factors are vital because of the accumulation effect of the factors.

Chapter 6

Conclusion

Throughout this research, it was attempted to explain sonic logging history, applications, LSTM architecture and eventually to find an alternative faster and more reliable solution to the conventional calculations by using two different methods which were Global Maxima – Local Minima approach and Deep Learning Implementation with LSTM Architecture.

The Global Maxima – Local Minima approach did not work due to the unstability of the waveforms in the datasets even though filtering algorithms were used.

When the improvements were observed on LSTM Neural Network as the parameters were changed, it proved itself as it was a promising solution for the problem. Yet, the model resulted in wrongly when a different type of data provided as input which was the limitation of this approach.

To summarise, it was observed that the LSTM networks are feasible on acoustic wave processing only if the model has been trained on specific data before. Otherwise, the prediction power of the model gets significantly weak. This also showed that the more the neural network is got trained, the more accurate results are gained.

The model has only trained over 25592 samples. Undoubtedly amount of samples is relatively low considering the complexity of the matrices and it was desired to solve a regression problem.

For further studies, a custom loss function was suggested which could avoid calculation any mean. Also, optimum learning ratio could be determined by another study so gradient descent becomes more efficient.

Finally, it was observed that LSTM Neural Networks are capable to replace conventional analysts.

Bibliography

[1]. Akhundi, H., Ghafoori, M., & Lashkaripour, G.-R. (2014). *Prediction of Shear Wave Velocity Using Artificial Neural Network Technique, Multiple Regression and Petrophysical Data: A Case Study in Asmari Reservoir (SW Iran)*. Mashhad, Iran.

[2]. An, P., Yang, X., & Zhang, M. (2018). *Porosity prediction and application with multiwell-logging curves based on deep neural network*. California: Society of Exploration Geophysicists.

[3]. Archie, G. (1942). The Electrical Resistivity Log as an Aid in Determining Some Reservoir Characteristics. *Society of Petroleum Engineers*.

[4]. Brownlee, J. (2017). *Long Short-Term Memory Networks With Python*.

[5]. Chen, J., & Zeng, Y. (2018). *Application of Machine Learning in Rock Facies Classification with Physics-Motivated Feature*.

[6]. Close, D., Cho, D., Horn, F., & Edmundson, H. (2009). The Sound of Sonic: A Historical Perspective and Introduction to Acoustic Logging. *CSEG*.

[7]. Crain, E. (2004). How Many Acoustic Waves Can Dance On The Head Of A Sonic Log? *Canadian Well Logging Society*, 11.

[8]. Crain, E. (2004). How Many Acoustic Waves Can Dance On The Head Of A Sonic Log? *Canadian Well Logging Society*, 10.

[9]. Dahai, W., Jun, P., Qian, Y., Yuanyuan, C., & Hanghang, Y. (2019). *Support Vector Machine Algorithm for Automatically Identifying Depositional Microfacies Using Well Logs*. Chengdu: College of Earth Science and Technology.

[10]. Glover, D. P. (n.d.). The Sonic or Acoustic Log. In D. P. Glover. Leeds.

[11]. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[12]. Haldorsen, J., Johnson, D., Plona, T., Sinha, b., Valero, H.-P., & Winkler, K. (2006). *Borehole Acoustic Waves*. Ridgefield, Connecticut, USA.

[13]. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 1735-1780.

[14]. Izurieta, C., Rocha, L., & Sui, D. (2019). *An Approach in Caving Recognition by An Integrated Model of Computer Vision and Machine Learning for Any Drilling Environment.*

[15]. Jianhua, C., Yancui, S., Dan, W., & Xiankun, Z. (2017). Acoustic Log Prediction on the Basis of Kernel Extreme Learning Machine for Wells in GJH Survey. *Journal of Electrical and Computer Engineering*, 7.

[16]. Konate, A. A., Khalid, M. A., Pan, H., & Li, G. (2015). *Machine Learning Interpretation of Conventional Well Logs in Crystalline Rocks.* Beijing.

[17]. MathCentre. (2009). *Minima and Maxima.*

[18]. Metz, C. (2016). *An Infusion of AI Makes Google Translate More Powerful Than Ever.*

[19]. Nesterov, Y. (1983). A Method of Solving A Convex Programming Problem with Convergence Rate. *Soviet Mathematics Doklady*, 372-376.

[20]. Olah, C. (2015). *Understanding LSTM Networks.* California.

[21]. Pham, N., & Naeini, E. (2019). *Missing well log prediction using deep recurrent neural networks.* EAGE.

[22]. Ruder, S. (2017). *An overview of gradient descent optimization.* Dublin: Insight Centre for Data Analytics, NUI Galway.

[23]. Schlumberger, C., & Schlumberger, M. (1932). Electrical Coring: A Method of Determining Bottom-Hole Data by Electrical Measurements. *American Institute of Mining and Metallurgical Engineers.* New York.

[24]. Smith, C. (2016). *iOS 10: Siri now works in third-party apps, comes with extra AI features.*

[25]. Stewart, J. (2008). *Calculus: Early Transcendentals.*

[26]. Tariq, Z., Elkatatny, S., Mahmoud, M., & Abdulraheem, A. (2016). *A New Artificial Intelligence Based Empirical Correlation to Predict Sonic Travel Time.* Bangkok: International Petroleum Technology Conference.

[27]. Vogels, W. (2016). *Bringing the Magic of Amazon AI and Alexa to Apps on AWS. - All Things Distributed.*

[28]. Walia, A. S. (2017). *Activation Functions and It's Types - Which Is Better?* Towards Data Science.

[29]. Walia, A. S. (2017). *Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent.* Towards Data Science.

[30]. Zemanek, J. (May 1991). *Shear Wave Logging Using Multipole Sources.* The Log Analyst.

[31]. Zhang, D., Chen, Y., & Meng, J. (2018). *Synthetic well logs generation via Recurrent Neural Networks.*

APPENDIX 1: Global Maxima – Local Minima Source Code

Reading CSV

```
import pandas as pd
df = pd.read_csv(r'C:\Users\~\FileName.csv')
print(df)
```

Cleaning The Data

```
df1 = df[(df["XR1A[1]"]!=0) & (df["XR1A[1]"]!=-999.25)]
df1.reset_index(drop=True, inplace=True)

print(df1)
```

Finding The Max-Min and Linear Regression

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.pyplot import figure
import scipy
from scipy import signal
figure(num=None, figsize=(12, 9), dpi=80, facecolor='w', edgecolor='k')
depth = 0
samplingFrequency = 20
shift = 20 *3.28084 # feet conversion
shiftMult = 0

# Slope Calculation Function
def slope(x1,x2,y1,y2):
    m = (y2-y1)/(x2-x1)
    return m

# Axis ticks and tick names
xAxis = np.linspace(1,512*20,512)
xAxis = np.arange(0, 512*samplingFrequency, 20)
yAxis = []
y_ticks = ["R1", "R2", "R3", "R4", "R5", "R6", "R7", "R8"]

# Singular Savitzky-Golay Filter
savgol = signal.savgol_filter(df1.loc[depth,"XR"+str(1)+"A[1]":"XR"+str
(1)+"A[512]"], \
                              axis=0, window_length=41, polyorder=3)

plt.plot(xAxis, df1.loc[depth,"XR"+str(1)+"A[1]":"XR"+str(1)+"A[512]"],
label="Raw Data")
```

```
plt.plot(xAxis, savgol, label="Filtered Data")
plt.legend()
plt.ylabel('Values')
plt.xlabel('Time (µs)')
plt.show()


# Max, Min1 and Min2 Identification
maxsVal = []
mins1Val = []
mins2Val = []

maxsLoc = []
mins1Loc = []
mins2Loc = []


figure(num=None, figsize=(12, 9), dpi=80, facecolor='w', edgecolor='k')

# Raw Data - Location and Value Determination for Extreme Points
for i in range(8):

    y = df1.loc[depth,"XR"+str(i+1)+"A[1]":"XR"+str(i+1)+"A[512]"]

    # Max1
    max1 = np.max(y)
    max1Loc = np.where(y == np.amax(y))[0][0]
    maxsVal.append(max1-shift*shiftMult)
    maxsLoc.append(max1Loc*samplingFrequency)

    # Min1
    for j in range(510):
        m1 = slope(max1Loc-j, max1Loc-j-1, y[max1Loc-j], y[max1Loc-j-1]
)
        m2 = slope(max1Loc-j-1, max1Loc-j-2, y[max1Loc-j-1], y[max1Loc-
j-2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min1 = y[max1Loc-j-1]
            min1Loc = max1Loc-j-1
            break
    mins1Val.append(min1-shift*shiftMult)
    mins1Loc.append(min1Loc*samplingFrequency)

    # Min2
    for k in range(510):
        m1 = slope(max1Loc+k, max1Loc+k+1, y[max1Loc+k], y[max1Loc+k+1]
)
        m2 = slope(max1Loc+k+1, max1Loc+k+2, y[max1Loc+k+1], y[max1Loc+
k+2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min2 = y[max1Loc+k+1]
            min2Loc = k+1
            break
    mins2Val.append(min2-shift*shiftMult)
    mins2Loc.append(min2Loc*samplingFrequency + max1Loc*samplingFrequen
cy)
```

```
    plt.plot(xAxis, df1.loc[depth,"XR"+str(i+1)+"A[1]":"XR"+str(i+1)+"A
[512]"]-shift*shiftMult)
    shiftMult = shiftMult + 1



# Filtered Data  - Location and Value Determination for Extreme Points
for i in range(8):

    filtered = signal.savgol_filter(df1.loc[depth,"XR"+str(i+1)+"A[1]":
"XR"+str(i+1)+"A[512]"], \
                                    axis=0, window_length=51, polyorder
=3)

    # Max1
    max1 = np.max(filtered)
    max1Loc = np.where(filtered == np.amax(filtered))[0][0]
    maxsVal.append(max1-shift*shiftMult)
    maxsLoc.append(max1Loc*samplingFrequency)

    # Min1
    for j in range(510):
        m1 = slope(max1Loc-j, max1Loc-j-1, filtered[max1Loc-j], filtere
d[max1Loc-j-1])
        m2 = slope(max1Loc-j-1, max1Loc-j-2, filtered[max1Loc-j-1], fil
tered[max1Loc-j-2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min1 = filtered[max1Loc-j-1]
            min1Loc = max1Loc-j-1
            break
    mins1Val.append(min1-shift*shiftMult)
    mins1Loc.append(int(min1Loc*samplingFrequency))

    for k in range(510):
        m1 = slope(max1Loc+k, max1Loc+k+1, filtered[max1Loc+k], filtere
d[max1Loc+k+1])
        m2 = slope(max1Loc+k+1, max1Loc+k+2, filtered[max1Loc+k+1], fil
tered[max1Loc+k+2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min2 = filtered[max1Loc+k+1]
            min2Loc = k+1
            break
    mins2Val.append(min2-shift*shiftMult)
    mins2Loc.append(int(min2Loc*samplingFrequency + max1Loc*samplingFre
quency))

    yAxis.append((filtered-shift*shiftMult)[0])
    plt.plot(xAxis, filtered-shift*shiftMult)
    shiftMult = shiftMult + 1


import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
```

```
from sklearn import metrics

# Linear Regression starts based on the locations and values of the ext
reme points

# MAXS
Xmaxs = np.array(maxsLoc).reshape(-1,1) # values converts it into a num
py array
Ymaxs = np.array(maxsVal).reshape(-1,1)  # -1 means that calculate the
dimension of rows, but have 1 column
linear_regressor_maxs = LinearRegression()  # create object for the cla
ss
linear_regressor_maxs.fit(Xmaxs, Ymaxs)  # perform linear regression
Y_pred_maxs = linear_regressor_maxs.predict(Xmaxs)  # make predictions

print("Maxs coef", linear_regressor_maxs.coef_, metrics.r2_score(Ymaxs,
Y_pred_maxs))

plt.scatter(Xmaxs, Ymaxs)
plt.plot(Xmaxs, Y_pred_maxs, color='black')


# MINS1
Xmins1 = np.array(mins1Loc).reshape(-1,1) # values converts it into a n
umpy array
Ymins1 = np.array(mins1Val).reshape(-1,1)  # -1 means that calculate th
e dimension of rows, but have 1 column
linear_regressor_mins1 = LinearRegression()  # create object for the cl
ass
linear_regressor_mins1.fit(Xmins1, Ymins1)  # perform linear regression
Y_pred_mins1 = linear_regressor_mins1.predict(Xmins1)  # make predictio
ns

print("Mins1 coef", linear_regressor_mins1.coef_, metrics.r2_score(Ymin
s1, Y_pred_mins1))

plt.scatter(Xmins1, Ymins1)
plt.plot(Xmins1, Y_pred_mins1, color='black')


# MINS2
Xmins2 = np.array(mins2Loc).reshape(-1,1) # values converts it into a n
umpy array
Ymins2 = np.array(mins2Val).reshape(-1,1)  # -1 means that calculate th
e dimension of rows, but have 1 column
linear_regressor_mins2 = LinearRegression()  # create object for the cl
ass
linear_regressor_mins2.fit(Xmins2, Ymins2)  # perform linear regression
Y_pred_mins2 = linear_regressor_mins2.predict(Xmins2)  # make predictio
ns

print("Mins2 coef", linear_regressor_mins2.coef_, metrics.r2_score(Ymin
s2, Y_pred_mins2))

plt.scatter(Xmins2, Ymins2)
plt.plot(Xmins2, Y_pred_mins2, color='black')

plt.ylabel('Receivers')
```

```
plt.xlabel('Time (µs)')
plt.yticks(yAxis, y_ticks)
plt.title("X-Pole Sonic Logs at the depth of "+str(dfDiff.loc[depth,"DE
PTH"]))
plt.show()


print("Maxs", maxsLoc)
print("Maxs", maxsVal)

print("mins1", mins1Loc)
print("mins1", mins1Val)

print("mins2", mins2Loc)
print("mins2", mins2Val)
```

Difference C - A

```
import numpy as np

columnNames = []

xAxis = np.linspace(1,512,512)
for r in range(8):
    for i in range(512):
        columnNames.append(str(r+1) + "_" + str(i+1))



dfReceiverA = df1.loc[:,df1.columns.str.contains('A')]
dfReceiverA.columns = columnNames

dfReceiverC = df1.loc[:,df1.columns.str.contains('C')]
dfReceiverC.columns = columnNames

dfDiff = dfReceiverC.subtract(dfReceiverA)
dfDiff.insert(0, column = "DEPTH", value = df1.iloc[:,0:1]) #Inserting
df1's DEPTH values
print(dfDiff)
```

Diff Linear Regression C - A

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.pyplot import figure
import scipy
from scipy import signal
figure(num=None, figsize=(12, 9), dpi=80, facecolor='w', edgecolor='k')
# depth = 193
depth = 0
samplingFrequency = 20
shift = 0.20 * 3.2808399 # feet conversion
shiftMult = 0

# Savgol Filter Parameters
winLen = 41
```

```
polyOrd = 3


# Slope Calculation Function
def slope(x1,x2,y1,y2):
    m = (y2-y1)/(x2-x1)
    return m


xAxis = np.linspace(1,512*20,512)
xAxis = np.arange(0, 512*samplingFrequency, 20)
yAxis = []
y_ticks = ["R1", "R2", "R3", "R4", "R5", "R6", "R7", "R8"]

savgol = signal.savgol_filter(dfDiff.loc[depth, str(1)+"_1": str(1)+"_5
12"], \
                              axis=0, window_length=winLen, polyorder=p
olyOrd)

plt.plot(xAxis, dfDiff.loc[depth, str(1)+"_1": str(1)+"_512"], label="U
nfiltered")
plt.plot(xAxis, savgol, label="Filtered")
plt.legend(loc="best")
plt.xlabel("Time (Hz)")
plt.title("R1 Sonic Logs at the depth of "+str(dfDiff.loc[depth,"DEPTH"
]))
plt.show()


# Max, Min1 and Min2 Identification
maxsVal = []
mins1Val = []
mins2Val = []

maxsLoc = []
mins1Loc = []
mins2Loc = []


figure(num=None, figsize=(12, 9), dpi=80, facecolor='w', edgecolor='k')


# Raw Data

yFirstLine = dfDiff.loc[depth,"1_1":"1_512"]

maxN = np.max(yFirstLine)
minN = abs(np.min(yFirstLine))

for i in range(8):

    y = dfDiff.loc[depth,str(i+1)+"_1":str(i+1)+"_512"]

    # Max1
    if maxN >= minN:
        print("Global Maximum Worked")
        max1 = np.max(y)
        max1Loc = np.where(y == np.amax(y))[0][0]
```

```
    else:
        print("Global Minimum Worked")
        max1 = np.min(y)
        max1Loc = np.where(y == np.amin(y))[0][0]

    maxsVal.append(max1-shift*shiftMult)
    maxsLoc.append(max1Loc*samplingFrequency)

    # Min1
    for j in range(510):
        m1 = slope(max1Loc-j, max1Loc-j-1, y[max1Loc-j], y[max1Loc-j-1]
)
        m2 = slope(max1Loc-j-1, max1Loc-j-2, y[max1Loc-j-1], y[max1Loc-
j-2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min1 = y[max1Loc-j-1]
            min1Loc = max1Loc-j-1
            break
    mins1Val.append(min1-shift*shiftMult)
    mins1Loc.append(min1Loc*samplingFrequency)



    # Min2
    for k in range(510):
        m1 = slope(max1Loc+k, max1Loc+k+1, y[max1Loc+k], y[max1Loc+k+1]
)
        m2 = slope(max1Loc+k+1, max1Loc+k+2, y[max1Loc+k+1], y[max1Loc+
k+2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min2 = y[max1Loc+k+1]
            min2Loc = k+1
            break
    mins2Val.append(min2-shift*shiftMult)
    mins2Loc.append(min2Loc*samplingFrequency + max1Loc*samplingFrequen
cy)

    plt.plot(xAxis, dfDiff.loc[depth,str(i+1)+"_1":str(i+1)+"_512"]-shi
ft*shiftMult)
    shiftMult = shiftMult + 1


# Filtered Data



filtered1stLine = signal.savgol_filter(dfDiff.loc[depth,"1_1":"1_512"],
\
                                       axis=0, window_length=winLen, polyo
rder=polyOrd)

maxN = np.max(filtered1stLine)
minN = abs(np.min(filtered1stLine))
```

```
for i in range(8):
    filtered = signal.savgol_filter(dfDiff.loc[depth,str(i+1)+"_1":str(
i+1)+"_512"], \
                                    axis=0, window_length=winLen, polyo
rder=polyOrd)

    # Max1
    if maxN >= minN:
        print("Global Maximum Worked")
        max1 = np.max(filtered)
        max1Loc = np.where(filtered == np.amax(filtered))[0][0]

    else:
        print("Global Minimum Worked")
        max1 = np.min(filtered)
        max1Loc = np.where(filtered == np.amin(filtered))[0][0]

    maxsVal.append(max1-shift*shiftMult)
    maxsLoc.append(int(max1Loc*samplingFrequency))

    # Min1
    for j in range(510):
        m1 = slope(max1Loc-j, max1Loc-j-1, filtered[max1Loc-j], filtere
d[max1Loc-j-1])
        m2 = slope(max1Loc-j-1, max1Loc-j-2, filtered[max1Loc-j-1], fil
tered[max1Loc-j-2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min1 = filtered[max1Loc-j-1]
            min1Loc = max1Loc-j-1
            break
    mins1Val.append(min1-shift*shiftMult)
    mins1Loc.append(int(min1Loc*samplingFrequency))

    # Min2
    for k in range(510):
        m1 = slope(max1Loc+k, max1Loc+k+1, filtered[max1Loc+k], filtere
d[max1Loc+k+1])
        m2 = slope(max1Loc+k+1, max1Loc+k+2, filtered[max1Loc+k+1], fil
tered[max1Loc+k+2])
        if (m1 > 0 and m2 < 0) or \
            (m1 < 0 and m2 > 0):
            min2 = filtered[max1Loc+k+1]
            min2Loc = k+1
            break
    mins2Val.append(min2-shift*shiftMult)
    mins2Loc.append(int(min2Loc*samplingFrequency + max1Loc*samplingFre
quency))

    yAxis.append((filtered-shift*shiftMult)[0])

    plt.plot(xAxis, filtered-shift*shiftMult)
    shiftMult = shiftMult + 1

print("Global finished")
```

```
import numpy as np
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression
from sklearn import metrics


# MAXS
Xmaxs = np.array(maxsLoc).reshape(-1,1) # values converts it into a num
py array
Ymaxs = np.array(maxsVal).reshape(-1,1)  # -1 means that calculate the
dimension of rows, but have 1 column
linear_regressor_maxs = LinearRegression()  # create object for the cla
ss
linear_regressor_maxs.fit(Xmaxs, Ymaxs)  # perform linear regression
Y_pred_maxs = linear_regressor_maxs.predict(Xmaxs) # make predictions
R2 = metrics.r2_score(Ymaxs, Y_pred_maxs)


# Finding the best fit

Xmaxs_Temp = Xmaxs
Ymaxs_Temp = Ymaxs
noisyReceivers = []
for i in range(8):
    itemX = Xmaxs[i]
    itemY = Ymaxs[i]

    Xmaxs_Temp = np.delete(Xmaxs_Temp, i) # X value is removed
    Xmaxs_Temp = np.array(Xmaxs_Temp).reshape(-1,1) # reshaping
    Ymaxs_Temp = np.delete(Ymaxs_Temp, i) # Y value is removed
    Ymaxs_Temp = np.array(Ymaxs_Temp).reshape(-1,1) # reshapinged

    linear_regressor_maxs.fit(Xmaxs_Temp, Ymaxs_Temp)  # perform linear
regression
    Y_pred_maxs_Temp = linear_regressor_maxs.predict(Xmaxs_Temp)  # mak
e New predictions
    R2_Temp = metrics.r2_score(Ymaxs_Temp, Y_pred_maxs_Temp) # New R2 v
alue with 7 items

    if (R2_Temp-0.1) > R2 and R2_Temp >= 0.9:
        noisyReceivers.append(i)

    Xmaxs_Temp = np.insert(Xmaxs_Temp, i, itemX)
    Xmaxs_Temp = np.array(Xmaxs_Temp).reshape(-1,1) # reshaping
    Ymaxs_Temp = np.insert(Ymaxs_Temp, i, itemY)
    Ymaxs_Temp = np.array(Ymaxs_Temp).reshape(-1,1) # reshapinged


print("Noisy Receivers", noisyReceivers)


# Cleaning Noisy Receiver
if len(noisyReceivers) > 0:
    for r in range(len(noisyReceivers)):

        # Maxs
```

```
        Xmaxs = np.delete(Xmaxs, noisyReceivers[r]) # Noisy receiver is
removed
        Xmaxs = np.array(Xmaxs).reshape(-1,1) # reshaping
        Ymaxs = np.delete(Ymaxs, noisyReceivers[r]) # Noisy receiver is
removed
        Ymaxs = np.array(Ymaxs).reshape(-1,1) # reshaping

        # Mins1
        Xmins1 = np.delete(mins1Loc, noisyReceivers[r]) # Noisy receive
r is removed
        Xmins1 = np.array(Xmins1).reshape(-1,1) # values converts it in
to a numpy array
        Ymins1 = np.delete(mins1Val, noisyReceivers[r]) # Noisy receive
r is removed
        Ymins1 = np.array(Ymins1).reshape(-1,1)  # -1 means that calcul
ate the dimension of rows, but have 1 column

        # Mins2
        Xmins2 = np.delete(mins2Loc, noisyReceivers[r]) # Noisy receive
r is removed
        Xmins2 = np.array(Xmins2).reshape(-1,1) # values converts it in
to a numpy array
        Ymins2 = np.delete(mins2Val, noisyReceivers[r]) # Noisy receive
r is removed
        Ymins2 = np.array(Ymins2).reshape(-1,1)  # -1 means that calcul
ate the dimension of rows, but have 1 column

else:
# # Re-Shaping Data
# #     Xmaxs = np.array(maxsLoc).reshape(-1,1) # values converts it in
to a numpy array
# #     Ymaxs = np.array(maxsVal).reshape(-1,1)  # -1 means that calcul
ate the dimension of rows, but have 1 column

    Xmins1 = np.array(mins1Loc).reshape(-1,1) # values converts it into
a numpy array
    Ymins1 = np.array(mins1Val).reshape(-1,1)  # -1 means that calculat
e the dimension of rows, but have 1 column

    Xmins2 = np.array(mins2Loc).reshape(-1,1) # values converts it into
a numpy array
    Ymins2 = np.array(mins2Val).reshape(-1,1)  # -1 means that calculat
e the dimension of rows, but have 1 column



# MAXS Re-Regression
linear_regressor_maxs = LinearRegression()  # create object for the cla
ss
linear_regressor_maxs.fit(Xmaxs, Ymaxs)  # perform linear regression
Y_pred_maxs = linear_regressor_maxs.predict(Xmaxs) # make predictions
R2Maxs = metrics.r2_score(Ymaxs, Y_pred_maxs)


print("Maxs coef", linear_regressor_maxs.coef_, metrics.r2_score(Ymaxs,
Y_pred_maxs))

plt.scatter(Xmaxs, Ymaxs)
```

```
plt.plot(Xmaxs, Y_pred_maxs, color='black')


# MINS1 Regression
linear_regressor_mins1 = LinearRegression()  # create object for the cl
ass
linear_regressor_mins1.fit(Xmins1, Ymins1)  # perform linear regression
Y_pred_mins1 = linear_regressor_mins1.predict(Xmins1)  # make predictio
ns
R2Mins1 = metrics.r2_score(Ymins1, Y_pred_mins1)

print("Mins1 coef", linear_regressor_mins1.coef_, metrics.r2_score(Ymin
s1, Y_pred_mins1))

plt.scatter(Xmins1, Ymins1)
plt.plot(Xmins1, Y_pred_mins1, color='black')


# MINS2
linear_regressor_mins2 = LinearRegression()  # create object for the cl
ass
linear_regressor_mins2.fit(Xmins2, Ymins2)  # perform linear regression
Y_pred_mins2 = linear_regressor_mins2.predict(Xmins2)  # make predictio
ns
R2Mins2 = metrics.r2_score(Ymins2, Y_pred_mins2)

print("Mins2 coef", linear_regressor_mins2.coef_, metrics.r2_score(Ymin
s2, Y_pred_mins2))

plt.scatter(Xmins2, Ymins2)
plt.plot(Xmins2, Y_pred_mins2, color='black')

averageSlow = (1/abs(linear_regressor_maxs.coef_)*R2Maxs + \
               1/abs(linear_regressor_mins1.coef_)*R2Mins1 + \
               1/abs(linear_regressor_mins2.coef_)*R2Mins2 ) / (R2Maxs
+R2Mins1+R2Mins2)
print("averageSlow:", averageSlow)
print("Depth:", dfDiff.loc[depth,"DEPTH"])



plt.ylabel('Receivers')
plt.xlabel('Time (µs)')
plt.yticks(yAxis, y_ticks)
plt.title("X-Pole Sonic Logs at the depth of "+str(dfDiff.loc[depth,"DE
PTH"]))
plt.show()

print("Maxs", maxsLoc)
print("Maxs", maxsVal)

print("mins1", mins1Loc)
print("mins1", mins1Val)

print("mins2", mins2Loc)
print("mins2", mins2Val)
```

Iterate All The Data

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.pyplot import figure
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import scipy
from scipy import signal

lastDepth = 4239
samplingFrequency = 20
shift = 0.2 * 3.2808399 # feet conversion
shiftMult = 0

# Savitzky-Golay filter Paramaters
winLen = 33
polyOrd = 3


def slope(x1,x2,y1,y2):
    m = (y2-y1)/(x2-x1)
    return m



# Preparation of Result Lists
dfDiffResults = pd.DataFrame(columns=["Index", "Depth", "Method",
                                      "Average Slowness",
                                      "Global Max Coef", "Local Min1 Co
ef", "Local Min2 Coef",
                                      "Global Max R2", "Local Min1 R2",
"Local Min2 R2",

                                      "Global MaxLoc[1]","Global MaxLoc
[2]","Global MaxLoc[3]",
                                      "Global MaxLoc[4]","Global MaxLoc
[5]","Global MaxLoc[6]",
                                      "Global MaxLoc[7]","Global MaxLoc
[8]",

                                      "Local Min1Loc[1]","Local Min1Loc
[2]","Local Min1Loc[3]",
                                      "Local Min1Loc[4]","Local Min1Loc
[5]","Local Min1Loc[6]",
                                      "Local Min1Loc[7]","Local Min1Loc
[8]",

                                      "Local Min2Loc[1]","Local Min2Loc
[2]","Local Min2Loc[3]",
                                      "Local Min2Loc[4]","Local Min2Loc
[5]","Local Min2Loc[6]",
                                      "Local Min2Loc[7]","Local Min2Loc
[8]",
```

```
                                    "Global MaxValue[1]","Global MaxV
alue[2]","Global MaxValue[3]",
                                    "Global MaxValue[4]","Global MaxV
alue[5]","Global MaxValue[6]",
                                    "Global MaxValue[7]","Global MaxV
alue[8]",

                                    "Local Min1Value[1]","Local Min1V
alue[2]","Local Min1Value[3]",
                                    "Local Min1Value[4]","Local Min1V
alue[5]","Local Min1Value[6]",
                                    "Local Min1Value[7]","Local Min1V
alue[8]",

                                    "Local Min2Value[1]","Local Min2V
alue[2]","Local Min2Value[3]",
                                    "Local Min2Value[4]","Local Min2V
alue[5]","Local Min2Value[6]",
                                    "Local Min2Value[7]","Local Min2V
alue[8]"

                                   ])


# xAxis = np.linspace(1,512,512)
xAxis = np.arange(0, 512*samplingFrequency, 20)

# Depth iteration
for depth in range(4240):

    print("Depth:", depth)

    try:

        # Max, Min1 and Min2 Identification
        max1 = None
        min1 = None
        min2 = None

        max1Loc = None
        min1Loc = None
        min2Loc = None

        maxsVal = []
        mins1Val = []
        mins2Val = []

        maxsLoc = []
        mins1Loc = []
        mins2Loc = []

        Xmaxs = None
        Xmins1 = None
        Xmins2 = None

        Ymaxs = None
        Ymins1 = None
        Ymins2 = None
```

```
        linear_regressor_maxs = None
        linear_regressor_mins1 = None
        linear_regressor_mins2 = None

        Y_pred_maxs = None
        Y_pred_mins1 = None
        Y_pred_mins2 = None


        shift = 0.2 * 3.2808399 # feet conversion
        shiftMult = 0


            # Raw Data

            yFirstLine = dfDiff.loc[depth,"1_1":"1_512"]

            maxN = np.max(yFirstLine)
            minN = abs(np.min(yFirstLine))

            for i in range(8):

                y = dfDiff.loc[depth,str(i+1)+"_1":str(i+1)+"_512"]

                # Max1
                if maxN >= minN:
                    method = "Global Maximum Worked"
                    max1 = np.max(y)
                    max1Loc = np.where(y == np.amax(y))[0][0]

                else:
                    method = "Global Minimum Worked"
                    max1 = np.min(y)
                    max1Loc = np.where(y == np.amin(y))[0][0]

                maxsVal.append(max1-shift*shiftMult)
                maxsLoc.append(max1Loc*samplingFrequency)

                # Min1
                for j in range(510):
                    m1 = slope(max1Loc-j, max1Loc-j-1, y[max1Loc-j], y[
max1Loc-j-1])
                    m2 = slope(max1Loc-j-1, max1Loc-j-2, y[max1Loc-j-1]
, y[max1Loc-j-2])
                    if (m1 > 0 and m2 < 0) or \
                        (m1 < 0 and m2 > 0):
                        min1 = y[max1Loc-j-1]
                        min1Loc = max1Loc-j-1
                        break
                mins1Val.append(min1-shift*shiftMult)
                mins1Loc.append(min1Loc*samplingFrequency)

                # Min2
                for k in range(510):
                    m1 = slope(max1Loc+k, max1Loc+k+1, y[max1Loc+k], y[
max1Loc+k+1])
```

```
                    m2 = slope(max1Loc+k+1, max1Loc+k+2, y[max1Loc+k+1]
, y[max1Loc+k+2])
                    if (m1 > 0 and m2 < 0) or \
                        (m1 < 0 and m2 > 0):
                        min2 = y[max1Loc+k+1]
                        min2Loc = k+1
                        break
                mins2Val.append(min2-shift*shiftMult)
                mins2Loc.append(min2Loc*samplingFrequency + max1Loc*sam
plingFrequency)

                shiftMult = shiftMult + 1


        # Filtered Data

        filtered1stLine = signal.savgol_filter(dfDiff.loc[depth,"1_1":"
1_512"], \
                                        axis=0, window_length=winLen, polyo
rder=polyOrd)

        maxN = np.max(filtered1stLine)
        minN = abs(np.min(filtered1stLine))


        for i in range(8):
            filtered = signal.savgol_filter(dfDiff.loc[depth,str(i+1)+"
_1":str(i+1)+"_512"], \
                                                axis=0, window_length=winLe
n, polyorder=polyOrd)

            # Max1
            if maxN >= minN:
                method = "Global Maximum Worked"
                max1 = np.max(filtered)
                max1Loc = np.where(filtered == np.amax(filtered))[0][0]

            else:
                method = "Global Minimum Worked"
                max1 = np.min(filtered)
                max1Loc = np.where(filtered == np.amin(filtered))[0][0]

            maxsVal.append(max1-shift*shiftMult)
            maxsLoc.append(int(max1Loc*samplingFrequency))

            # Min1
            for j in range(510):
                m1 = slope(max1Loc-j, max1Loc-j-1, filtered[max1Loc-j],
filtered[max1Loc-j-1])
                m2 = slope(max1Loc-j-1, max1Loc-j-2, filtered[max1Loc-j
-1], filtered[max1Loc-j-2])
                if (m1 > 0 and m2 < 0) or \
                    (m1 < 0 and m2 > 0):
                    min1 = filtered[max1Loc-j-1]
                    min1Loc = max1Loc-j-1
                    break
            mins1Val.append(min1-shift*shiftMult)
            mins1Loc.append(int(min1Loc*samplingFrequency))
```

```
            # Min2
            for k in range(510):
                m1 = slope(max1Loc+k, max1Loc+k+1, filtered[max1Loc+k],
filtered[max1Loc+k+1])
                m2 = slope(max1Loc+k+1, max1Loc+k+2, filtered[max1Loc+k
+1], filtered[max1Loc+k+2])
                if (m1 > 0 and m2 < 0) or \
                    (m1 < 0 and m2 > 0):
                    min2 = filtered[max1Loc+k+1]
                    min2Loc = k+1
                    break
            mins2Val.append(min2-shift*shiftMult)
            mins2Loc.append(int(min2Loc*samplingFrequency + max1Loc*sam
plingFrequency))

            shiftMult = shiftMult + 1

        # MAXS
        Xmaxs = np.array(maxsLoc).reshape(-1,1) # values converts it in
to a numpy array
        Ymaxs = np.array(maxsVal).reshape(-1,1)  # -1 means that calcul
ate the dimension of rows, but have 1 column
        linear_regressor_maxs = LinearRegression()  # create object for
the class
        linear_regressor_maxs.fit(Xmaxs, Ymaxs)  # perform linear regre
ssion
        Y_pred_maxs = linear_regressor_maxs.predict(Xmaxs) # make predi
ctions
        R2 = metrics.r2_score(Ymaxs, Y_pred_maxs)


        # Finding the best fit

        Xmaxs_Temp = Xmaxs
        Ymaxs_Temp = Ymaxs
        noisyReceivers = []
        for i in range(8):
            itemX = Xmaxs[i]
            itemY = Ymaxs[i]

            Xmaxs_Temp = np.delete(Xmaxs_Temp, i) # X value is removed
            Xmaxs_Temp = np.array(Xmaxs_Temp).reshape(-1,1) # reshaping
            Ymaxs_Temp = np.delete(Ymaxs_Temp, i) # Y value is removed
            Ymaxs_Temp = np.array(Ymaxs_Temp).reshape(-1,1) # reshaping
ed

            linear_regressor_maxs.fit(Xmaxs_Temp, Ymaxs_Temp)  # perfor
m linear regression
            Y_pred_maxs_Temp = linear_regressor_maxs.predict(Xmaxs_Temp
)  # make New predictions
            R2_Temp = metrics.r2_score(Ymaxs_Temp, Y_pred_maxs_Temp) #
New R2 value with 7 items

            if (R2_Temp-0.1) > R2 and R2_Temp >= 0.9:
                noisyReceivers.append(i)

            Xmaxs_Temp = np.insert(Xmaxs_Temp, i, itemX)
```

```
            Xmaxs_Temp = np.array(Xmaxs_Temp).reshape(-1,1) # reshaping
            Ymaxs_Temp = np.insert(Ymaxs_Temp, i, itemY)
            Ymaxs_Temp = np.array(Ymaxs_Temp).reshape(-1,1) # reshaping
ed


        print("Noisy Receivers", noisyReceivers)


        # Cleaning Noisy Receiver
        if len(noisyReceivers) > 0:
            for r in range(len(noisyReceivers)):

                # Maxs
                Xmaxs = np.delete(Xmaxs, noisyReceivers[r]) # Noisy rec
eiver is removed
                Xmaxs = np.array(Xmaxs).reshape(-1,1) # reshaping
                Ymaxs = np.delete(Ymaxs, noisyReceivers[r]) # Noisy rec
eiver is removed
                Ymaxs = np.array(Ymaxs).reshape(-1,1) # reshaping

                # Mins1
                Xmins1 = np.delete(mins1Loc, noisyReceivers[r]) # Noisy
receiver is removed
                Xmins1 = np.array(Xmins1).reshape(-1,1) # values conver
ts it into a numpy array
                Ymins1 = np.delete(mins1Val, noisyReceivers[r]) # Noisy
receiver is removed
                Ymins1 = np.array(Ymins1).reshape(-1,1)  # -1 means tha
t calculate the dimension of rows, but have 1 column

                # Mins2
                Xmins2 = np.delete(mins2Loc, noisyReceivers[r]) # Noisy
receiver is removed
                Xmins2 = np.array(Xmins2).reshape(-1,1) # values conver
ts it into a numpy array
                Ymins2 = np.delete(mins2Val, noisyReceivers[r]) # Noisy
receiver is removed
                Ymins2 = np.array(Ymins2).reshape(-1,1)  # -1 means tha
t calculate the dimension of rows, but have 1 column

        else:
        # Re-Shaping Data

            Xmins1 = np.array(mins1Loc).reshape(-1,1) # values converts
it into a numpy array
            Ymins1 = np.array(mins1Val).reshape(-1,1)  # -1 means that
calculate the dimension of rows, but have 1 column

            Xmins2 = np.array(mins2Loc).reshape(-1,1) # values converts
it into a numpy array
            Ymins2 = np.array(mins2Val).reshape(-1,1)  # -1 means that
calculate the dimension of rows, but have 1 column



        # MAXS Re-Regression
```

```
        linear_regressor_maxs = LinearRegression()  # create object for
the class
        linear_regressor_maxs.fit(Xmaxs, Ymaxs)  # perform linear regre
ssion
        Y_pred_maxs = linear_regressor_maxs.predict(Xmaxs) # make predi
ctions
        Y_maxs_R2 = metrics.r2_score(Ymaxs, Y_pred_maxs)

        # MINS1 Regression
        linear_regressor_mins1 = LinearRegression()  # create object fo
r the class
        linear_regressor_mins1.fit(Xmins1, Ymins1)  # perform linear re
gression
        Y_pred_mins1 = linear_regressor_mins1.predict(Xmins1)  # make p
redictions
        Y_mins1_R2 = metrics.r2_score(Ymins1, Y_pred_mins1)

        # MINS2
        linear_regressor_mins2 = LinearRegression()  # create object fo
r the class
        linear_regressor_mins2.fit(Xmins2, Ymins2)  # perform linear re
gression
        Y_pred_mins2 = linear_regressor_mins2.predict(Xmins2)  # make p
redictions
        Y_mins2_R2 = metrics.r2_score(Ymins2, Y_pred_mins2)

        averageSlow = (1/abs(linear_regressor_maxs.coef_)*Y_maxs_R2 + \
                       1/abs(linear_regressor_mins1.coef_)*Y_mins1_R2
+ \
                       1/abs(linear_regressor_mins2.coef_)*Y_mins2_R2
) / (Y_maxs_R2+Y_mins1_R2+Y_mins2_R2)


        # inserting values to results dataframe
        dfDiffResults.loc[depth,"Index"] = depth
        dfDiffResults.loc[depth,"Depth"] = dfDiff.loc[depth,"DEPTH"]
        dfDiffResults.loc[depth,"Method"] = method

        dfDiffResults.loc[depth,"Average Slowness"] = averageSlow[0][0]
        dfDiffResults.loc[depth,"Global Max Coef"] = linear_regressor_m
axs.coef_[0][0]
        dfDiffResults.loc[depth,"Local Min1 Coef"] = linear_regressor_m
ins1.coef_[0][0]
        dfDiffResults.loc[depth,"Local Min2 Coef"] = linear_regressor_m
ins2.coef_[0][0]

        dfDiffResults.loc[depth,"Global Max R2"] = Y_maxs_R2
        dfDiffResults.loc[depth,"Local Min1 R2"] = Y_mins1_R2
        dfDiffResults.loc[depth,"Local Min2 R2"] = Y_mins2_R2

        dfDiffResults.iloc[depth, 9:17] = maxsLoc
        dfDiffResults.iloc[depth, 17:25] = mins1Loc
        dfDiffResults.iloc[depth, 25:33] = mins2Loc

        dfDiffResults.iloc[depth,33:41] = maxsVal
        dfDiffResults.iloc[depth,41:49] = mins1Val
        dfDiffResults.iloc[depth,49:57] = mins2Val
```

```
    except:
        print(depth, "an error occured")
        print("")
        print("")


        dfDiffResults.loc[depth,"Index"] = depth
        dfDiffResults.loc[depth,"Depth"] = dfDiff.loc[depth,"DEPTH"]

dfDiffResults.to_csv(r'C:\Users\~\FileName.csv')
print(dfDiffResults)
```

APPENDIX 2: LSTM Pre-Processing Source Code

Reading CSV - Data

```
import pandas as pd
df = pd.read_csv(r'C:\Users\~\FileName.csv')
print(df.shape)
print(df)
```

```
# Splitting The Data

dfX = df.iloc[:, 1:4097]
dfY = df.loc[:, "DTX"]
```

```
print(dfX.shape)
print(dfX)

print(dfY.shape)
print(dfY)
```

DataFrames to Matrices

```
# importing numpy
import numpy as np

# output array
randomLines = np.random.randint(low = 0, high = 25592, size = 10000)
print(randomLines.shape)
print(len(randomLines))
print(randomLines)
```

```
# X Dataset to Matrix FILTERED
import numpy as np
import scipy
from scipy import signal

x_train = None
y_train = None


# Savgol Filter Parameters
winLen = 41
polyOrd = 3

# Scaler finds the absolute maximum in order to normalize the data
scaler = max(abs(dfX.loc[0, "1_1":"8_512"].min()),
             abs(dfX.loc[0, "1_1":"8_512"].max())
```

```
            )


x_train = np.array([[signal.savgol_filter(dfX.loc[randomLines[0], "1_1"
:"1_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "2_1"
:"2_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "3_1"
:"3_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "4_1"
:"4_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "5_1"
:"5_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "6_1"
:"6_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "7_1"
:"7_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[0], "8_1"
:"8_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd)
                   ]])

y_train = np.array([dfY.loc[randomLines[0], ]])

for i in range(len(randomLines)-1):
    tempX = None
    tempY = None
    scaler = None
    scaler = max(abs(dfX.loc[randomLines[i+1], "1_1":"8_512"].min()),
                 abs(dfX.loc[randomLines[i+1], "1_1":"8_512"].max())
                )

    tempX = np.array([[signal.savgol_filter(dfX.loc[randomLines[i+1], "
1_1":"1_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "2_
1":"2_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "3_
1":"3_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "4_
1":"4_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "5_
1":"5_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "6_
1":"6_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "7_
1":"7_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd),
                    signal.savgol_filter(dfX.loc[randomLines[i+1], "8_
1":"8_512"]/scaler, axis=0, window_length=winLen, polyorder=polyOrd)
                   ]])
    tempY = np.array([dfY.loc[randomLines[i+1], ]])

    x_train = np.insert(x_train, len(x_train), tempX, axis=0)
    y_train = np.insert(y_train, len(y_train), tempY, axis=0)


# Y Dataset to Matrix
y_data = y_train

y_data = np.reshape(y_data, (len(randomLines),1))
```

```
y_data = y_data.astype('float32')
print(type(y_data))
print(y_data.shape)

# Assigning the y_data to y_data_scaled in order to avoid to break the
original arrays
y_train = y_data / 500
print(y_train)


print("Merged")
print(type(x_train))
print(x_train.shape)

print(type(y_train))
print(y_train.shape)
```

```
# Numpy Arrays are saved in order to avoiding loading the raw files aga
in
np.save(r"C:\Users\Seckin\OneDrive - University of Leicester\3. Dissert
ation\1. Python Codes\merged_x_data_arr", x_data)
np.save(r"C:\Users\Seckin\OneDrive - University of Leicester\3. Dissert
ation\1. Python Codes\merged_y_data_arr", y_data)
```

```
# Checking the range of the input and output arrays
print(np.min(x_train))
print(np.max(x_train))
print(np.min(y_train))
print(np.max(y_train))
```

TENSORFLOW

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Activation
from keras.optimizers import SGD, Adadelta
from keras.callbacks import EarlyStopping

# Building The Model
model = Sequential()

model.add(LSTM(512, input_shape=(8, 512), activation="relu", return_seq
uences=False))
model.add(Dense(512, activation="relu"))
model.add(Dense(100, activation="relu"))
model.add(Dense(50, activation="relu"))

model.add(Dense(1))
model.load_weights("Model_Master_Merged_19.h5")
model.compile(loss="mse", optimizer="adam") # , metrics=["mse", "accura
cy"]

checkpointer = EarlyStopping(monitor="val_loss", min_delta=1e-4, patien
ce=5, verbose=1, mode="auto")
```

```
history = model.fit(x_train, y_train, validation_split=0.5, callbacks=[
checkpointer], epochs=10,  verbose=1)
```

```
# SAVING THE MODEL

from keras.models import load_model
model.save("Model_Master_Merged_20.h5")
```

PREDICTING

```
# LOADING THE MASTER MODEL
from keras.models import load_model
model = load_model("Model_Master_Merged_20.h5")
```

```
# importing numpy
import numpy as np

# Loading pre-saved numpy arrays
x_data = np.load(r"C:\Users\Seckin\OneDrive - University of Leicester\3
. Dissertation\1. Python Codes\00. merged_x_data_arr.npy")
y_data = np.load(r"C:\Users\Seckin\OneDrive - University of Leicester\3
. Dissertation\1. Python Codes\00. merged_y_data_arr.npy")
```

```
# TESTING THE WHOLE DATA

realOutputList = []
predictionList = []

for i in range(len(randomLines)-1):
    x_new = np.reshape(x_train[i], (1, 8, 512))
    y_new = np.reshape(y_train[i], (1,1))

    # Model Prediction Based on The New Input
    predict = model.predict(x_new)
    realOutputList.append(y_train[i][0]*500)
    predictionList.append(predict[0][0]*500)
```

```
# COMPARISON OF REAL AND PREDICTED OUTPUTS
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

plt.plot(realOutputList)
plt.plot(predictionList)
plt.show()
```

```
# SCATTER PLOT IN ORDER TO COMPARE FITNESS
plt.scatter(realOutputList, predictionList)
plt.show()
```

```
# DRAWING LINEAR REGRESSION LINE OVER PREDICTION
# IN ORDER TO GET R2 VALUE

from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(9, 6), dpi=80, facecolor='w', edgecolor='k')

# Output Regression
realValues = np.array(realOutputList).reshape(-1,1) # values converts i
t into a numpy array
predictedValues = np.array(predictionList).reshape(-1,1)  # -1 means th
at calculate the dimension of rows, but have 1 column
linear_regressor_output = LinearRegression()  # create object for the c
lass
linear_regressor_output.fit(realValues, predictedValues)  # perform lin
ear regression
Y_pred_output = linear_regressor_output.predict(realValues)  # make pre
dictions

print("Maxs coef", linear_regressor_output.coef_, metrics.r2_score(real
Values, predictedValues))

plt.scatter(realValues, predictedValues)
plt.plot(realValues, Y_pred_output, color='black')
plt.xlabel("Real Values")
plt.ylabel("Predicted Values")
plt.legend(('R2={}'.format(round(metrics.r2_score(realValues, predicted
Values),4)),), loc="upper left")
plt.title("Randomly Trained Data Results")
```

```
# TRAINING - TEST LOSS VISUALISATION
import matplotlib.pyplot as plt

figure(num=None, figsize=(9, 6), dpi=80, facecolor='w', edgecolor='k')

dfLoss = None

dfLoss = pd.DataFrame({'Loss':history.history["loss"]})
dfLoss.to_csv(r'C:\Users\~\FileName.csv')
plt.plot(history.history["loss"], label = "Training Loss")
plt.plot(history.history["val_loss"], label = "Validation Loss")
plt.xlabel("Number of Epochs")
plt.ylabel("MSE Loss")
plt.show()
```