INTERPRETING DRILLING RECORDS TO PREDICT WELL SUCCESS

A REPORT
SUBMITTED TO THE DEPARTMENT OF ENERGY RESOURCES
ENGINEERING
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dang Ton
August 2021

I certify that I have read this report and that, in my opinion, it is fully adequate, in scope and quality, as partial fulfillment of the degree of Master of Science in Energy Resources Engineering.

Prof. Roland N. Horne (Principal Adviser)

Approved for the Stanford University Committee on Graduate Studies

Abstract

For the last decade, modern machine learning, especially the deep neural network, has made tremendous impacts to all facets of life. The energy industry is not oblivious to the trend, and machine learning has found its applications in all of kind of operations. Often, the most cost-intensive operation that is carried out in the energy industry is the well drilling operation. This report is an extensive study on the application of deep neural networks in optimizing geothermal well drilling operation, but can be readily applied to oil and gas well due to their similarities.

This study shows that drilling optimization by optimizing rate-of-penetration from past drilling records is a highly delicate process. Not only a sufficient amount of data from multiple sources of information is required, the data also must be properly recorded and standardized, or it must go through preprocessing to filter out invalid entries and to standardize. In this study, a R^2 score of **0.53** has been achieved for the rate-of-penetration prediction deep neural network model with random train/validation splitting scheme. However, as the rate-of-penetration prediction model trained with random splitting scheme has limited usage in production, an alternative splitting scheme called well-by-well train/validation splitting is used. With the well-by-well train/validation splitting scheme, the rate-of-penetration prediction deep neural network model was only able to achieve a R^2 score of **0.1**.

Rate-of-penetration is not the only criterion used in drilling optimization. This study also considered optimization by reducing nonproductive times. The most common reasons for nonproductive times are tripping, and the drillers encountering problems. Therefore, having a model that can forecast these nonproductive times is helpful to the overall quest of minimizing the total costs. In this study, machine learning models that can predict potential tripping/problems from drilling records have been developed, and both models have satisfactory accuracy to be used in real life situations.

The problem of incorporating nonnumerical entries in drilling records was also studied in this study. In addition to the standard numerical-type entries, textural-type entries are often found in

drilling records. These textural-type entries are often nonstandardized written English remarks/-comments from the drillers, which may carry useful information about the condition of the well that does not available elsewhere in the records. However, these remarks/comments require expensive and time-consuming manual data preprocessing in order to incorporate them into machine learning models. Bidirectional Encoder Representations from Transformers (BERT) provides a solution for automating preprocessing of textural data. Using textual information, together with numerical information, has improved the quality of predictions in most cases.

Acknowledgments

I would like to express my sincere gratitude to my adviser, Prof. Roland N. Horne, for his assistance at every stage of this study, and his insightful comments, suggestions, and support throughout my time at Stanford. This study is not possible without him.

I also would like to extend my sincere thanks to everyone in the EDGE project, which provided financial support and the dataset used in this study. With their immense knowledge, expertise, and devotion, the people in the EDGE project have provided me with invaluable support throughout this study.

Finally, I would like to offer my special thanks to all people at Stanford, including but not limited to my colleagues in the SUPRI-D research group. I appreciate all the supports I received at Stanford, which helped me through the most difficult time.

Glossary

Backpropagation: An algorithm for training feed-forward neural networks. Backpropagation involves calculation of the gradients backward from the last layer to the first layer.

Correlation plot: A scatter plot between predicted values and true values

Feed-forward neural network: A neural network where output of the previous layer is served as input of the next layer. Each layer has the form of $y(x) = \mathscr{F}(x)$ where x is the input and $\mathscr{F}(x)$ is a learnable mapping

$\mathbf{R^2}$ score: Also called Coefficient of determination, normally lies between 0.0 and 1.0. Measure the goodness-of-fit between true and predicted values

Overfit: A machine learning model is overfitted when it fits the training data well but performs poorly on validation data.

Recurrent neural network: A type of neural network that works only on sequential data, and the outputs depend on the prior elements of the input sequence

Stuck pipe: Stuck pipe is when the drill pipe cannot be rotated, or moved. The pipe is considered stuck if it cannot be freed without damaging the pipe, wellbore, or equipment.

Training set: The dataset used during the training process.

Tripping: The act of pulling the drill string out the wellbore and putting it back in. Usually done to replace the drill bit or retrieve loose items in the wellbore

Validation set: The dataset used to provide unbiased evaluation of a model.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Learning (ML), and to some extent Artificial Neural Network (ANN) are rising trends that have dominated many different industries. The oil and gas industry, and the geothermal industry, are not exceptions to these trends. There has been a great interest in these industries to take advantage of the recent progresses in ML to uncover valuable insights for profitability and safety. Naturally, drilling has been a frequent application of interest.

Drilling operations use highly intricate equipment in a very uncertain and hostile downhole environment, while keeping the costs to a minimum. Due to the high cost of renting and operating a drilling rig, reducing the total drilling time always has high priority when talking about drilling optimizations. Typical optimizations include increasing rate-of-penetration (ROP), and reducing nonproductive times (due to unscheduled tripping and problems, etc.) However, there is no clear relationship between the hundreds of parameters sent from the rigs and the optimizing parameters. Compounded with the complexities in downhole lithology and wellbore, drilling optimizations can be unreliable and depend strongly on who is making the prediction.

Machine learning offers a promising solution to the intricate problem of drilling optimization. Modern ML methods like ANN not only can accurately describe complex relationship between rig data and optimizing parameters, but can also process nonnumeric data like written text, sound or image recordings. Coupled with the ability to process inputs with missing data, modern ML offers a new perspective on drilling optimization where classical statistical methods have failed before. With modern ML, the dream of autonomous drilling in complex formation with minimal prior knowledge may be a reality in the future.

A collection of drilling records from different geothermal field in the United States and Iceland,

each with different geological properties, was used in this study. A preliminary analysis of the collection of drilling records is the main topic of Chapter 3. Chapter 4 will discuss about the deep neural network model developed to predict rate-of-penetration, and potential future tripping/problems. The problem of integrating text data into the predictions is also in the scope of this study, and will be discussed in Chapter 5.

Chapter 2

Background & Related Work

Using machine learning in drilling optimization is not an entirely new concept. Although not as popular as ML in production forecasting or reservoir engineering, there have been many papers written on the topic of using ML in drilling.

Rate-of-penetration (ROP) has often been the benchmark of drilling performance of a well, and positively correlated to cost. Low ROP results in more days required to drill which in turn increases the overall cost. However, increase in ROP, which reduces the cost of renting the rig, can also result in increase in overall costs because high ROP drilling may be associated with frequent bit changes or high risk in drilling downtime due to problems like stuck pipe. Therefore, drilling optimization involves finding a sweet spot in ROP that minimizes costs. Given the importance and complexity of ROP modeling, data-driven modeling of ROP has been a focal point of drilling optimizations.

There have been many empirical mathematical models for ROP, with the earliest dating back to 1964 with the Bingham model[2]. Unfortunately, the accuracy of these models has been routinely questioned[3] as their applicability on different formations are highly uncertain. Data-driven modeling with ML is a promising new direction in ROP modeling. These models have been shown to be able to generalize across different datasets.

2.1 Empirical models for rate of penetration

Bingham ROP model[2] is an early ROP model documented in literature. The model is bit-type agnostic model:

$$\mathrm{ROP} = \alpha \cdot \mathrm{RPM} \cdot \left[\frac{\mathrm{WOB}}{D_b}\right]^{\beta} \tag{2.1}$$

where:

ROP is the rate of penetration

α, β are formation dependence constants

RPM is the rotary speed of the drill string

WOB is the weight on bit

D_b is the bit diameter

An improved ROP model is the Bourgoyne and Young model[4], which models ROP as a function of eight parameters:

$$\mathrm{ROP} = \exp\left[a_0 + \sum_{i=2}^{8} a_i x_i\right] \tag{2.2}$$

where:

ROP is the rate of penetration (ft/hr)

a_1 is the formation strength parameter

x_2, a_2 are the normal compaction trend and its exponent

x_3, a_3 are the under compaction trend and its exponent

x_4, a_4 are the pressure differential and its exponent

x_5, a_5 are the weight on bit and its exponent

x_6, a_6 are the rotary speed of the drill string and its exponent

x_7, a_7 are the tooth wear parameter and its exponent

x_8, a_8 are the bit hydraulics parameter and its exponent

With the rise of polycrystalline diamond compact (PDC) bits, a PDC-specific model was developed[5], which includes the effect of rock strength and bit wear:

$$\mathrm{ROP} = W_f \cdot \frac{G \cdot \mathrm{WOB}^{\alpha} \cdot \mathrm{RPM}^{\gamma}}{D_b \cdot \mathrm{CCS}} \tag{2.3}$$

where:

ROP is the rate of penetration

α, γ are formation dependence constants

RPM is the rotary speed of the drill string

WOB is the weight on bit

D_b is the bit diameter

CCS is the confined rock strength

The Bingham model is the most basic ROP model, it assumes that the two main factors that affecting ROP are WOB and RPM. The Motahhari model includes the effect of rock strength, and the Bourgoyne and Young model adds mud flow-rate as a factor for ROP data-driven modeling. The common point of these models is that they all have empirical constants that are unique for each formation and bit design. Usually, these constants are found using history matching, where they are fitted to minimize errors over past drilling records of the same or similar wells. However, having empirical constants makes these models unable to generalize well across different geology and bit design.

2.2 Data-driven models for rate of penetration

Empirical models for rate of penetration, while they are easy to interpret, are not robust enough to account for the effect each drilling parameter has on others[6]. Compounding with the fact that most empirical models have dataset-specific parameters makes generalizing these models difficult.

In oil and gas, Big Data has become an integral part of operations, where an astounding amount of data is generated and processed every second. This is also undoubtedly true for the drilling industry, where a modern drilling rig can output hundreds of measurement each second. Huge volumes of fluid flow rates, pressure, temperature, drill string and bit conditions, etc. are the expected output of a modern rig. Given the complexity and nonlinear nature of these data, an equally complex and sophisticated model is needed to process and model them.

Data-driven modeling with ANN provides a solution to model the Big Data encountered daily in drilling operations. With proper usage, ANN can unravel complex relationships that enable accurate prediction. In addition, ANN models can predict future performance based on information of past wells, and enable transfer of knowledge between different geological domains.

There have been many attempts at using data-driven models to optimize ROP. Li and Samuel[7] predicted future ROP using ANN with surface and downhole values of torque, WOB, and rotary speed measurement as inputs. The paper shows good accuracy for ROP predictions. There are also several similar attempts to optimize the ROP that have been done[6 8 9 10], which all yielded promising

results using ML in drilling optimization.

2.3 Criteria for drilling optimization

ROP is the most common optimization criterion for drilling optimization because it is strongly correlated to costs. However, drilling optimization should not stop at ROP because there are also other optimization criteria that can bring similar level of cost savings. ROP optimization has the final goal to reduce the amount of time spent drilling, and consequently reduce the final costs. In addition to optimizing the amount of time spent drilling (productive time), minimizing nonproductive time will also have similar effect on overall costs. Nonproductive times are usually the result of unscheduled drilling problems such as top drive and circulation system failures, well cleaning, lost circulation, etc.

In addition to having a good model to finding ROP sweet spot, drillers also benefit from having a model that can give them warning about potential future drilling problems using past data. This will help drillers to further reduce rig-on-site time, which can result in significant cost savings.

Chapter 3

Dataset

This chapter introduces the dataset used in this study, and discusses the origin, contents, structures and problems with the dataset. This chapter also discusses the exploratory data analysis done on this dataset and how to prepare the data for later machine learning study.

3.1 Introduction

The dataset used in this study is a part of the EDGE project, which is a research project aimed at developing machine learning strategies in geothermal drilling optimization under the support of the EDGE Program of the US Department of Energy (DOE) Geothermal Technologies Office. The EDGE project aims to build a database of geothermal drilling data and from there develop optimization schemes based on machine learning and deep learning methodologies. Currently, the dataset includes data from 113 wells from different geothermal projects developed over the past thirty years. These data represent vastly different geologic and operational settings.

For each well, records are collected, averaged daily, and tallied by record number. All collected records are stored in a relational database for the ease of query and modification. Each relational table in the database is related to other tables by using WellID and Record Number as keys. There are a total of 63 tables, each corresponding to a different source of information. Sources of information include drilling rigs, drill string, drill bit, mud logging, etc. The high variety in information sources is very beneficial to the process of ROP optimization as modeling the subsurface environment accurately requires detailed surface measurements but also detailed wellbore measurements.

Although having numerous sources of information, it is not recommended to use all tables as input due to both the nature of the dataset and computational power constraint. The continuity

of some sources of information is not guaranteed. In some wells, the collection frequency of some sources is low enough that there are only few datapoints for the entire drilling operation. In some other cases, the information is only collected in short bursts a few times. Due to the sparsity of some sources, it is not possible to make assessments about the quality of the collected information. Therefore, only the frequently collected features were selected to use as inputs to the models. There is also the problem of nonnumeric records, which will be addressed in Chapter 5 of this study. The relational tables that were selected to use in our model are shown in Table 3.1, with focus on the `dailydrilldetail` table and `dailybitinfo` table. These two relational tables contain basic parameters that are frequently used to predict ROP. Another other problem with using all relational tables is the limits in computational power. If all relational tables are used as input, the number of features will be in the thousands. Training big ML models with that many input features is computationally intractable.

3.2 Exploratory data analysis

By selecting frequently collected features from the database, it is possible to do exploratory data analysis (EDA) on the collected data to make assessments about their quality and find relationships between variables.

The first problem in the dataset is that data gaps appear frequently in many features as seen in Table 3.2. As some machine learning methods cannot deal with missing data, this will have negative impacts in the analysis and the modeling of the dataset.

The first step in the EDA is to graph the histograms of all features. Because naturally occurring variables will have the tendency to be distributed under some common probability distributions (Gaussian, Exponential, etc.), any feature that is unusually distributed is worth examining. It is easy to see from Figure 3.1 and Figure 3.2 that while some features exhibit common distributions (MudFlowMax, AnnVelocityDC, AnnVelocityDP, BitHrs), most features do have a spike in the histogram around zero value. This is an unexpected behaviour, which indicates these features may have extreme positive outliers or wrong units. Another unexpected problem is that some features have negative values when they should be positive (BitFootage, BitROPAvg).

Histograms of some example features that exhibit the zero are in Figure 3.3 and Figure 3.4. It is easy to see that most recorded values of BitWOBAvg and BitTorqAvg data are in the [0-1000] range, both features contain some values that go up to 35000, or about three orders of magnitude difference compared to typical values. A theory for this behavior is that the different drillers mixed up the unit

Table 3.1: Selected relational tables

Table name	Content	Example features
`bhainfo`	Basic info about each BHA used	BHA Length; BHA Weight; Jars Weight; WOB; etc.
`bhadetail`	Detailed about constructions of each BHA used	Inner diameter; Outer diameter; Tensile strength; Collar types; etc.
`bitinfo`	Basic info about each Bit used	Bit manufacture; Diameter; Types; IADC code; etc
`bitruninfo`	Detailed about constructions of each Bit used	Jets diameter; Time on Bit; Pump pressure; Wear and Dull condition; etc.
`dailybitinfo`	Daily drilling parameters from bits data (Collected daily)	Bit footage; Bit ROP Average; Bit ROP Max; Bit WOB Average; Bit WOB Max; Jet Velocity; etc.
`dailyreport`	Basic summary of daily operations from rigs data (Collected daily)	Start Depth; Ending Depth; Current Operation; Future Operation; etc.
`dailydrilldetail`	Daily drilling parameters from rigs data (Collected daily)	Mud flow rate; ROP Average; ROP Max; WOB Average; WOB Max; Pump pressure; etc.
`dailydrillmisc`	Daily miscellaneous information from rigs data (Collected daily)	Hook load rotate; Hook load pickup; Pump 1-4 pressure; Drag Average; Drag Max; Annulus pressure Average; etc.
`dailyoperations`	Coded of daily operations with driller comments (Collected daily)	Operations Code; Phase Code; Descriptions
`dailysolidscontrol`	Daily parameters from rigs shakers (Collected daily)	Shaker 1-6 Top and Bottom; Desanders hours; Degasses hours; etc.
`dailymudreport`	Daily mud parameters from rigs data (Collected daily)	Mud density; Mud viscosity; Mud resistivity; etc

Table 3.2: Missing data rate of some features

Feature name	Number of missing values	Total number of values	Data missing rate (%)
Mud flow Average	848	4624	18.34
ROP Average	346	4624	7.49
Pump pressure Average	628	4624	13.58
Torque Average	1943	4624	42.02
Bit ROP Average	555	4341	12.79
Bit WOB Average	753	4341	17.34
Bit Pressure Drop	1159	4341	26.7

when recording the values: some recorded BitWOBAvg as lbs while some as kilo-lbs, etc. An easy fix to this problem is noticing the original units for BitWOBAvg and BitTorqAvg are lbs and lbf·ft. Therefore, any value under 100 is incorrectly recorded and should be multiplied by 1000. The results of the proposed correction method are in Figure 3.5 and Figure 3.6. It is observed that in Figure 3.5, the proposed correction method turns the BitWOBAvg distribution into a Gaussian distribution, which indicates that our theory likely is correct. However, the opposite happened in Figure 3.6, where the proposed correction method does not significantly change the distribution shape, which may indicate that our theory is not applicable for BitTorqAvg. In addition, because torque values in drilling can vary significantly when drilling when factoring in stuck pipe, lost circulation and mud motor, BitTorqAvg should be left as is. Another problem with BitTorqAvg is that some extreme values (20000, 35000, etc.) only appear a few times in the entire dataset. These datapoints can be considered extreme outliers and removed.

By following the same procedures as BitWOBAvg and BitTorqAvg, improper and extreme outlier values can be identified and fixed/filtered out as seen in Table 3.3.

After all improper values in the dataset are taken care of, the next step is to find if there is any underlying structure in the dataset. This is crucial as if some features are incorrectly recorded, or the features have no correlation to each other, there will be no or very weak underlying structures and the modeling job will be much harder. A commonly used analysis method for this step is principal component analysis (PCA), which is a dimensionality reduction method so the data can be visualized. However, for a dataset with a large number of dimensions like ours, there are superior dimensionality reduction methods. In this study, t-distributed stochastic neighbor embedding (t-SNE)[11] was used for dimensionality reduction. Because t-SNE cannot function properly with missing data, any data row with one or more missing points is dropped completely. This act of dropping rows with missing values significantly reduces the number of data points available for t-SNE analysis

Table 3.3: Correction methods used

Relational table	Feature name	Correction method
`dailybitinfo`	BitMudDensity	Remove all values BitMudDensity < 1 and BitMudDensity > 20
`dailybitinfo`	BitWOBAvg	BitWOBAvg ≤ 100 *= 1000
`dailybitinfo`	BitTorqAvg	Remove all values BitTorqAvg > 20000
`dailybitinfo`	BitRPMAvg	Remove all values BitRPMAvg > 2000
`dailybitinfo`	BitMudFlowAvg	Remove all values BitMudFlowAvg > 1600
`dailybitinfo`	BitROPAvg	Remove all values BitROPAvg > 1600
`dailybitinfo`	BitWOBAvg	Remove all values BitWOBAvg > 300
`dailydrilldetail`	WOBAvg	WOBAvg ≤ 100 *= 1000
`dailydrilldetail`	RPMAvg	Remove all values RPMAvg $>$ 2000
`dailydrilldetail`	MudFlowAvg	Remove all values MudFlowAvg > 1600
`dailydrilldetail`	ROPAvg	Remove all values ROPAvg $>$ 200
`dailydrilldetail`	PumpPSIAvg	Remove all values PumpPSIAvg > 5000

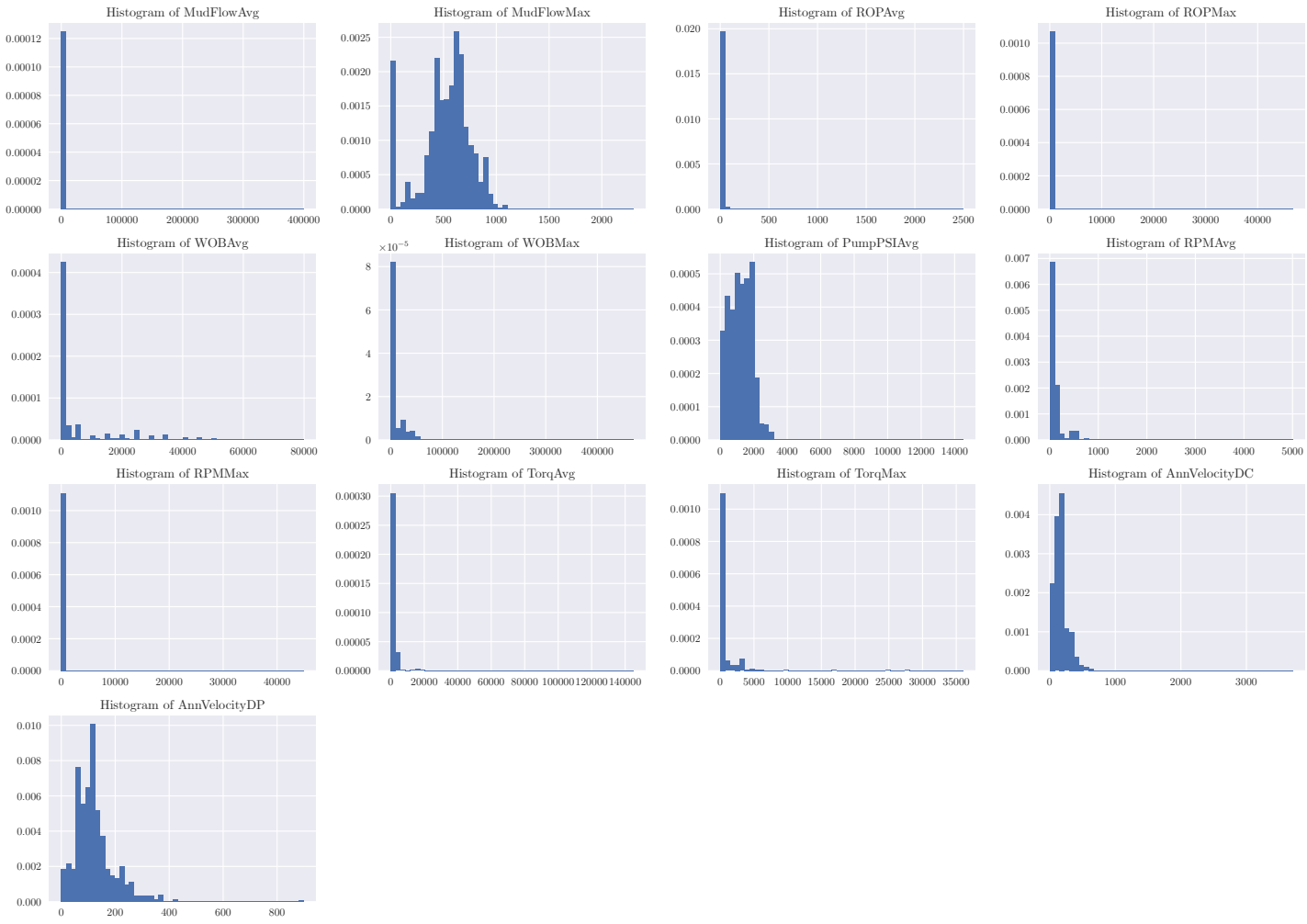

Figure 3.1: Histogram of `dailydrilldetail` data

as seen in Table 3.4.

Table 3.4: Dataset after removing rows with missing values

Relational table	Number of rows after drop	Number of rows before drop	Drop rate (%)
`dailybitinfo`	1375	4341	68.32
`dailydrilldetail`	1144	4624	75.26

From Figure 3.7 and Figure 3.8, it is observed that in both cases dropping the missing data rows has no detrimental effect on the EDA as t-SNE was still able to separate different geothermal fields

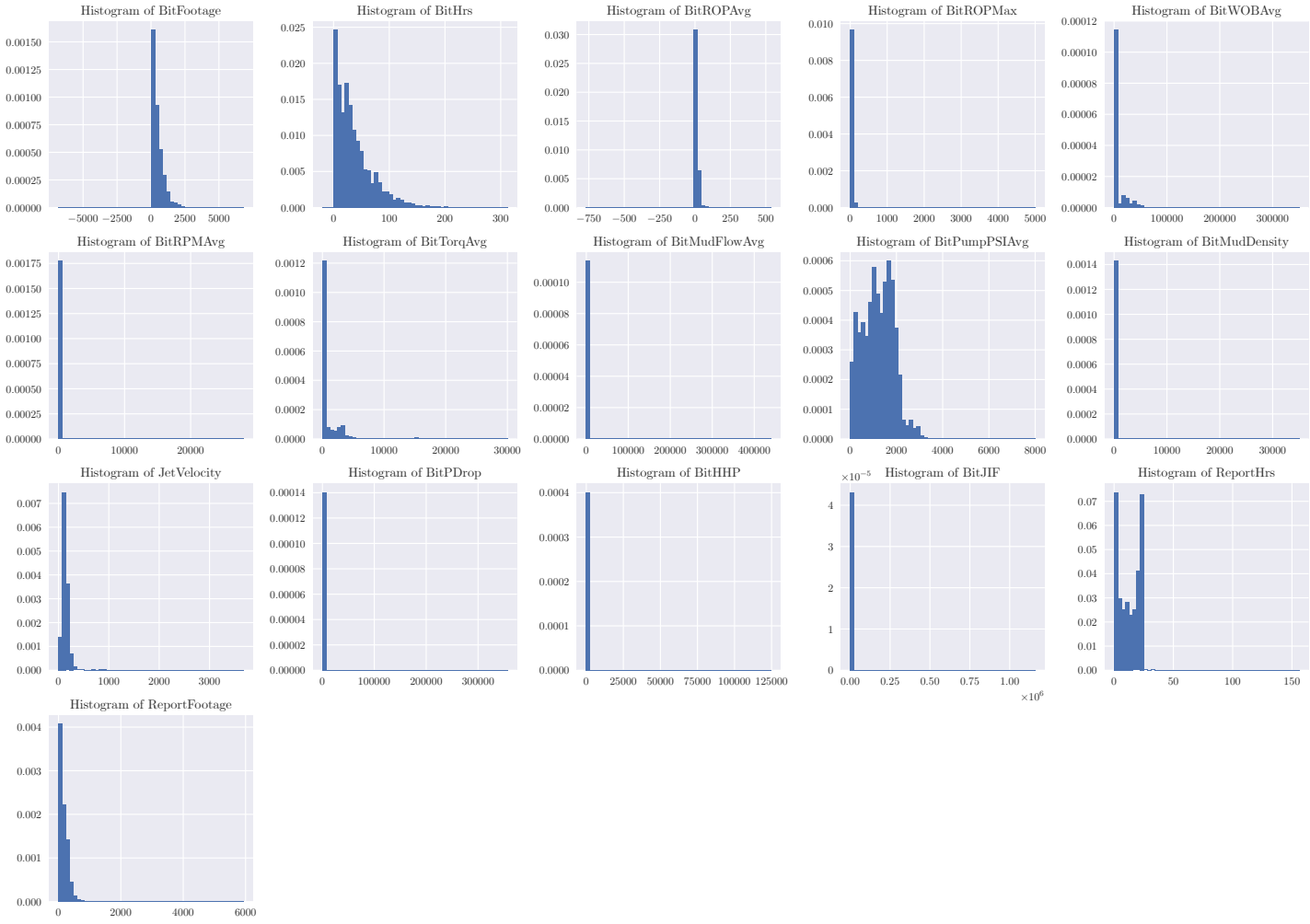

Figure 3.2: Histogram of `dailybitinfo` data

cleanly. This is a good news for later analyses because dropping rows with missing values is a valid strategy for some ML methods, and in this case dropping does not destroy the structures of the dataset. Another interesting feature from Figure 3.7 and Figure 3.8 is the amount of separation between `dailydrilldetail` data and `dailybitinfo` data. Each geothermal field is more cleanly delineated in the t-SNE using `dailybitinfo` compared to the one from `dailydrilldetail`. This indicates the data collected from `dailybitinfo` is of higher quality for ML modeling purpose. However, each geothermal field can be easily identified from t-SNE graphs from `dailydrilldetail` data, showing that there is a strong underlying data structure even in `dailydrilldetail` data. Finally, the results from t-SNE analysis confirmed that there is data sufficiency for further ROP modeling.

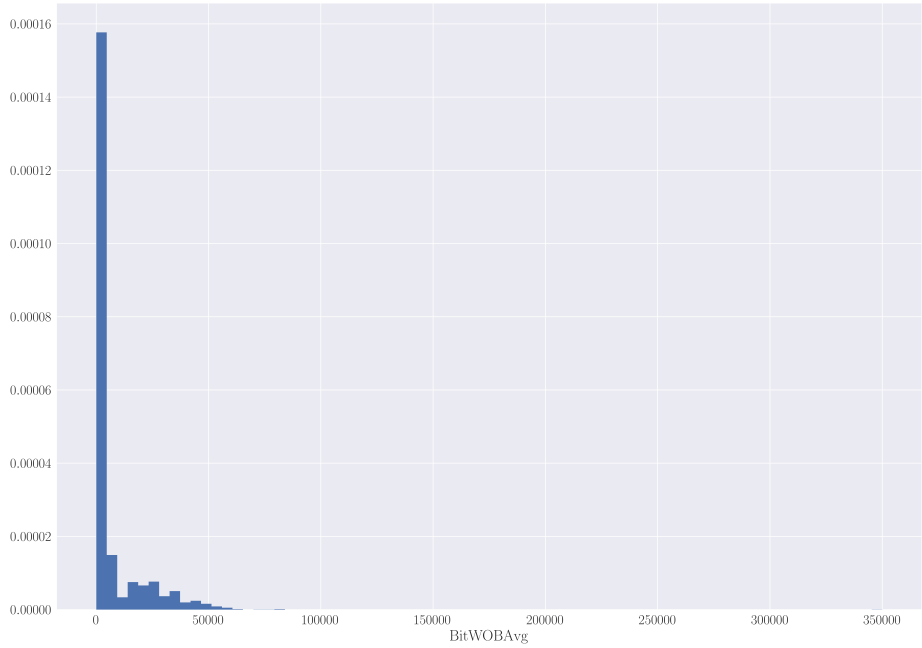

Figure 3.3: Histogram of BitWOBAvg data

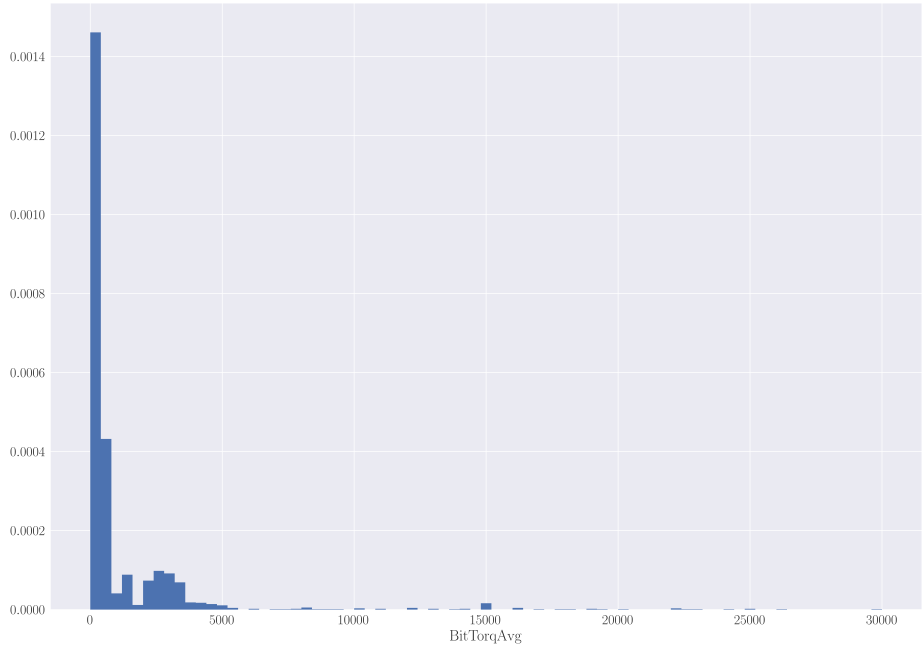

Figure 3.4: Histogram of BitTorqAvg data

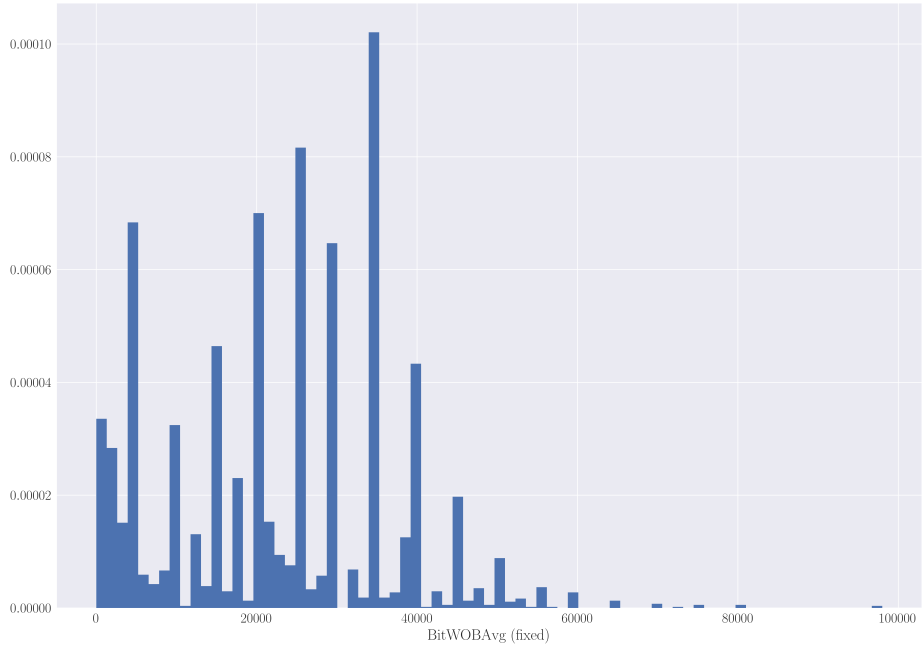

Figure 3.5: Histogram of BitWOBAvg data (fixed)

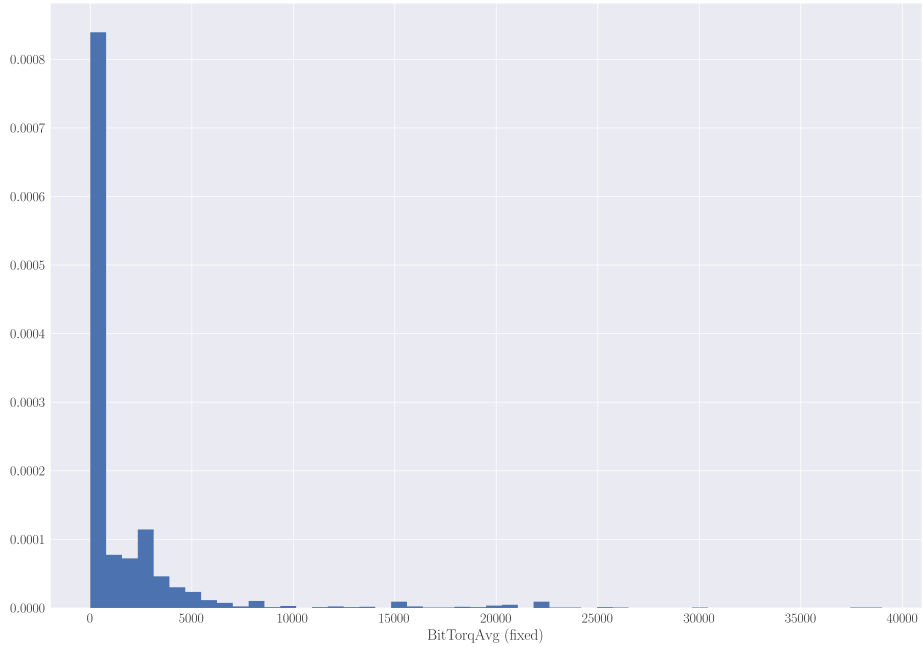

Figure 3.6: Histogram of BitTorqAvg data(fixed)

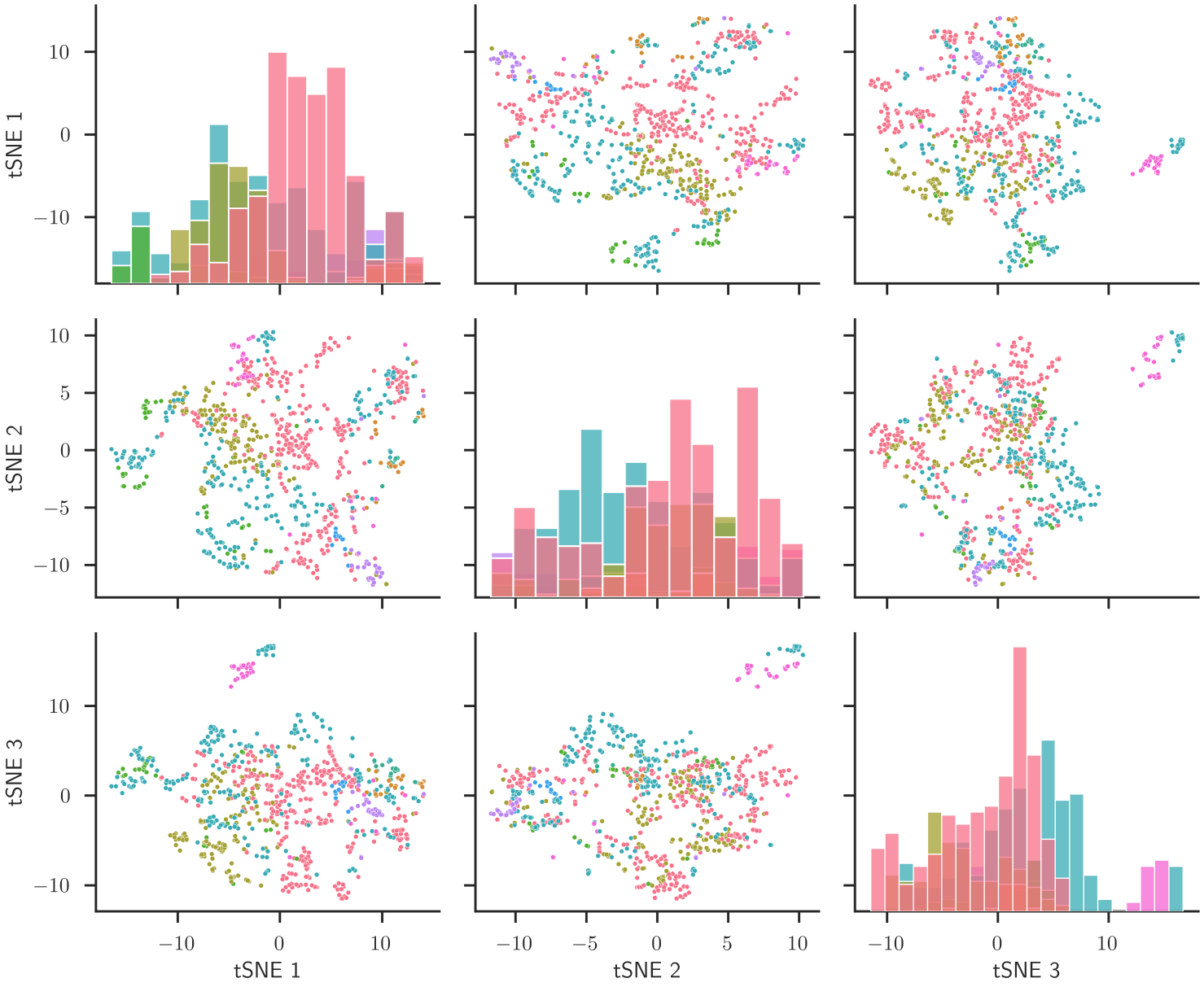

Figure 3.7: t-SNE results on `dailydrilldetail` data. Each dot represents a daily datapoint and each color represents a different geothermal field

3.3 Nonnumerical data

In additional to standard drilling records described above, there are also nonstandard drilling records in the database. These records are in textual information format, rather than numerical format and need to be processed separately. One such feature that carries important drilling information is

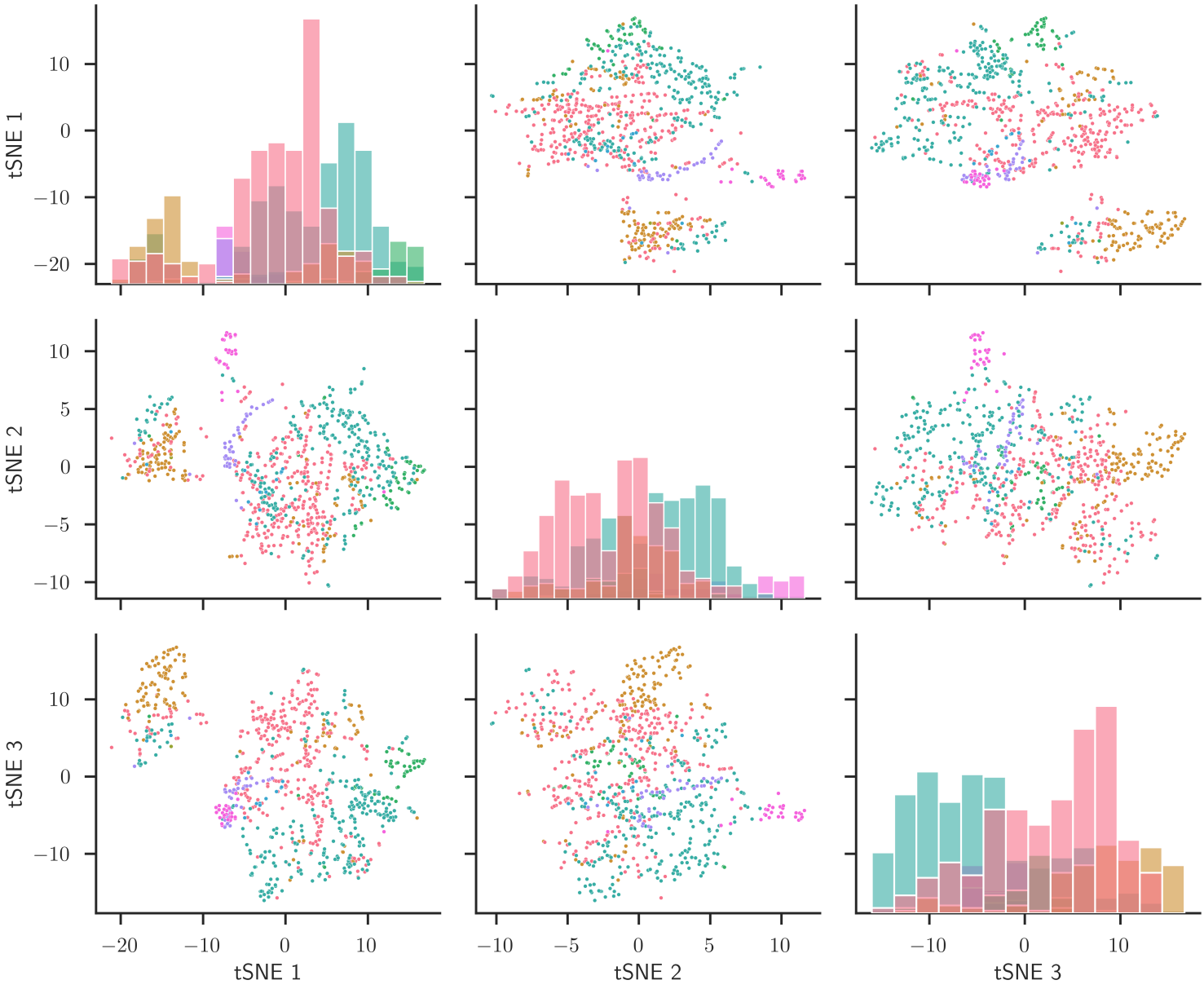

Figure 3.8: t-SNE results on `dailybitinfo` data. Each dot represents a daily datapoint and each color represents a different geothermal field

the "OpsGroup" column. The "OpsGroup" column encodes what kind of drilling operations happen during a drilling day, from a predefined categorical format: "DRILL" for normal drilling, "TRIP" for tripping, "PROBLM" for problems, and "OTHER" for other operations (i.e. "DRILL"-"DRILL"-"TRIP"-"TRIP"). This feature is important as the content of the daily operations are not transparent

from the daily average numerical records alone (i.e. it is hard to infer whether trippings occur or not solely from the numerical records). This will also serve as indicator of nonproductive times, which sometimes are correlated to significant cost-saving opportunities as discussed in Chapter 2.

Additionally, the records also contain daily remarks from the drillers in the "Description" column. This feature includes remarks from drillers which contain detailed descriptions of what happened that day that cannot be easily described using numbers. There are four different categories that these remarks belong to, depending on the purposes of the remarks (Table 3.5 and Table 3.6).

Table 3.5: Drilling remarks' categories

Remark category	Purposes
CurrentOps	Records of current drilling operations
FutureOps	Tentative lists of future operations
Daily Comment	Comments from the drillers about any current drilling operation
MgmtSummary	Summary and remarks from rig managements

Table 3.6: Sample of remarks from the "Description" column

Remark category	Example
CurrentOps	Drill 12 1/4" hole from xxxx to xxxx using mud pump 2; worked on mud pump 1; replaced 4 seats & valves. Drilled 12 1/4" hole from xxxx to xxxx
FutureOps	Recover fish, make up and inspect new bottom hole assembly, run in hole and continue drilling 12 1/4" hole f/ xxxx
Daily Comment	No mud loss. No new fractures. Rig up drag at xxxx - 200,000. Rig still pulling good
MgmtSummary	No problems with 20" casing run. NO fill on bottom

As pointed out in Chapter 2, the lithology of the drilling formations is indispensable in creating an accurate model. However, accurate lithological information is hard and expensive to acquire in advance. Therefore, other sources of information are usually used as proxies for lithology information. In our dataset, the rock compositions and properties from the mud-shakers, which record a "Litho" column, can serve as a proxy. However, the problems of processing textual data encountered with the "Description" column also apply to the "Litho" column . In fact, the records from mud-shakers are even harder to understand compared remarks from the drillers (Table 3.7).

Table 3.7: Sample of daily records from the "Litho" column

40-60% Phyllite, 40-60% Silstone and Clay
xxxx-xxxx 0-20% Phyllite 20-40% Siltstone 40-80% Clay, xxxx-xxxx 0-20% Qtz Vng 30-50% Phyllite 20-50% Silstone 20-40% Clay, xxxx-xxxx 0-20% Phyllite 20-40% Silstone 40-60% Clay
Mod ylsh brn; firm to hd; sbrnd ctgs; aphnc w rd glassy incl; com qtz & calct fill amyg; tr clystn
Rhyolite/ Basalt; lt-med gry, prplish gry, hd, subblky-subang ctngs, aphnc-sli porh, (qtz, calct, chalcedony) fillied amyg

It will be very beneficial for the modeling process if the model can incorporate textual information in additional to the conventional numerical information. However, despite the wealth of information that the textual data contains, extracting that information is not easy as the remarks are records in written English with no standardized structure. Conventionally, manual processing is the only way to extract useful information out of the remarks, which is highly time, and resource, consuming. Even then, it is hard to incorporate the extracted information into the ML model to use together with other drilling records. Hence, these kinds of textual data are very often ignored in ML modeling. This study will discuss about how to incorporate these remarks effectively without manual processing in Chapter 5.

Chapter 4

Modeling with Deep Neural Networks

4.1 Introduction

From 2010 onward, deep neural network (DNN) methods have gained remarkable improvements, and sometimes outperform humans in some tasks. From the humble beginning of AlexNet (2012) with merely 66 million parameters to the modern GPT-3 (2020) with more than 17 billion parameters, DNN sizes have grown astoundingly in the last decade, and so have their capabilities. AlexNet started out with simple object classification tasks, but now GPT-3 is capable of holding natural language conversation with humans[12]. This remarkable advance strongly corresponds to how complex the network is, i.e., the number of layers in the network.

However, simply stacking layers upon layers does not really make a DNN powerful. DNNs are notoriously hard to train properly, coupled with the fact that adding more parameters to the network makes the network tend to overfit the training dataset. It took research and experimentation to arrive at modern DNN architecture.

This chapter discusses the dual-path dual-branch deep residual neural network (DPDBN) architecture used in this study: the history, the mathematical theory, and the evolution to the DPDBN architecture. In addition, this chapter also discusses about the methodologies and the results of using DPDBN in well optimizations.

4.2 Dual-path dual-branch deep residual neural network

The DNN AlexNet (2012) was the 2012 winner of object classification task ImageNet (ILSVRC competition) with a predictions error rate of 15.3%, 10.8% higher than the runner-up[13]. It was a sensational result at that time, where a neural network model outperformed sophisticated hand-crafted feature statistical models. The authors cite that it is the depth of 8 layers that enables AlexNet to achieve high performance. This was also the trend for the DNN industry for a while, where progressively deeper neural network architectures achieved higher accuracy in ImageNet predictions.

Despite the vast differences in architecture, most neural networks must rely on back-propagation to be trained efficiently. However, the fundamental problem of vanishing/exploding gradients of back-propagation hampers the convergence of DNN from the beginning. To understand this problem, consider a L layers conventional feed-forward neural network; at the ℓth layer, the weight is W^{ℓ}, and the activation function is f^{ℓ}. Let the input and true output pair be x and y, then the output of the neural network $g(x)$ is:

$$g(x) = f^{[}W^{L} \cdot f^{L-1}[W^{L-1} \cdot ...f^{1}[W^{1} \cdot x]...]] \tag{4.1}$$

Given a criteria function C, the errors term of the neural network respects to true output y is:

$$C(y, g(x)) = C(y, f^{L}[W^{L} \cdot f^{L-1}[W^{L-1} \cdot ...f^{1}[W^{1} \cdot x]...]]) \tag{4.2}$$

Denote the output of the ℓth layer as a^{ℓ}, then from chain rule, the gradient of criteria with respect to x is:

$$\nabla_{x} C(y) = (W^{1})^{T} \cdot (f^{1})' \cdot (W^{2})^{T} \cdot (f^{2})' \cdot ...(W^{L-1})^{T} \cdot (f^{L-1})' \cdot (W^{L})^{T} \cdot (f^{L})' \cdot \nabla_{a^{L}} C(y) \tag{4.3}$$

Or, at the ℓth layer, the gradient of the weight can be computed recursively as:

$$\nabla_{W^{\ell}} C(y) = (f^{\ell})' \cdot (W^{\ell})^{T} \cdot \nabla_{W^{\ell+1}} C(y) \tag{4.4}$$

From Equation 4.4, it easy to see that the gradient at the ℓth layer is the product of $(f^{\ell})'[(W^{\ell})^{T}]$. Unless the values of all $(f^{\ell})'[(W^{\ell})^{T}]$ are close to 1, the value of $\nabla_{W^{\ell}} C(y)$ can be exceeding large or exceeding small, and the problem worsens with larger L. This behavior is called the exploding/vanishing gradients problem.

Due to the exploding/vanishing gradients problem, training a DNN with a large number of layers is notoriously hard. A lot of efforts have been made to remedy this problem. Normalized initialization and intermediate normalization[14], whose goal is to bring the values of all $(f^{\ell})'[(W^{\ell})^{T}]$ in

Equation 4.4 close to 1, are the most typical methods used to fix the gradients problem, which was considered largely addressed by 2015. However, when deeper networks start converging, another serious problem was discovered; the network performance decreased rapidly with depth despite proper convergence. Overfitting is not the cause for such degradations as the network performance on both the training and validation sets degrade (Figure 4.1). This indicates that the training process is not to blame, but there is a architectural problem with the conventional DNNs themselves that prevents these networks from training.

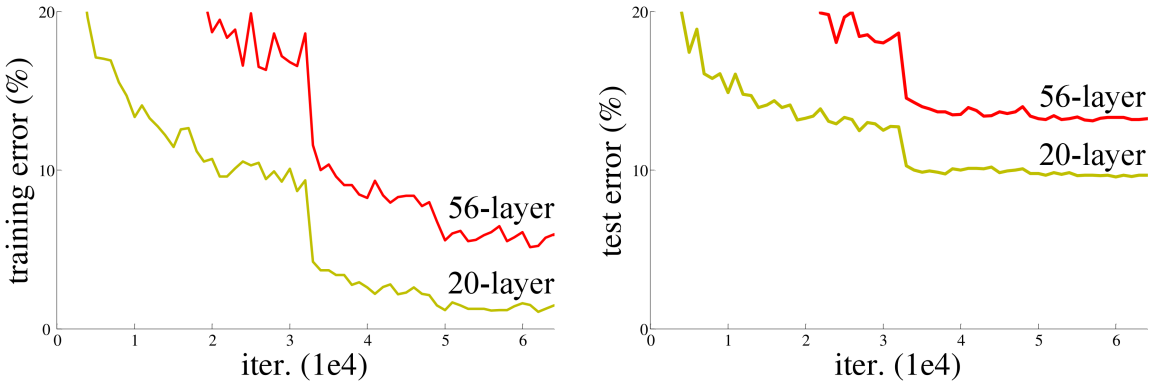

Figure 4.1: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer conventional DNN[1]

Similarly, there are also efforts in creating alternative architectures to remedy the problem with conventional DNN. On of the early successes is with the winner of ILSVRC 2014: GoogleNet[15], which utilizes a Network-in-Network structure. A breakthrough came in 2015 with development of Deep Residual Neural Network (ResNet)[1], which was the winner of ILSVRC 2015 by a wide margin. ResNet, together with its proposed deep residual learning framework, enables researchers to create and train neural networks hundreds of layers deep, while still maintaining superior performance. The deep residual learning framework is also the foundation of the dual-path dual-branch deep residual neural network used in this study.

The key idea of deep residual learning framework is that conventional deep networks performances degrade because they struggle to learn identity mapping due to multiple layers of nonlinearity mapping. If this hypothesis holds true, then a solution is rather than ask the network to find the direct mapping function $f : f(x) = y$, the network is asked to find the residual mapping $h : h(x) = y - x$. By asking the network to find the residual mapping, if the identity mapping is the optimal mapping then the network simply drives the weight to zero to approximate the mapping. More specifically, consider x is the input and $h(x)$ is the output of a one/many layer(s) neural network block, then a

building block of the deep residual network is defined as:

$$y = h(x) + x \tag{4.5}$$

where + denotes element-wise addition.
The dimension of $h(x)$ has to match the dimension of x. If the dimensions are not matched, then the building block is:

$$y = h(x) + W_s \cdot x \tag{4.6}$$

where W_s is a linear mapping so that $W_s \cdot x$ and $f(x)$ have the same dimensions.

The deep residual network architecture is composed of many residual building blocks defined in Equation 4.6 (Figure 4.2) stacked on top of each other. The original author has successful trained a 152-block deep residual convolution neural network, each block containing three convolutional layers, for the ILSVRC 2015 competition. The results were excellent, just like AlexNet brought forward in 2012, as seen in Table 4.1. ResNet-152 outperformed other state-of-the-art neural networks by a significant amount on the ImageNet dataset. The result is also an important proof for the deep residual learning framework hypothesis. Although originally used for image classifications, deep residual neural network architecture can be readily modified to accept tabular information, like that used in this study.

Table 4.1: Top-5 prediction errors rate ImageNet validation set of some neural network architectures

Network	Top-5 prediction errors rate (%)
VGG (ILSVRC 2014 Runner-up)	8.43
GoogLeNet (ILSVRC 2014 Winner)	7.89
ResNet-34	5.60
ResNet-50	5.25
ResNet-101	4.60
ResNet-152 (ILSVRC 2015 Winner)	4.49
DenseNet-264 (2016)	5.29
DPN-131 (2017)	4.16

However, the residual neural network is not the only successful attempt to circumvent the performance degradation problem. Another successful deep neural network is Densely Connected Neural Networks (DenseNet)[16], which is based on a different but similar hypothesis. DenseNet architecture is based on the hypothesis that when a network becomes increasingly deep, any information about the inputs, after passing through many layers, vanishes by the time it reaches the end of the network (or the beginning in backpropagation). ResNet remedies this problem by creating shortcuts that allow information to flow freely from the beginning to the end of the network. DenseNet argues

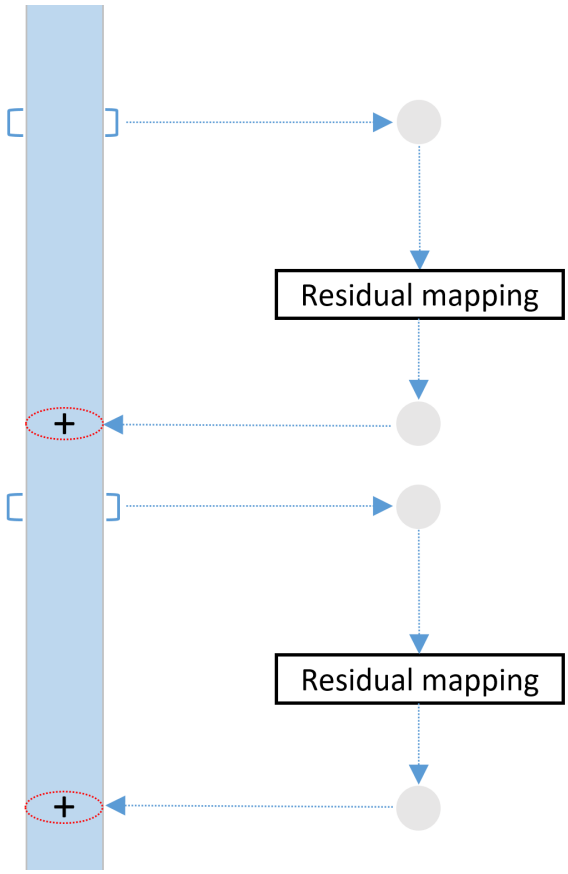

Figure 4.2: Deep Residual Network

that rather than using roundabout methods to create these shortcuts, the network should have these shortcuts built-in explicitly. DenseNet connects all layers directly with each other to ensure information flows between layers as seen in Figure 4.3. Different from ResNet, DenseNet never combines features with summation before passing them to another layer; these features are simply concatenated with each other before passing.

More specifically, consider the layer ℓ of densely connected neural networks, denoting the outputs of all previous layers as $z_1, z_2, z_3, ..., z_{\ell-1}$, then the output of the ℓth layer is:

$$y_\ell = f([z_1, z_2, z_3, ..., z_{\ell-1}]) \tag{4.7}$$

where $f(x)$ is the mapping function of the ℓth layer and $[z_1, z_2, z_3, ..., z_{\ell-1}]$ is the concatenation of all previous layers features.

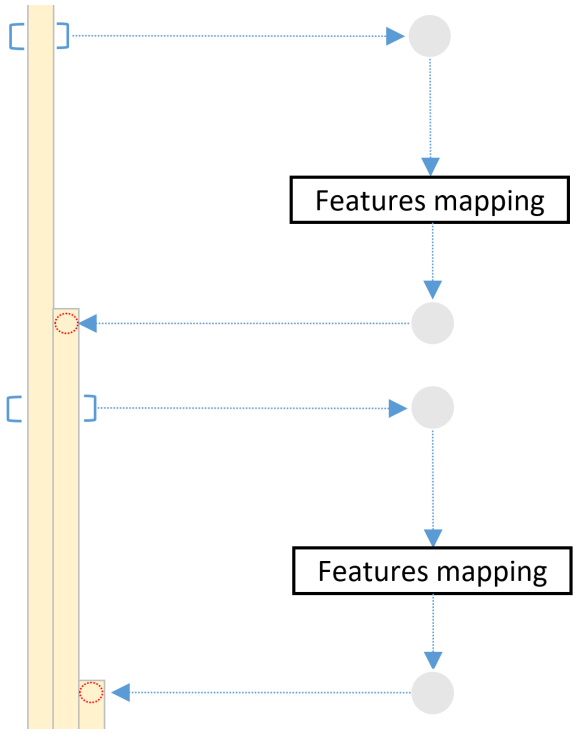

Figure 4.3: Densely Connected Network

Densely connected neural networks are also composed of basic building blocks (Figure 4.3) stacked on top of each other, similar to the ResNet architecture. One important parameter of DenseNet, in addition to depth, is growth rate k of the network. The growth rate is defined as follows: a DenseNet has a growth rate of k if the ℓth layer has $k_0 + k(\ell - 1)$ feature maps, where k_0 is the number of feature maps in the initial layer. A simple way to understand the role of growth rate k is to consider all the feature maps in the network is the global state of the network, then each layer contributes k feature maps to this global state. Due to these densely connected layers, the network's global state can be accessed from anywhere in the network, eliminating the need for feature maps replications in traditional feed-forward network.

Despite the seemingly large differences in architectures, DenseNet is conceptually similar to ResNet and able to produce a similar result to ResNet on the ImageNet dataset (Table 4.1). One reasonable explanation for this is that both ResNet and DenseNet help increase the parameter utilization rates; there is no need to relearn a feature map if it is previously learnt. ResNet makes these known feature maps accessible implicitly through identity additions, while DenseNet make them available explicitly. However, each approach also has its drawbacks: ResNet is good at finding and reusing dominant feature maps, but bad at finding new feature maps; DenseNet on the other hand

tends to explore many feature maps, but suffers from redundancy as newly explored feature maps may be very similar to the known ones.

Naturally, due to the similarities and drawbacks, there were efforts to try to combine the ResNet architecture with DenseNet architecture, and one of the more successful attempts is with the Dual Path Networks (DPN)[17]. The DPN architecture not only has the advantage of ResNet where feature maps are reused efficiently and also the ability to explore new feature maps of DenseNet. More specifically, in the DPN architecture, the ℓth building block $F_\ell(x)$ has the form:

$$\begin{aligned}
&\text{Let:}\\
&x_\ell = [z_1, z_2, z_3, ..., z_{\ell-1}, y_{\ell-1}]\\
&r = f_\ell(x_\ell)\\
&F_\ell(x_\ell) = [z_\ell, y_\ell]\\
&\text{Then:}\\
&[z_\ell, y_\ell] = [g_\ell(r), y_{\ell-1} + h_\ell(r)]
\end{aligned} \tag{4.8}$$

where $[z_\ell, y_\ell]$ are the two outputs of the ℓth building block, f_ℓ, g_ℓ, h_ℓ are subsequently the common path, the individual path, and the residual path mapping functions.

It is easy to see from Equation 4.8 that there are two distinct flows of information in DPN. Information can flow from layer to layer through y_ℓ, which goes through an identity addition transformation after every block, or through $[z_1, z_2, ..., z_\ell]$, where a new mapping z_ℓ is added after every block. If all z_ℓ of a DPN are discarded, then the resulting network is a ResNet, whereas if all y_ℓ are ignored, the resulting network will be a DenseNet. Hence, y_ℓ resemble for residual connections the enable feature maps reuses, and z_ℓ resemble densely connected paths that enable efficient feature map explorations. By using both z_ℓ and y_ℓ as input of any building block, DPN was able to achieve superior results on the ImageNet dataset compared to both ResNet and DenseNet despite having fewer parameters (Table 4.1).

While a neural network architecture is vital to its performance, there are also other considerations in order to achieve good results. One of most important criteria when designing a machine learning model is its ability to generalize, a machine learning model is only useful if it is able to give good predictions on unseen datasets. This is undoubtedly true with neural networks, which have far more learnable parameters than traditional machine learning methods. This is the common explanation for why neural networks tend to be over-fitted. While it is easy to come up with a neural network that predicts well only on the training dataset (memorization of the dataset), developing a network that gives good predictions on both seen and unseen dataset is hard. Unfortunately, there is not a concrete theory to explain why some neural network architectures generalize exceptionally well

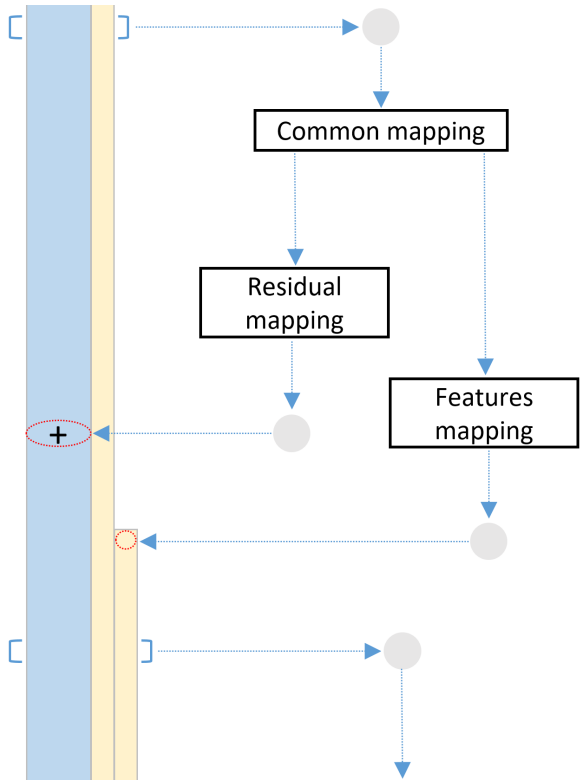

Figure 4.4: Dual Path Network

while some do not. However, when the total number of learnable parameters is large, some form of regularization is needed to avoid poor generalizations.

The uses of regularization has accompanied the development of machine learning in general. In addition to the architecture used, a proper regularization scheme can also greatly improve the final predictions of a model. In classical machine learning, the most common methods for regularization include L_1, L_2 regularizations and early stopping. L_1 and L_2 regularizations both constrain the growth of the model's coefficients in order to avoid over-fitting. However, although they work well in classical machine learning, L_1 and L_2 regularizations are considered harmful in neural networks and are seldom used. On the other hand, early stopping stops the training process before the model becomes over-fitted. Unfortunately, it is not easy to use in conjunction with neural network models as these models can have complicated training processes where over-fitting is not easy to identify.

Because of the needs for regularizations in neural networks, specific methods have been developed. In conventional feed-forward neural networks, the standard method for regularizations is using Dropout [18], where in each training iteration, some randomly selected nodes in the network are

zeroed out. This means a random portion of the network is deactivated and does not contribute to the training process. Through the Dropout process, random noises are added into the information flow of the neural network, and consequently into its gradients during training. Dropout reduces the dependency of each neuron to the input data and the outputs of other neurons, which results in better generalization of the model. In addition, a side effect of Dropout noises is they help the optimizers to get out of local minima, and to explore "difficult to reach" parts of the loss surface that are not easily accessible normally. Despite that, due the specific natures of ResNet, Dropout is not as helpful and sometimes undesirable. A more specific regularization scheme should be employed when dealing with ResNet and its derivatives. A regularization scheme specifically designed for ResNet, called Shake-Shake regularization, was used in this study.

Shake-Shake regularization is an attempt to improve the generalization ability of multibranch neural networks (ResNet, DenseNet, DPN, etc.) by introducing stochastic elements into the network[19]. The original method is based on ResNet architecture, but can be modified to accommodate other multibranch neural network models. In Shake-Shake regularization, for the ℓth building block, instead of only one residual mapping $h(x)$ (Equation 4.6), there are two residual mappings (or branches) $h_1(x)$ and $h_2(x)$. Let α and β be two variables sampled from a uniformly random distribution, then Shake-Shake regularization is defined as:

$$y(x) = \begin{cases} x + \alpha h_1(x) + (1-\alpha)h_2(x), & \text{in forward phase} \\ x + \beta h_1(x) + (1-\beta)h_2(x), & \text{in backpropagation phase} \\ x + \mathbb{E}[\alpha]h_1(x) + \mathbb{E}[1-\alpha]h_2(x), & \text{in validation} \end{cases} \tag{4.9}$$

Normally, the range of the uniform random distribution that α and β are sampled from is chosen as $[0, 1]$, which results in $\mathbb{E}[\alpha] = \mathbb{E}[1-\alpha] = 0.5$.

In forward calculation of a ResNet with Shake-Shake regularization, the residual mapping is the sum of two separate residual branches with a fuzzy number α. The purpose of α is to let the two evolve independently rather than converging to the same residual mapping during the training phase. The result is that the final residual mapping is the average of two different residual branches, which hopefully increases the generalization abilities of the network. Normally, during the backpropagation phase, if a branch is multiplied by α then its gradient will also be scaled by α. However, the gradient here is scaled by β (or $1-\beta$) rather than α (or $1-\alpha$). This has the effect of introducing noises into the gradients during the backpropagation phase, which helps the optimizing process, similarly to what Dropout does.

In this project, a neural network architecture, called dual-path dual-branch neural network

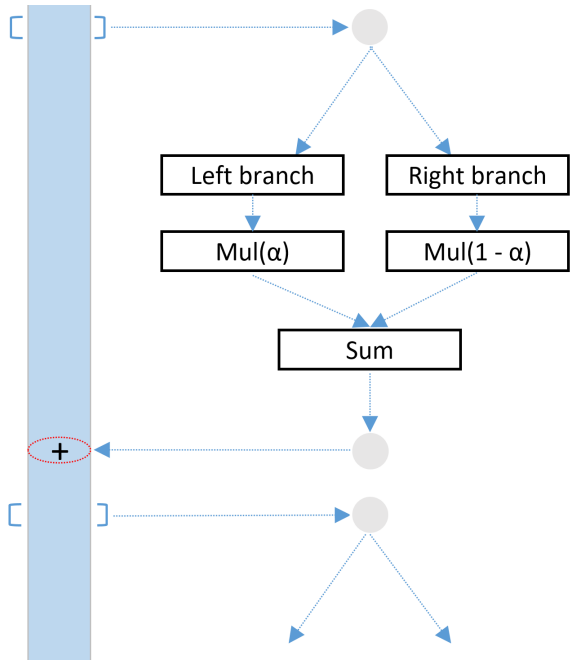

Figure 4.5: Deep Residual Network with Shake-Shake regularization

(DPDBN) was used. DPDBN is a modified DPN architecture that includes Shake-shake regularization. In DPDBN, each ℓth building block $F_\ell(x)$ has twice the amount of mapping functions compared to Equation 4.8. More specifically, f_ℓ^1 and f_ℓ^2; g_ℓ^1 and g_ℓ^2; h_ℓ^1 and h_ℓ^2 are subsequently the two branches of the common path, the individual path, and the residual path mapping functions, the ℓth building block $F_\ell(x)$ can be described as:

Let:
$x_\ell = [z_1, z_2, z_3, ..., z_{\ell-1}, y_{\ell-1}]$
$r^1 = f_\ell^1(x_\ell); r^2 = f_\ell^2(x_\ell)$
$F_\ell(x_\ell) = [z_\ell, y_\ell]$
Then:

$$[z_\ell, y_\ell] = \begin{cases} [\alpha g_\ell^1(r^1) + (1-\alpha) g_\ell^2(r^2), y_{\ell-1} + \beta h_\ell^1(r^1) + (1-\beta) h_\ell^2(r^2)], & \text{in forward phase} \\ [\gamma g_\ell^1(r^1) + (1-\gamma) g_\ell^2(r^2), y_{\ell-1} + \delta h_\ell^1(r^1) + (1-\delta) h_\ell^2(r^2)], & \text{in backpropagation phase} \\ [\mathbb{E}[\alpha] g_\ell^1(r^1) + (1-\mathbb{E}[\alpha]) g_\ell^2(r^2), y_{\ell-1} + \mathbb{E}[\beta] h_\ell^1(r^1) + (1-\mathbb{E}[\beta]) h_\ell^2(r^2)], & \text{in validation} \end{cases} \tag{4.10}$$

where α, β, γ, and δ are variables sampled from a uniformly random distribution in the range $[0, 1]$.

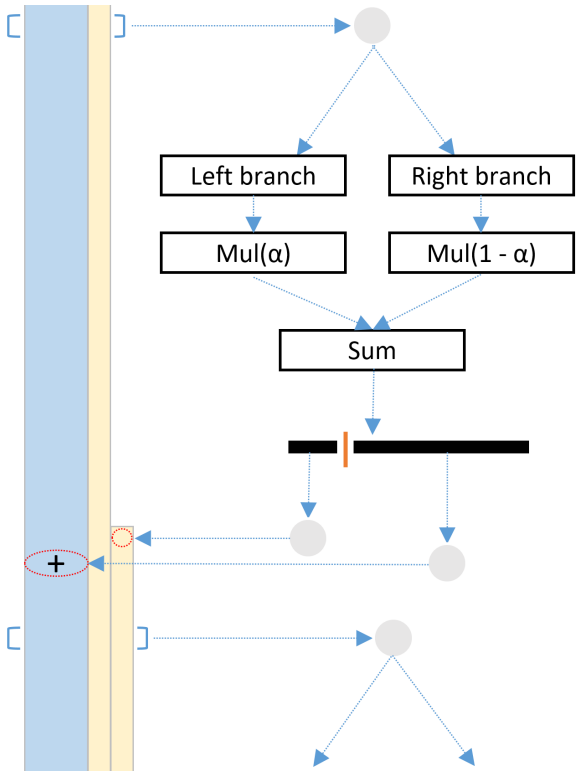

Figure 4.6: Dual-path dual-branch Network

4.3 Methodology

4.3.1 ROP modeling

As discussed in Chapter 2, the ML model developed to help drill a successful well will have ROP as the main criterion. For that purpose, a ML model based on DPDBN has been developed. The target is to create a ML model that is able to accurately predict ROP from typical drilling records.

The daily-averaged drilling records described in Chapter 3 serve as inputs, and the daily-averaged ROP will be the target. However, these inputs and outputs cannot be used directly as they are due to multiple problems in the original records, as uncovered in the second section of Chapter 3. Therefore, some data preprocessing works are needed to make the data ready for the later modeling part. The data preprocessing work done on the inputs includes the follow steps:

1. Take care of improper values
2. Fill missing values or extreme outliers with `nan`
3. Scale the features to have mean of zero and standard deviation of one

4. Create the feature mask
5. Replace missing values or extreme outliers with zero
6. Concatenate scaled data with the feature mask to create the final input

The first step is described in detail in Chapter 3. In addition, to prevent data leaking from the inputs to the outputs, some features must be removed from the inputs. The removed data includes features that record footage or total drilling time of the bits (`BitFootage`, `BitHrs`, etc.). In step two, any missing data, or any extreme outlier deemed by the previous step, is replaced with a `nan` value. The purpose of this step is to remove the influences of any extreme outlier, and make the data easier to work with in the later steps. The third step is scaling the data, which is a requirement for most neural network models. One of the most common scaling methods is standard scaling, which transforms the data so that it has the mean of zero and standard deviation of one feature-wise. Standard scaling is also the chosen scaling method for this project due to its effectiveness and simplicity. After scaling, a feature mask is created. A feature mask is a table which has the same dimension as the scaled data, where a value of one in the table indicates a missing/outlier value in the original data, and value of zero indicate the data is present. The purpose of the features mask is to inform the locations of missing data so the model can react accordingly. Using a feature mask is a strategy that helps a ML model deal with incomplete information, which can also be used to describe the dataset used in the project. Without this feature mask, any row with missing data has to be dropped, which may reduce the size of data by up to 75% (Table 3.4). Finally, the input is the simple concatenation of the scaled-data and the feature mask.

The same process repeats for preparing the outputs, albeit with a few small differences: there is only one feature, and rather than using a feature mask, missing data or extreme outlier (and the corresponded row in the inputs) are dropped. A subset of the inputs is shown in Table 4.2.

Table 4.2: Sample of the inputs used for ROP modeling

BitWOB Avg	BitRPM Avg	BitTorq Avg	BitMud FlowAvg	...	Mask_1	Mask_2	Mask_3	Mask_4	...
-1.69918	0	-2.13653	0	...	0	1	0	1	...
-1.17123	0	-1.98401	0	...	0	1	0	0	...

Considering the purpose of this model, which is to help drillers to model and maximize the ROP, there are two different approaches to construct such model. The estimate can either try to model the current ROP using the current drilling record from the rigs, so that the drillers can use the model as an anomaly detector when there is a big discrepancy between true ROP value and the predicted ROP value, and the model can also help the drillers to maximize the ROP by allowing them to

adjust the input parameters and observe the response from the model. In the second approach, the model can try to predict the next day ROP from the current drilling record, giving drillers ample warning times for potential problems. For the first approach, the model will be trained with drilling record at time t and asked to predict the ROP at time t, while for the second approach, the model will be trained with drilling record at time $t-1$ and asked to predict the ROP at time t. This project considered the latter approach.

Conventionally in machine learning, the training set and the validation set are constructed by randomly selecting datapoints from the original dataset. This is to ensure that both the training and the validation data have the same underlying distributions, which is an important assumption when using them to gauge the network performances/generalizations. However, considering how a ROP model would be used in drilling, the model is trained on drilling records of known wells so that it can accurately model the ROP of any new similar well in the same area. This means that a random train/validation splitting scheme is not suitable as there is no information about the new well. To deal with this problem, another train/validation splitting scheme must be used; rather than splitting the information randomly, the train/validation is chosen well-by-well (i.e. there are drilling records of well `10`, `11`, `12`, `13`, `14`, and `15`; then the records of wells `10`, `11`, `12`, and `13` will be included in the training set while the records of wells `14`, and `15` will be included in the validation set). This train/validation splitting scheme mimics how drillers would use a ROP model in production. This project considered both approaches.

With the inputs and outputs clearly defined, the next step is to train the neural network model on the dataset. The DPDBN used in this study is constructed with the parameters described in Table 4.3.

Table 4.3: DPDBN parameters for ROP modeling

Parameter name	Value
Number of elementary block, n_b	15
Growth rate, k	5
Number of elementary block n_t	3
Activation function	`ReLU`
Total number of parameters	2595904
Loss criteria function	Huber loss

The common path, the individual path, and the residual path mapping functions f_ℓ, g_ℓ, and h_ℓ in each branch of the DPDBN model is simply a feed-forward neural network with three layers and two `ReLU` activation functions in between. The network will be trained on the dataset until over-fitting occurs (early stopping), and the results are the average of ten individual runs.

4.3.2 Non-ROP modeling

As discussed at the end of Chapter 3, there are other important parameters to the success of a well. Constructing a model that can give future warning about potential problems and nonproductive times is also beneficial to the drillers. Rather than ROP, this model will predict what operations will happen tomorrow (the "OpsGroup" feature). In this project, a model to predict the two leading causes of nonproductive times, "TRIP" and "PROBLM", was developed. This model takes the drilling record at time $t-1$ and predicts whether "TRIP" or "PROBLM" happens at time t. This will give drillers ample warning time to prevent/remediate potential nonproductive periods. For this purpose, the inputs, the train/validation splitting scheme, and the models are similar as in the ROP modeling process. The only difference is the output, which will be coming from the "OpsGroup" column.

4.4 Results

4.4.1 ROP modeling

From Figure 4.7, it is easy to see that in random train/validation splitting, the ROP model is doing reasonably well on the drilling records dataset. The model achieves an R^2 score of 0.56 on the training set and 0.54 on the validation set. The 95% confidence interval of the model on Figure 4.8 indicates good prediction quality.

However, switching from random to well-by-well train/validation splitting has a detrimental effect on the results. The model only achieves an R^2 score of 0.34 on the training set and 0.10 on the validation set (Figure 4.9). The 95% confidence interval of the model on Figure 4.10 shows the high uncertainties in the predictions (i.e. a real ROP of 10 ft/hr is often predicted to be in the range of 5 ft/hr to 15 ft/hr).

It is clear from the result that even with an identical machine learning model, using random train/validation splitting produced a model that is accurate enough for use in production, while using well-by-well train/validation splitting produced a model with subpar prediction quality. Unfortunately, as discussed before, the model trained on random train/validation splitting data has limited use in production despite it superior accuracy as it requires knowledge from the future. On the contrary, the model trained on well-by-well train/validation splitting has use in production, but its accuracy was not sufficient.

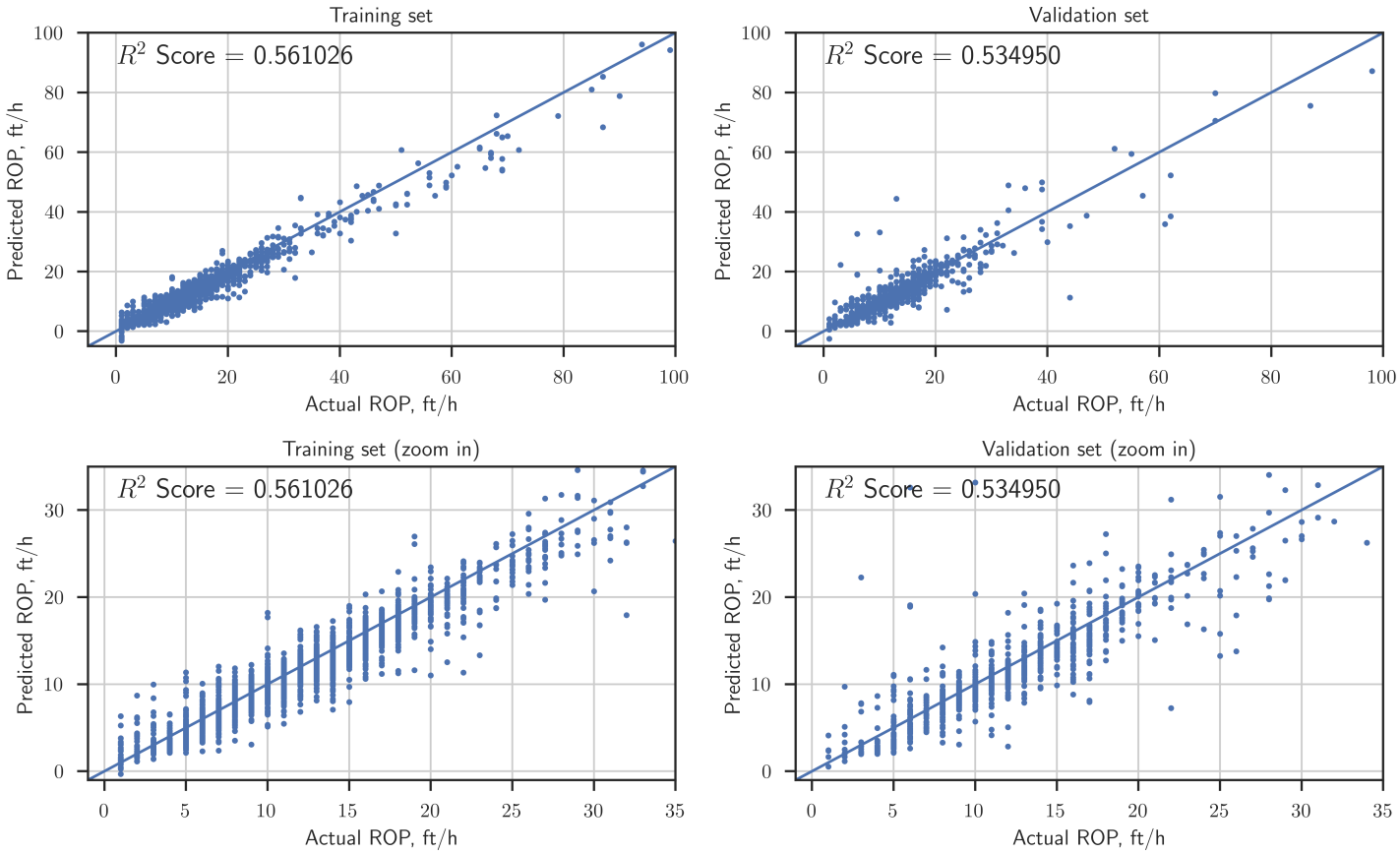

Figure 4.7: Correlation plot for ROP modeling, random train/validation splitting

There is a possible explanation for the discrepancies between the two models. Random train/-validation splitting is often used as it is an easy method to make sure that the training dataset and validation dataset have similar underlying distributions. On the other hand, it is shown that ROP modeling depends strongly on the lithological properties of the drilling formation (Chapter 2), hence only wells in the same field are chosen in well-by-well train/validation splitting as they have high probability to have similar geological properties. However, there are discrepancies between the two models, indicating that the training dataset and validation dataset have different underlying distributions, even though they belong to the same field. If that were that case, then a better result can be obtained by selecting wells that have similar geological properties, or by adding lithological information into the inputs.

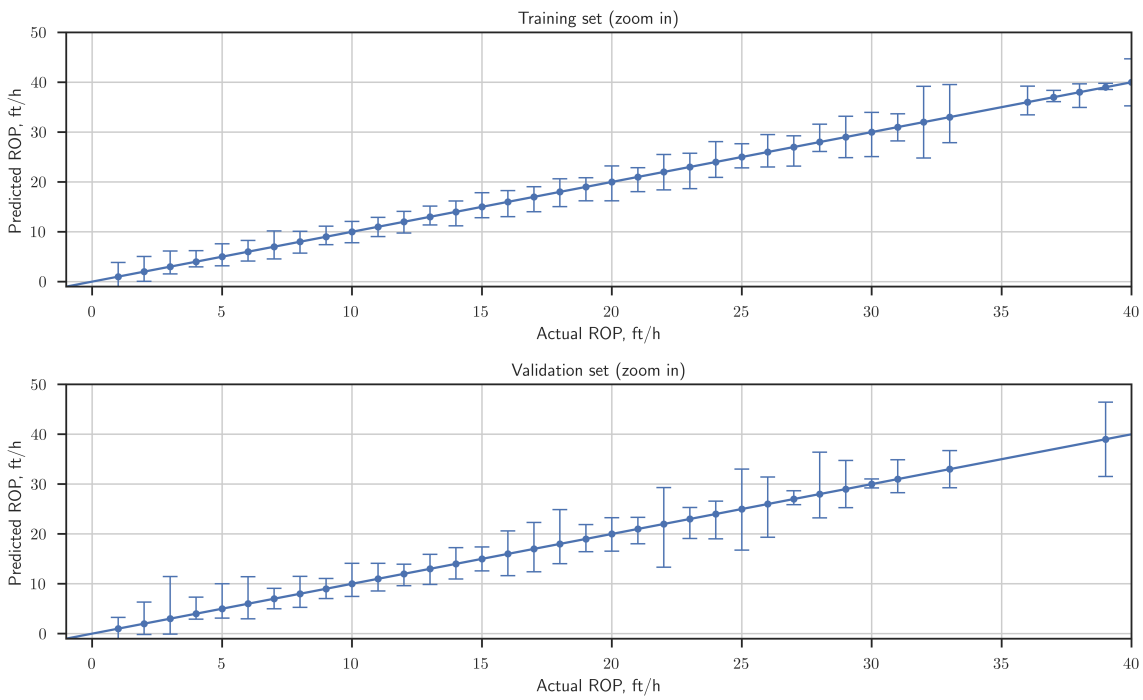

Figure 4.8: 95% confidence interval plot for ROP modeling, random train/validation splitting

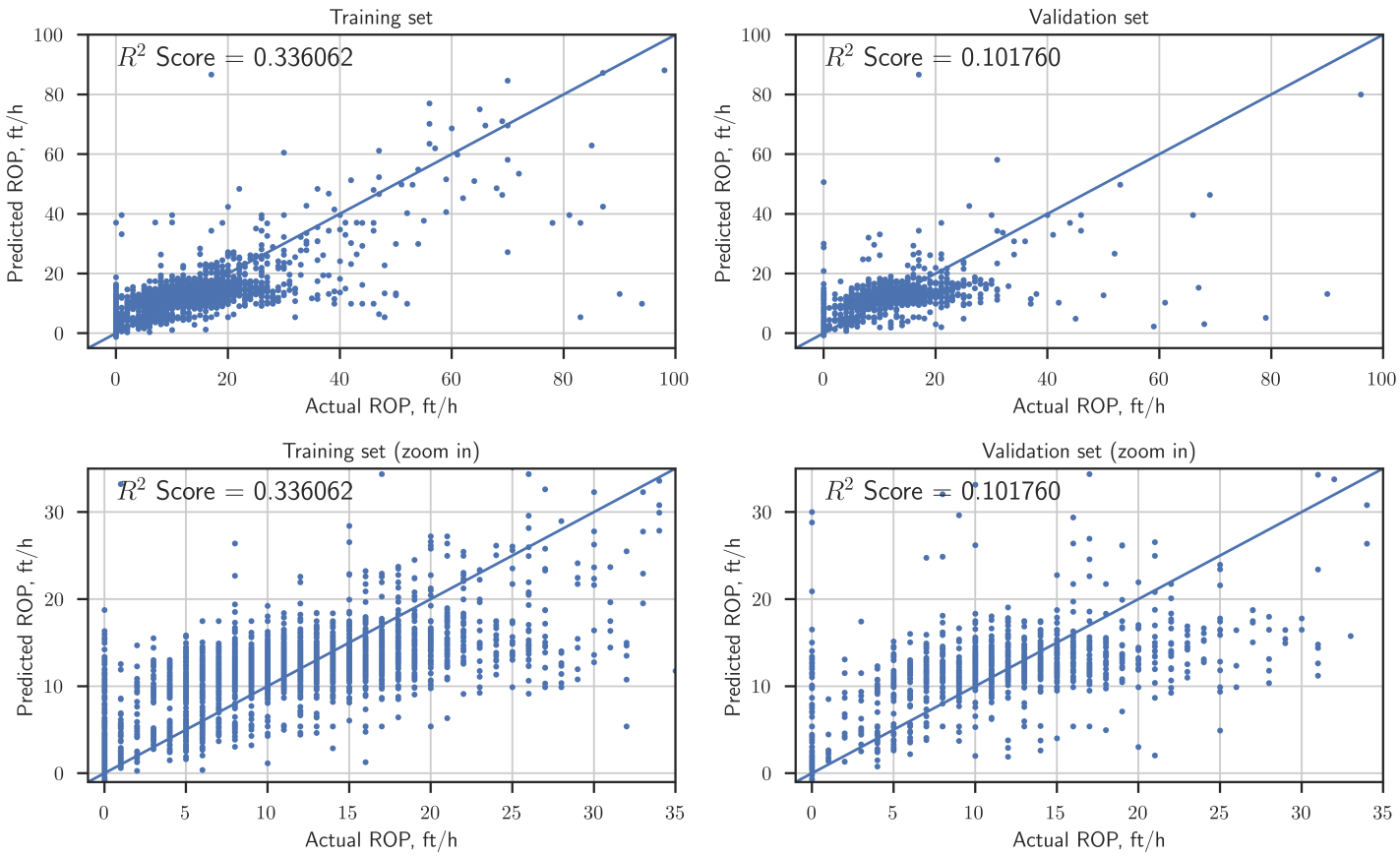

Figure 4.9: Correlation plot for ROP modeling, well-by-well train/validation splitting

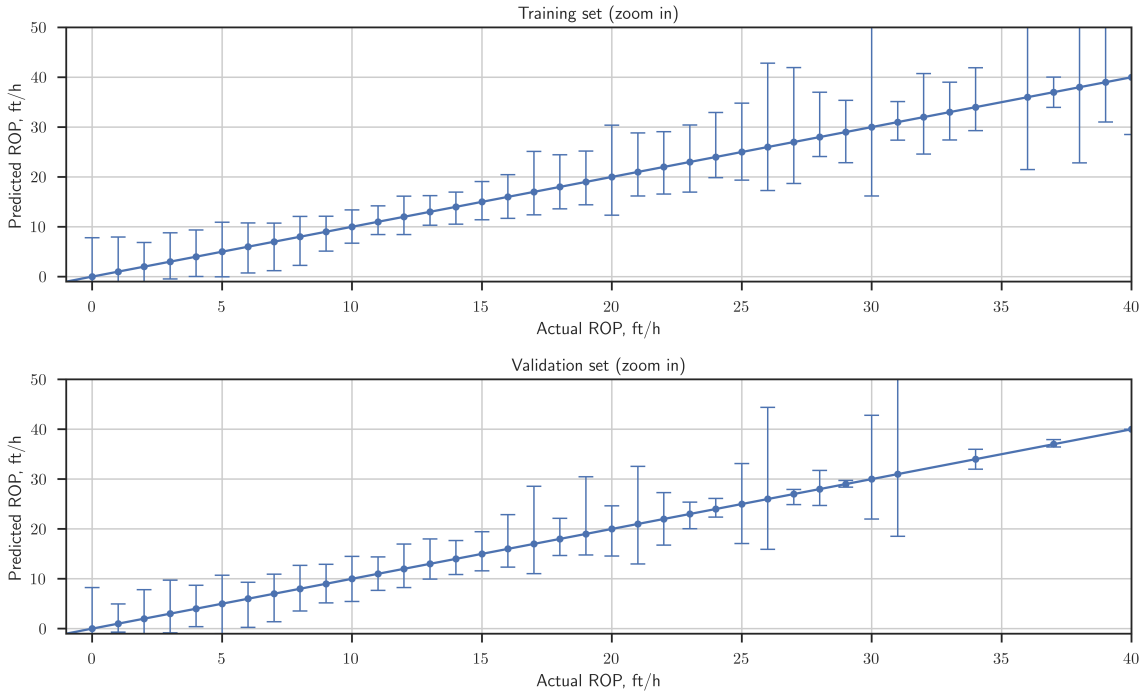

Figure 4.10: 95% confidence interval plot for ROP modeling, well-by-well train/validation splitting

4.4.2 Non-ROP modeling

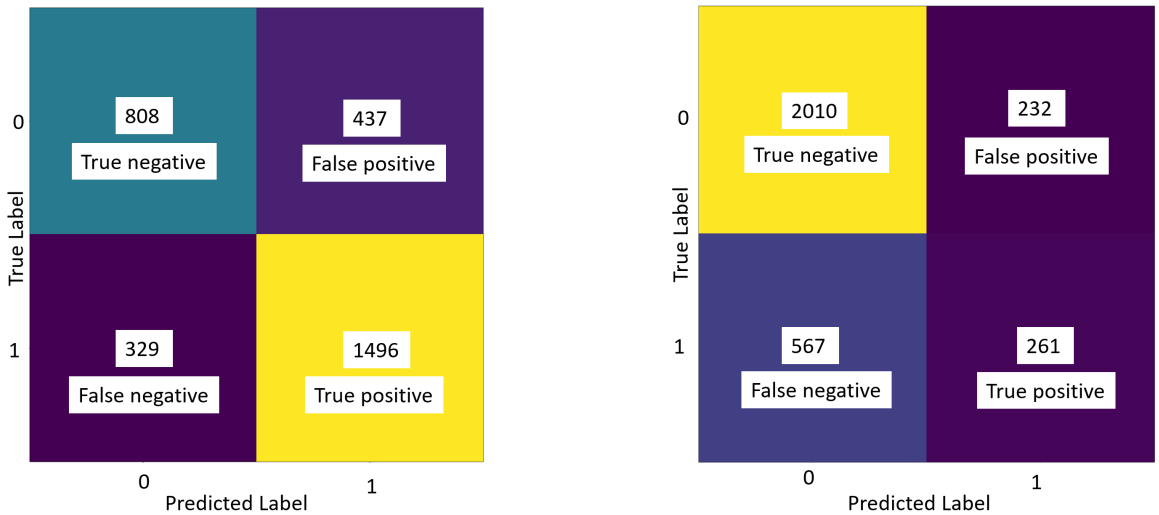

Figure 4.11: Confusion matrix for tripping and problem predictions: left for tripping predictions, right for problem predictions. One indicates tripping/problem did happen, zero indicates tripping/problem did not happen.

In contrast to the poor results with ROP modeling, modeling with non-ROP information did produce a model that is usable in production. In Figure 4.11, the left confusion matrix is from "TRIP" prediction model, while the right confusion matrix is from "PROBLM" prediction model. The left matrix indicates a really good result as the model predictions are correct most of the time, especially in when tripping happens, with ratio of True positive to False positive ratio nearly equal to 3.5:1.

On the other hand, with the right confusion matrix, while it shows good results when forecasting "PROBLM" in the cases where problem does not happen, the model does poorly when problem does happen. When problem does happen, the ratio of True positive to False positive ratio is 1:1. In other words, if the model forecasts problem in the future, it is just as reliable as flips of a coin. Unfortunately, although problem does not happen in most drilling processes, any instance of it can result in extensive nonproductive time and consequently significant costs. Therefore, it is important that the model be able to predict possible future problem correctly rather than the opposite.

The final result is a model that is useful in production. The trip prediction model, with its accurate prediction, is able to give drillers at least one day in advance warning, which can result in proper preparation and reduction of nonproductive time.

Chapter 5

Natural Language Processing

5.1 Introduction

In additional to impressive results in computer vision that made self-driving vehicles a reality, modern deep neural network also revolutionized the field of natural language processing (NLP). The most modern NLP models, with the help of deep neural networks, are capable of NLP tasks that exceed the highest expectation of a decade ago. With GPT-3[12] or RoBERTa[20], researchers have created NLP models that are capable of holding natural conversations with humans, or writing an essay from a single sentence prompt that is indistinguishable from human writings.

This chapter will discuss the usage of natural language processing (NLP) in the drilling modeling process. This includes constructing a BERT architecture used in this study, and the history, the theory and the evolution of the BERT architecture. In addition, this chapter also shows how to prepare the data and train the BERT model, and how to integrate the new results into the results obtained from Chapter 4.

5.2 Seq2seq model

The goals of natural language processing are to read, and to understand the human language. NLP with machine learning, by itself, is not a novel concept as it is dated from at least the 1950s. It is not easy for even humans to understand their own method of communication, and the problems only exacerbate when replacing the human with a machine. Due to being completely unstructured data, modeling natural language is a notoriously hard problem. It is very difficult to meaningfully construct numerical representations of a letter or a word (word embedding); natural language is full of ambiguities and can only be inferred within the context itself. Understanding a language requires

not only the model to understand the meaning of each word but also the abstract concepts that associate with that meaning. Consequently, it comes as no surprise that the histories of NLP are filled with ups and downs, and it was considered as a dead-end more often than not throughout its developments. Up until the 1990s, after hundreds of millions of dollars poured into research, NLP still used complex models with numerous "hand-written" exceptions that do not generalize well. In 1997, long short-term memory (LSTM)[21] recurrent neural network (RNN) architecture was introduced, and became the de-facto choice for NLP for the next two decades. However, similar to what happened in computer vision, NLP with deep neural network did not gain traction in the field of NLP until the mid 2010s.

One of the first breakthroughs of NLP that solved the problem of word embedding was with the Word2vec technique[22] (Figure 5.1). Previously, the most popular method of word embedding was one-hot-encoding, where each word is given a different one-hot vector (a binary vector with all zeros except for one place) representation. Unfortunately, one-hot-encoding word is extremely inefficient for this kind of task, where a vocabulary of a few thousand words is considered small. The ideal of Word2vec is rather than encoding words as one-hot vectors, encode them as dense vectors with fixed dimension (dense vector representation). The mappings of these dense representations are unknown, but they can be learnt. By training a neural network on any NLP task using these mappings as inputs, a more optimal mapping can be found at the end of the training. These new embeddings not only solve the dimensionality problems of one-hot-encoding, but also partially capture the meanings behind each word (words with similar meanings are mapped closer in the vector space). Since its introduction, Word2vec and its derivatives have become the first choice for many language modeling tasks. In 2014, Word2vec, coupled with LSTM or the newly introduced Gated Recurrent Unit (GRU)[23] recurrent neural network, opened the flood gate between deep neural networks and NLP.

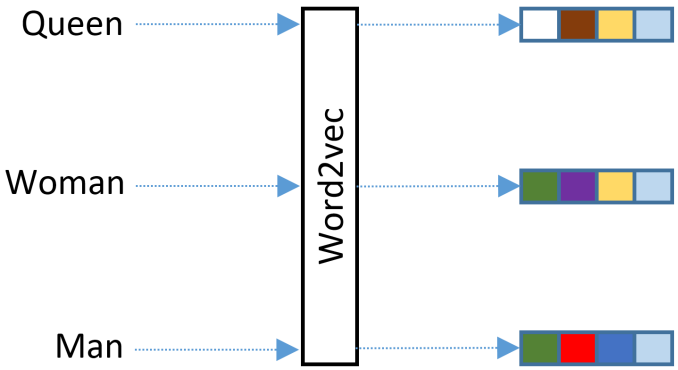

Figure 5.1: Simple Word2vec embedding

The most useful and obvious NLP task that was within reach is machine translation: translate

a sentence from one language to another language. A model that was very successful with this kind of NLP task was seq2seq[24]. The architecture is conceptually simple: the model consists of an encoder and a decoder, the encoder maps the embedded input into a internal representation while the decoder maps the internal representation of the inputs into the target language. The encoder and decoder are often based on the RNN architecture or its derivatives (LSTM, GRU).

Despite many variants, all RNN architecture bears the same structure as described in Figure 5.2:

1. The input is composed of multiple elements with a specific iterating order.
2. The network has multiple cells, with the number cells equal to the length of the input.
3. Each cell uses its inputs and optionally a hidden state vector to produce a new output and a new hidden state vector.
4. The network iterates on the input, with the nth cell uses the nth input and the $n^{\text{th}} - 1$ hidden state vector to produce the nth output and the nth hidden state vector.

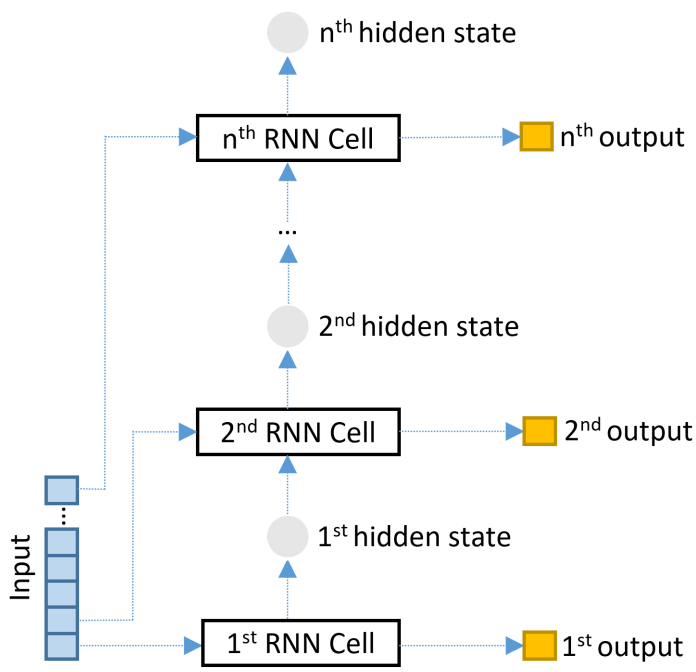

Figure 5.2: Common Recurrent Neural Network structure

Most of the differences between RNN architecture (plain RNN, LSTM, or GRU) are laid in the constructions of each cell, while the overall structure stays nearly unchanged. Consequently, the simplest architecture for seq2seq can be described as in Figure 5.3:

1. The input is the dense vector representation of each word in the original sentence, with special tokens at the beginning and the end: `<BOS>` indicates the beginning of a sentence, and `<EOS>` indicates the end of a sentence.

2. The encoder iterates on the input, producing the nth hidden state vector, while all other outputs are discarded.

3. The decoder starts by taking the `<BOS>` token and the nth hidden state vector as input, producing an embedded output in the target language. Then the decoder will take previous output, in conjunction and the last hidden state vector, to produce the next output. This process stops when the decoder produces the `<EOS>` token.

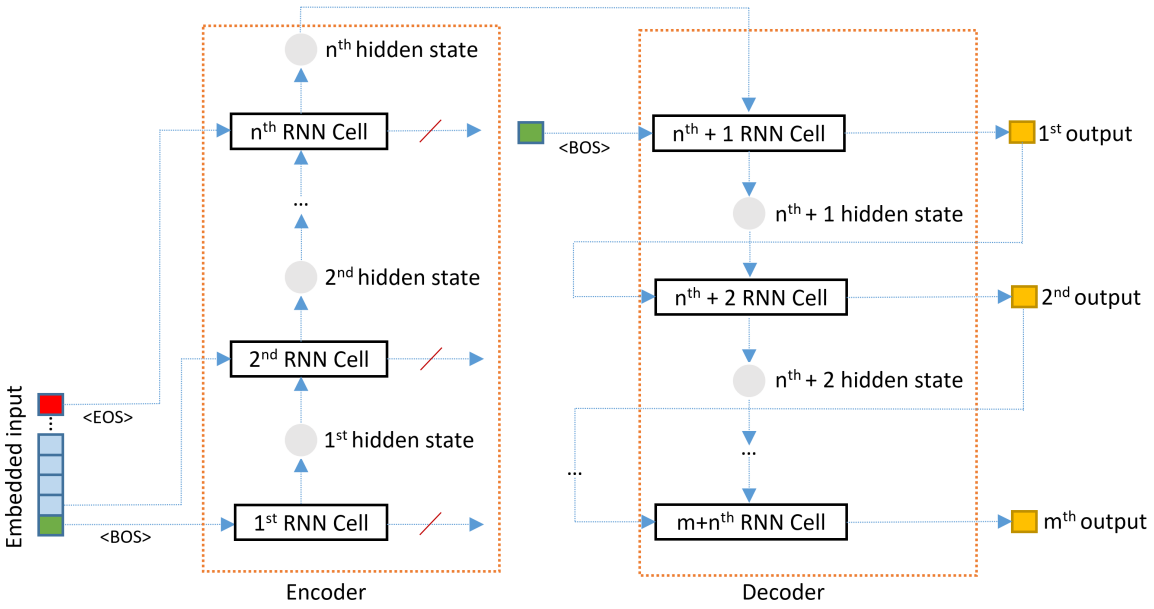

Figure 5.3: Simple seq2seq architecture

The simple seq2seq architecture, although it works well with short and simple sentences, has a significant drawback. The results of the decoder depend solely on the last hidden state vector of the encoder. In the case of long sequences, there is a very high probability that the final hidden state is strongly influenced by recent inputs, while the initial contexts may have been lost. This is called the "forgetting" in NLP. Changing the RNN cell structure to more modern ones like LSTM and GRU does improve the final results, but they still do not solve the "forgetting" at root. A new type of "hidden state" should be created for this specific problem.

The main problem of using only the last hidden state vector as input for the decoder is that the encoder cannot effectively compress all information from the input into a fixed-size vector. An

obvious solution is for the decoder to use all of the hidden state vectors, rather than only the last one. Although this solution looks simple at a glance, it is hard to achieve as a decoder, which produces hidden state vectors with the same dimension of all the encoder's hidden state vectors concatenated, will have its size grow exponentially to the size of the input. Such a decoder is not computational feasible on modern computers. Fortunately, with the introduction of the concept of **Attention** in 2014, an elegant and efficient solution to the 'forgetting" problem was found.

5.3 Attention, Self-attention, and Multihead attention

With the previous seq2seq problem, assuming that there are n encoder hidden state vectors $es_1, es_2, ..., es_n$, let us define the attention score to be s_k^m, then the attention weight a_k^m can be described as:

$$a_k^m = softmax(s_k^m) = \frac{\exp s_k^m}{\sum_i^n \exp s_i^m} \tag{5.1}$$

and the context vector for the mth decoder cell d_m is:

$$c^m = \sum_i^n a_i^m \cdot es_i \tag{5.2}$$

Here, a new concept called attention score is introduced: the attention score s_k^m, which is a function of the encoder hidden state es_k and the decoder hidden state ds_m, described the relevancy of the hidden state es_n to the mth decoder cell d_m. If there is a such function, then problem of "forgetting" is solved because at each mth step of the decoder, it can look at the mth context vector c^m which contains most relevance hidden states; or the mth decoder cell focuses its attentions on only the relevant parts of the input.

There are many ways to approach the problem of attention score function, one of the popular and effective ways is the bilinear method described in the Luong model[25]. In the bilinear model, for each decoder cell d_m, the attention score can be calculated as:

$$s_k^m = ds_m^T \times W \times es_k \tag{5.3}$$

where ds_m^T is the transposed of the mth decoder cell's hidden state, W is a learnable matrix, and es_k is the kth hidden state of the encoder.

With Equations 5.1, 5.2, and 5.3, a new seq2seq with attention model can be constructed as described in Figure 5.4. In this new seq2seq with attention model, the main difference compared to

the simple seq2seq is with the mth output of the decoder. Rather than using the output of the mth decoder cell directly, the output is calculated as:

$$y_m = \tanh(W_d \times [ds_m, c^m]) \tag{5.4}$$

where ds_m is the mth decoder cell's hidden state, W_d is a learnable matrix c^m is the context vector for the mth decoder cell, and $[ds_m, c^m]$ is the concatenation of two vectors ds_m and c^m.

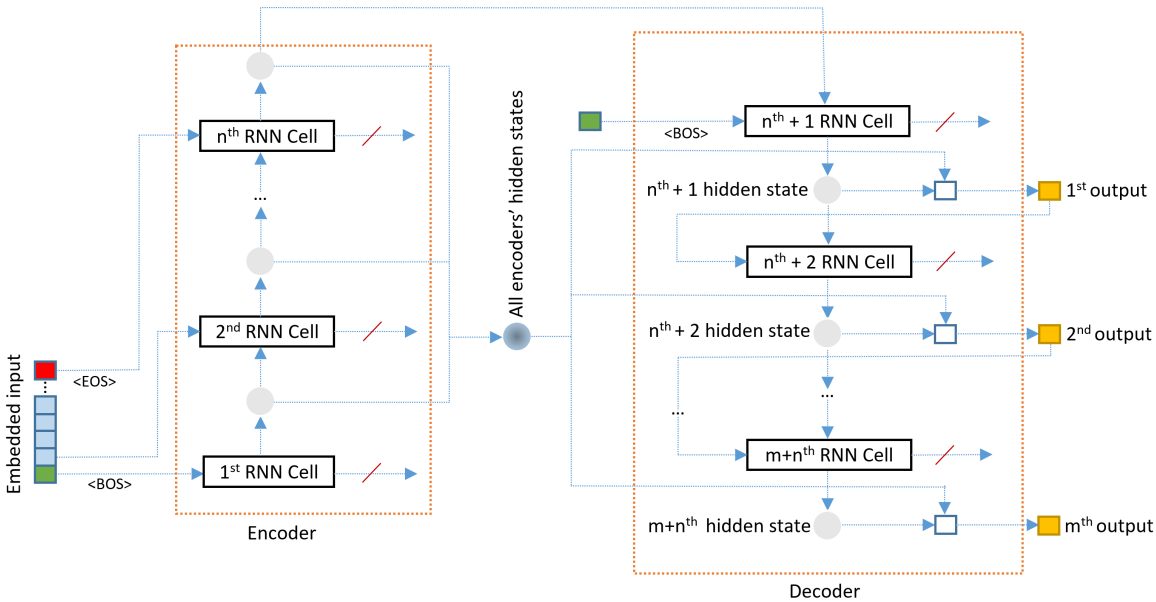

Figure 5.4: seq2seq with attention architecture

With the new concept of attention, the deep neural network is now capable of doing NLP tasks on very long input without the risk of "forgetting". This revolutionized the field of NLP, enabled researchers to do tasks that no other machine learning model could do before. Despite the tremendous success of seq2seq with attention, there is still room to improve the model further. The seq2seq uses RNN, which is very computationally intensive on longer inputs, hindering the training process.

Another significant breakthrough came in 2017 with the Transformer architecture[26]. This new architecture replaces the RNN in the encoder and the decoder of the seq2seq model with attention blocks. On top of the improved quality, this replacement also brings an order of magnitude of improvements in training speed. As of 2021, the Transformer model is the preferred architecture of nearly any NLP task.

A new and very important concept in the Transformer is the concept of self-attention. In normal seq2seq architecture, the attention mechanism is looking from one decoder state to all encoder states.

On the contrary, the new ideal with self-attention is that the attention mechanism is now from each state to all other states (within a same set). Consequently, with the normal attention mechanism, the encoder needs $O(N)$ steps to process an input of length N, where in the self-attention, the encoder only uses $O(1)$ (constant) steps to process any sentence. This not only brings significant speed improvements but also higher quality results compared to NLP model with RNN.

Formally, the concept of self-attention can be described with a concept of query-key-value attention mechanism, where query, key, and value are three vectors which can be defined as:

- Query: $Q = W_q \times$ [embedded input], vector the attention mechanism is looking **from**
- Key: $K = W_k \times$ [embedded input], vector the attention mechanism is looking **to**
- Value: $V = W_v \times$ [embedded input], the value of the attention

where W_q, W_k, and W_v are three learnable matrices with same dimension. There are a query, a key, and a value matrix corresponding to every embedded element in the input. Then the self-attention can be defined as:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d}}) \cdot V \tag{5.5}$$

where K^T is the transpose of the key matrix and d is the dimension of K. Consider an input x which has n embedded elements, then the output of a single self-attention head (Figure 5.5) on that input is:

$$\begin{aligned} &\text{Let:} \\ &self\_attention(x_k) = concat[Attention(Q_k, K_i, V_i)], \qquad \text{for } 0 \leq i \leq n, i \neq k \\ &\text{Then, the output of self-attention head is:} \\ &self\_attention\_head(x) = concat[self\_attention(x_i)], \quad \text{for } 0 \leq i \leq n \end{aligned} \tag{5.6}$$

The self-attention head described in Figure 5.5 and Equation 5.6 is only for the encoder. There is a difference between the head used in the encoder and the head of the decoder. The decoder does not know the complete full sequence before hand, hence it can only look at the previous outputs. Therefore, any future token of the output will be masked out (Figure 5.6).

One major benefit of the self-attention head model is it can easily expand, both in the horizontal direction and the vertical direction. If the model is expanded horizontally, then it is called a multihead self-attention model where each head produces a different self-attention. If the model is expanded vertically, then it is called a multilayer self-attention model where the previous head's self-attention is the input for next self-attention head. The Transformer encoder-decoder model (similar to seq2seq) takes full advantage of the self-attention mechanism by expanding the self-attention head vertically and horizontally.

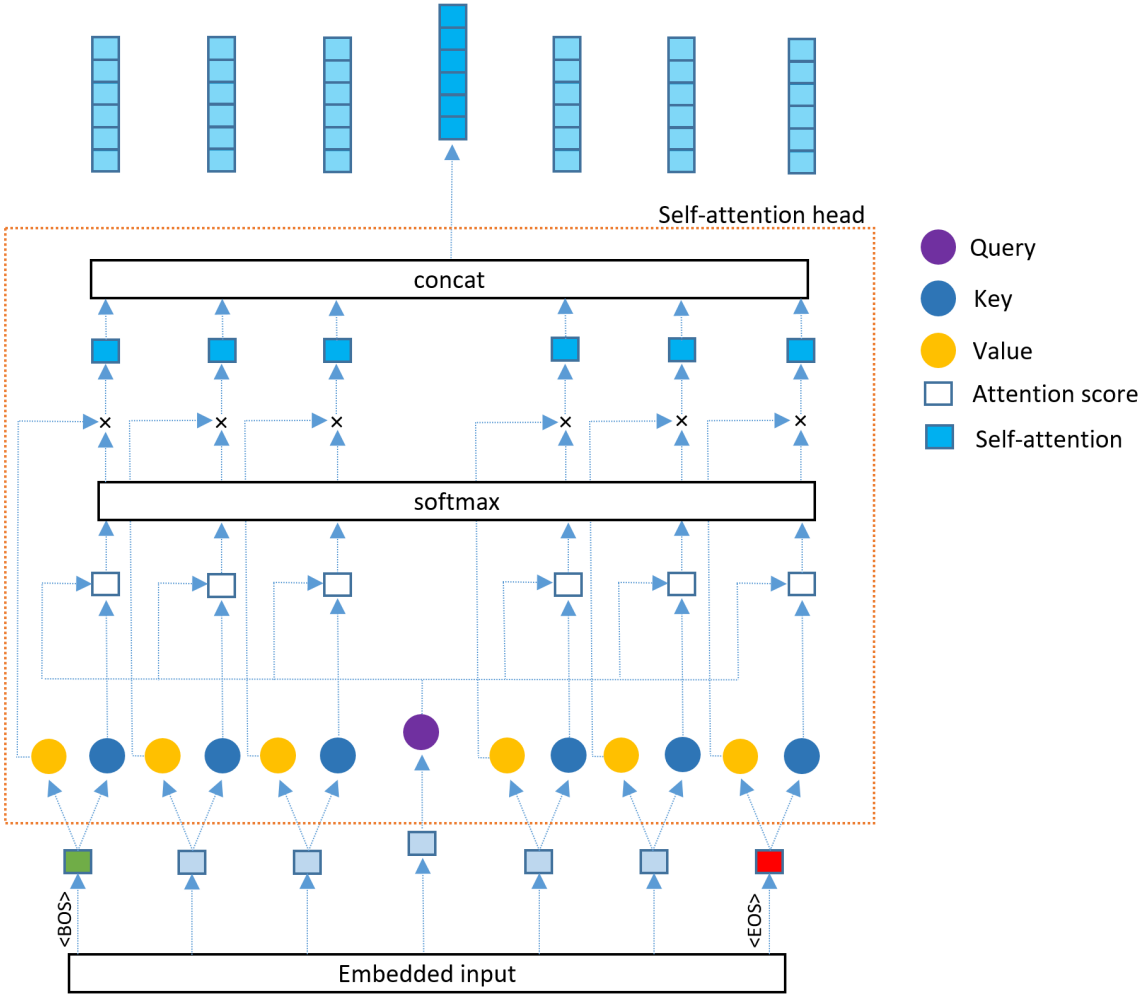

Figure 5.5: Self-attention head for encoder

The Transformer architecture is similar to the encoder-decoder structure of seq2seq, but uses multiple layers of multihead attention units for both its encoder and decoder. Consider an input x which has n embedded elements, let a self-attention head defined as Equation 5.6, then the output of a multihead attention unit composed of h heads can be defined as:

$$multi\_head\_attention(x) = concat[self\_attention\_head_j(x)], \quad \text{for } 0 \leq j \leq h \tag{5.7}$$

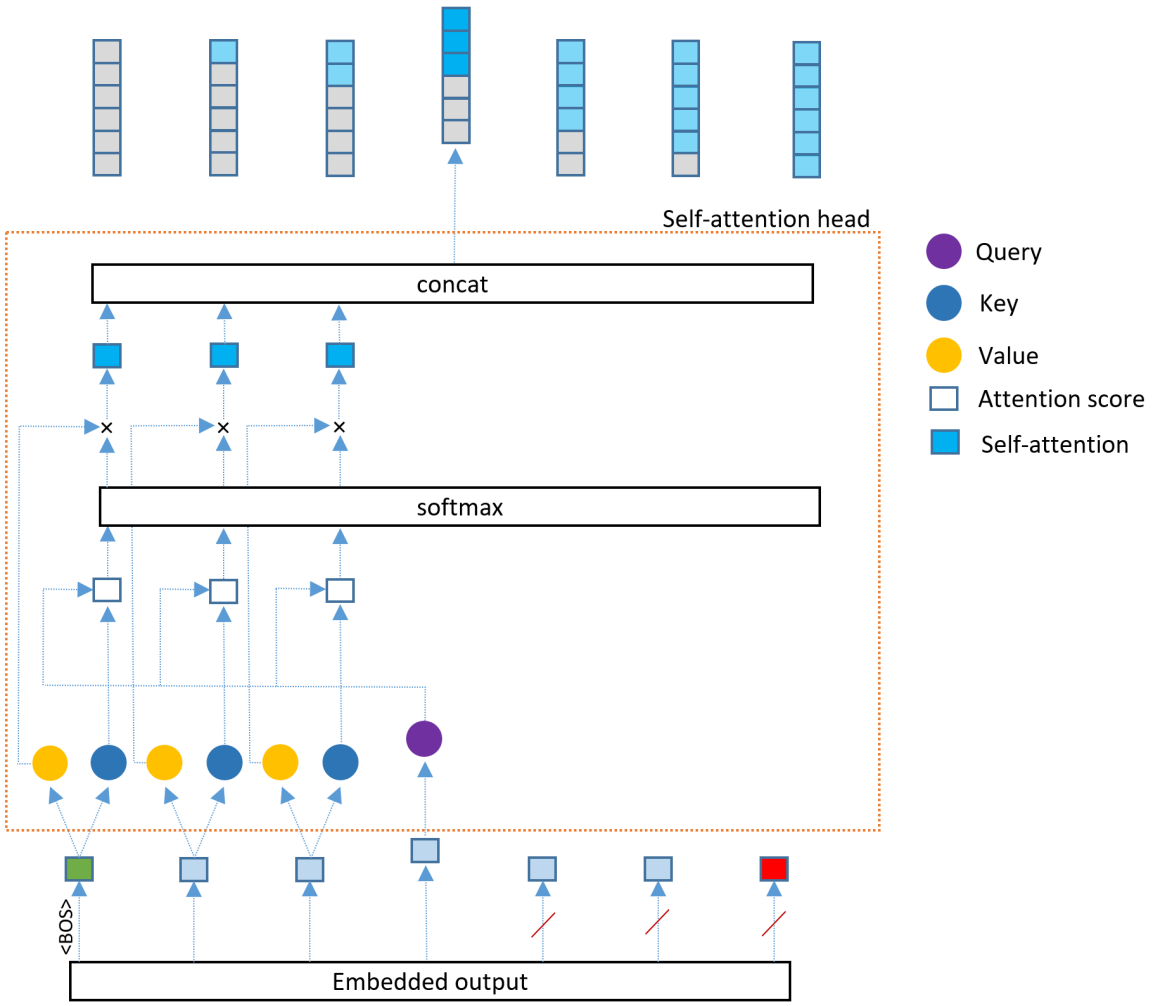

Figure 5.6: Masked self-attention head for decoder

where $self\_attention\_head_j$ is the jth self-attention head. Similarly, the output of a masked multihead attention unit composed of h heads is defined as:

$$masked\_multi\_head\_attention(x) = concat[masked\_self\_attention\_head_j(x)], \quad \text{for } 0 \leq j \leq h \tag{5.8}$$

where $masked\_self\_attention\_head_j$ is the jth masked self-attention head.

With the multihead attention unit properly defined, the Transformer model can be constructed as in Figure 5.7. In the Transformer model, the encoder/decoder is composed of encoder/decoder cells. Each encoder cell include a multihead attention unit, and a feed-forward unit. The feed-forward unit can be any feed-forward neural network without an activation function at the last layer. After each

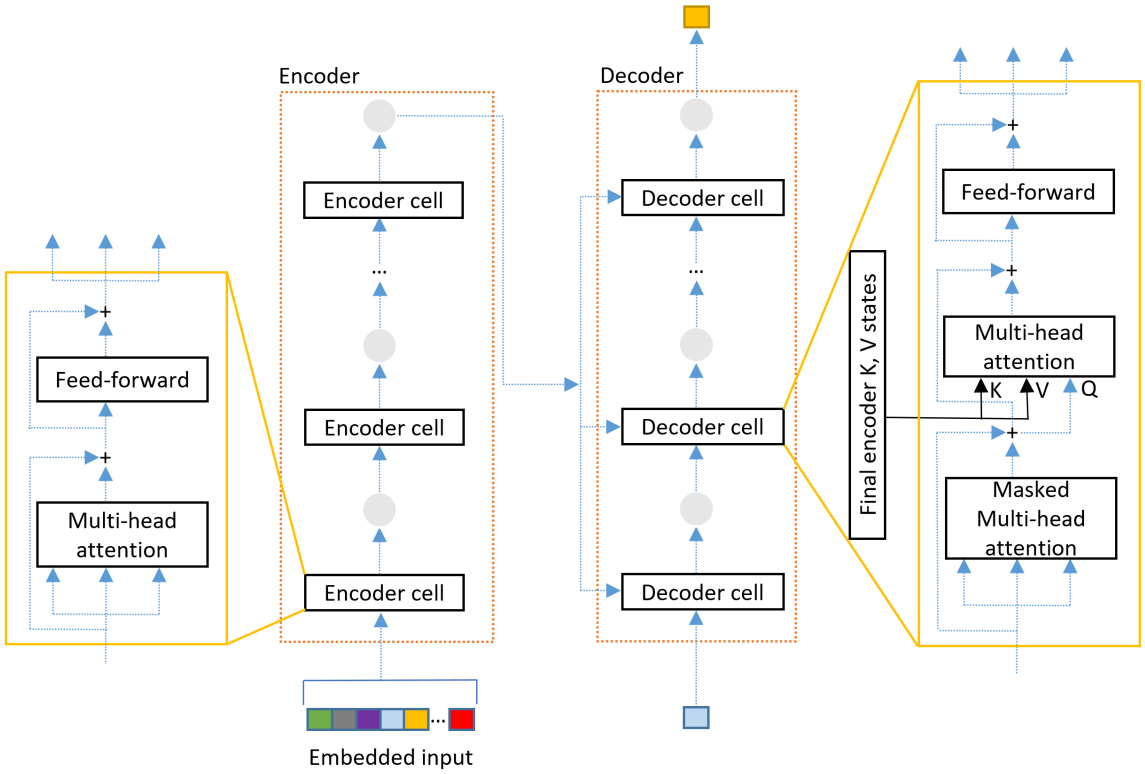

Figure 5.7: Transformer architecture

unit is a residual connection (described in Equation 4.6). The decoder unit is similar to the encoder unit, except that it has a masked multi-head attention unit in front. In addition, rather than using the Key and Value vectors from the previous masked multihead attention, the multihead attention uses the Key and Value vectors from the final cell of the encoder. The training process is similar to the seq2seq training process:

1. The embedded input is fed into the encoder, producing the Key and Value vectors K, V at the final encoder cell.

2. The decoder starts by taking the `<BOS>` token and the encoder's K, V vectors, producing an embedded output in the target language. Then the decoder will take previous output, in conjunction with the encoder's K, V vectors, to produce the next output. This process stops when the decoder produces the `<EOS>` token.

Up until now, the method of choice for input embedding has been Word2vec. Although this embedding method works reasonably well with any NLP model, there are still some problems that need to be resolved. One of the biggest drawbacks of Word2vec is that it cannot effectively deal with out-of-vocabulary words (words that do not appear in the training set). This out-of-vocabulary

problem can be alleviated to some degree by training on a bigger dataset and masking out-of-vocabulary words with a special token (`<UNK>`) so that the model can guess it by using the contexts. However, when the out-of-vocabulary words are crucial to the meanings of the sentence (subject, verb, etc.), then the models collapse. In addition, the relationships between words have to be learnt every time; if a Word2vec embedding learnt about relationships between "TALL" and "TALLER", it will not immediately result in the learning of the relationship between "SHORT" and "SHORTER".

There is another method of sentence segmentation that remedies most of Word2vec drawbacks, it is called Byte Pair Encoding (BPE)[27]. Although Byte Pair Encoding is known from 1994, it only became the norm in input embedding around 2017, together with the rise of Transformer. Byte Pair Encoding originally is a simple data compression technique, that seeks to replace the most used pair of bytes with a single byte that does not appear in original data. The BPE method that is used in NLP task referred to an adaptation of BPE to use in sentence segmentation.

One the main motivations for using BPE is that in most languages, a word is not smallest order of division that has meaning. Word can be divided into prefix, root, and suffix, where each division has meaning by itself. Therefore, a subword segmentation will be more beneficial compared to word-by-word splitting. Byte Pair Encoding is an effective solution for the subword segmentation problems as BPE seeks to construct the original input with small tokens that have the highest frequency of appearing in the input. A common process to train a BPE model is:

1. Add the special token "<\w>" to the end of every word of the input dataset to indicate the end of a word.
2. Splitting all words into characters tokens (i.e "NEWEST" into ["N", "E", "W", "E", "S", "T", "<\w>"]).
3. Count the frequency of all consecutive character pairs, merge the most frequent consecutive character pair into a new token (i.e ["E", "S"] into "ES").
4. Repeat Step 2 and Step 3 until the targeted number of merges, or the targeted vocabulary size, has been achieved.

To encode a sentence into its Byte Pair representations, iterate through the list of all tokens, from longest to the shortest, and replace any substring in the input with the token if they are matched. The decoding process is simply the merging of tokens to form words. Most of time, the tokens produced from BPE will be turned into dense vector representation forms and served as the inputs for downstream NLP models.

A consequence of BPE is that the resulted tokens are likely to be subword tokenizations and carry some meanings on their own; common tokens from BPE include "S<\w>", "ER<\w>", "EST<\w>", "NON", "UN", etc. It is easy to see that BPE not only is able to work with out-of-vocabulary words, but also can guess and learn the relationships between words very effectively. For example, let assume a BPE model that only trained on general English and has not encountered any geological term so far, encounters an out-of-vocabulary word "CLAYSTONE". The BPE model has the tokens "CLAY" and "STONE" as they are very common in any dataset. Therefore, the BPE model splits "CLAYSTONE" into "CLAY" and "STONE<\w>" and helps the downstream NLP model to correctly guess the meaning of the word "CLAYSTONE".

5.4 Bidirectional Encoder Representations from Transformers

With the introduction of Transformer, and to some lesser extent Byte Pair Encoding, a new era of NLP was opened. Now NLP models can process very long inputs, can be effectively trained on terabyte-size datasets, and can achieve near human performance in some NLP tasks. However, these state-of-the-art NLP models all share some common points: they are all enormous with billions of trainable parameters, and they are all trained on extremely large datasets, which requires an enormous amount of computational power. Therefore, training a model for a specific NLP task until it achieves state-of-the-art performance is something only the biggest organizations are capable of.

Due to the difficulty in training a good NLP model, transfer learning is an active field of research in NLP. The goal of transfer learning is to transfer knowledge and information between different models in order to accelerate the training process. Transfer learning is especially effective when dealing with NLP tasks. Although NLP tasks may be different drastically from each other, most of the time they all have same common requirement to the NLP models built to solve them, which is the ability to read and understand written natural language.

Some simple methods for transfer learning in NLP are already described previously. Word2vec with Byte Pair Encoding and its derivatives are a form of transfer learning. If the inputs between two NLP models are similar, then the input embedding mapping of one model can be used for the other model, which eliminates the need of training a new input embedding mapping from scratch and may improve the final results of the other model. However, only the relationship mappings between words/tokens are transferred in this case, any other knowledge (grammar, syntax, etc.) the other NLP model has to figure out from scratch. A better and more complete transfer learning is needed.

This is where the holders of the state-of-the-art results (GPT-3, RoBERTa, etc.) come in. These Transformer based models have achieved exceptional results on a wide range of NLP tasks, and have taken over the NLP field completely. Together with Byte Pair Encoding, Transformer based models have become the first choice for almost any NLP task. Despite many difference in architectures and implementations between these models, they all share the same design concept (aside based on Transformer), namely a model trained on general NLP tasks that fine-tuned for all tasks. The model is trained on a set of "general NLP tasks" with an extremely large dataset (billions of words), then the knowledge can be transferred to any NLP task by fine-tuning the model to that specific task. This resulted in a very versatile NLP model that can quickly be modified to achieve exceptional results on most tasks.

Bidirectional Encoder Representations from Transformers[28] (BERT) is a Transformer based NLP model with very good results on most NLP tasks and exceptional transfer learning capability. The architecture of BERT is remarkably simple: it is the encoder part of the Transformer architecture (Figure 5.7). What gives BERT its strength is the training objective and how BERT is used in further downstream tasks.

BERT is an encoder from the Transformer architecture with an additional linear and softmax layer at the end, trained on two "general NLP tasks": Masked Language Modeling (MLM), and Next Sentence Prediction (NSP). In Masked Language Modeling, BERT is asked to recover the original sentence from a corrupted input. In more detail, with the MLM, the training process can be described as (Figure 5.8):

1. Take any sentence as input. Add the special tokens `<CLS>` at the beginning and `<SEP>` at the end of the sentence.
2. Select some tokens from the input with $p = 15\%$ as the probability of being selected.
3. Replace the selected token with: a special `<MASK>` with $p = 80\%$, a random token with $p = 10\%$, the same token (remain unchanged) with $p = 10\%$.
4. Compute loss between the original input and the model predictions.

Another "general NLP task" that BERT is trained for is Next Sentence Prediction (NSP). NSP is a binary classification task, where the input is composed of two corrupted sentences, and the model is asked to predict whether the two sentences are consecutive sentence or not. For example:

- Input: `<CLS>` I `<MASK>` the report on my computer. `<SEP>` The report is fifty `<MASK>` long. `<SEP>`
  Output: `TRUE`

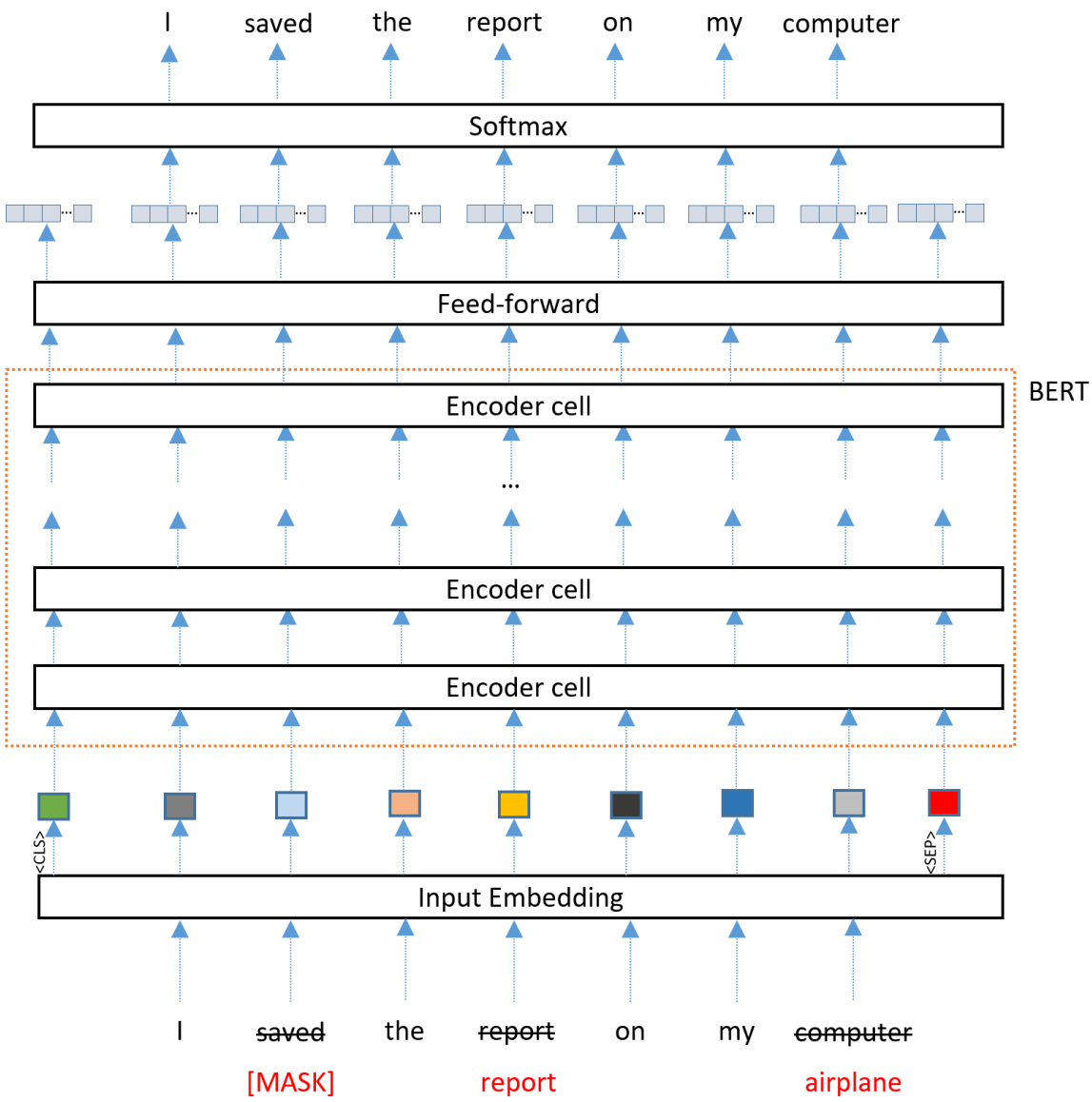

Figure 5.8: Masked Language Modeling with BERT

- Input: `<CLS>` I `<MASK>` the report on my computer. `<SEP>` The beach is a perfect `<MASK>` destination. `<SEP>`
  Output: `FALSE`

Usually, the BERT model will be trained to achieve the best results on both Masked Language Modeling task and Next Sentence Prediction task.

To fine tune the model to the downstream task, the input is fed into the pretrained BERT model

and the output corresponds to the <CLS> token will be used as input for a feed-forward neural network. This feed-forward neural network will have its outputs suitable for the downstream tasks, and will be trained together with the model used in the downstream tasks (Figure 5.9). In most cases, the pretrained BERT model's parameters are frozen and will not be modified in the fine tuning process.

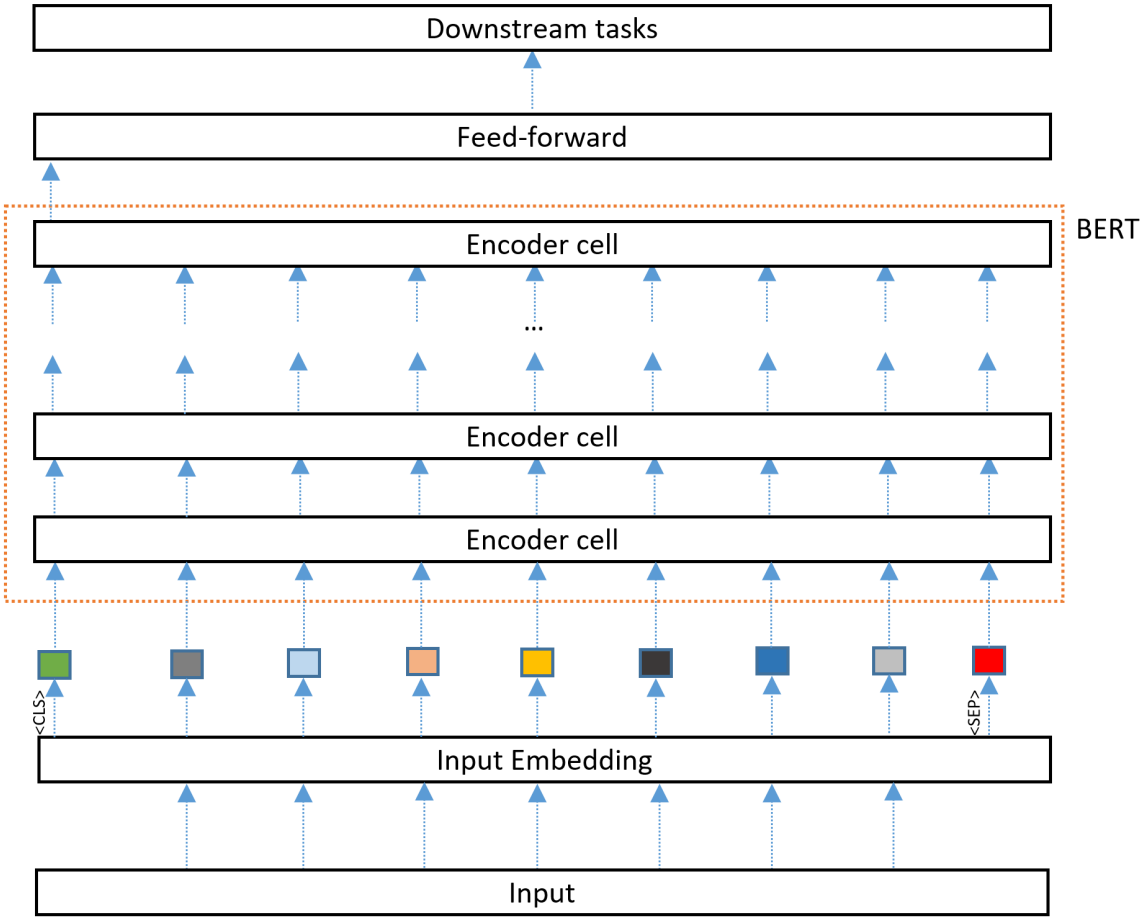

Figure 5.9: Fine-tuning with BERT

5.5 Methodology

5.5.1 Qualification of BERT

In this project, a pretrained BERT model called `bert-base-uncased` was used to process the textual drilling records. This pretrained model is a case-insensitive, 12 layers, 12 attention heads, 110 million parameters, BERT architecture neural network trained on the BookCorpus, a corpus of 11038 unpublished books, and the entire English Wikipedia. The model also comes with a WordPiece tokenizer (a slightly modified BPE tokenizer) that splits the string inputs into small tokens. This makes the model ready to receive textual inputs under the form of strings. The `bert-base-uncased` model can receive an up to 512 tokens-long input from the WordPiece tokenizer. If the number of tokens in the input is N, then the output will have the shape of $[N \times 768]$, where the first and the last outputs correspond to the `<CLS>` and `<SEP>` special tokens of the input string.

To process the textual records, each record is pass through the WordPiece tokenizer and the `bert-base-uncased` model, then the output corresponds to the `<CLS>` is taken (the first output), which has the size of [768]. This vector of size [768] will be served the numerical representation of the textual information in the original record.

The first task was to qualify the pretrained BERT model, so that is it is able to parse and understand these remarks and lithological comments from the drilling records. As described at the end of Chapter 2, there are four different categories for the remarks. Together with the lithological type, there are five categories that any piece of textual data can fall into. If the network can truly understand the contents of the remarks/comments, then it should be easy to categorize them into the correct type.

In order to test its performances on the drilling records, the remarks/comments were fed into the pretrained BERT model, then the output corresponds to the `<CLS>` token was fed to another feed-forward neural network to categorize the source of the input. All the remarks/comments in the drilling records were used as the dataset for this task; 70% of the randomly selected remarks/comments were used for the training, and the rests were used for validation.

5.5.2 Modeling with Natural Language Processing

As discussed at the end of Chapter 3, in addition to standard numerical records, there is also a wealth of textual data in the daily drilling record. This textual information often provides valuable information that is not available elsewhere. However, this textual information is recorded as written English remarks without any standardized structure. It is very hard to automatically process these

textual records, hence Chapter 4 inputs do not contain them. However, BERT provided an elegant solution to process these textual records.

The numerical representations of the remarks/comments described previously, together with the numerical drilling records , were used as inputs for this task. The new representations of the remarks/comments is concatenated into the final input described in Chapter 3. Except with the new inputs, the methodology and the goals of the studies described in this chapter are identical to the methodology and the goals of Chapter 3: DPDBN is the still the deep neural network architecture used; the target is still to create a ML model that is able to accurately predict the ROP/"TRIP"/"PROBLM" from the drilling records.

5.6 Results

5.6.1 Qualification of BERT

As the dataset is imbalanced (the numbers of datapoints belonging to each class are not equal), accuracy cannot be used to quantify the results of the described model as the class with the most datapoints will dominate the accuracy results. A better performance metric, which accounts for the imbalances between classes, like F-1 score, should be used in this case. The F-1 score metric can be described as:

$$\begin{aligned}
&\text{Let:}\\
&\text{precision} = \frac{\text{True Postive}}{\text{True Postive+False Postive}}\\
&\text{recall} = \frac{\text{True Postive}}{\text{True Postive+False Negative}}\\
&\text{Then:}\\
&\text{F-1} = 2 \cdot \frac{\text{precision}\cdot\text{recall}}{\text{precision+recall}}
\end{aligned} \tag{5.9}$$

which means the F-1 score must lie in the range of $[0, 1]$, where 1 is the best possible and 0 is the worst possible.

The `bert-base-uncased` was able to achieve an F-1 score of **0.9** on both training and validation dataset, indicating the capabilities to read and understand textual information included in the drilling records. The results show that the BERT model was also able to work with long input (up to 512 tokens) without the problem of "forgetting". It can be concluded from the results that BERT is strong enough to use in latter modeling parts.

5.6.2 Modeling with Natural Language Processing

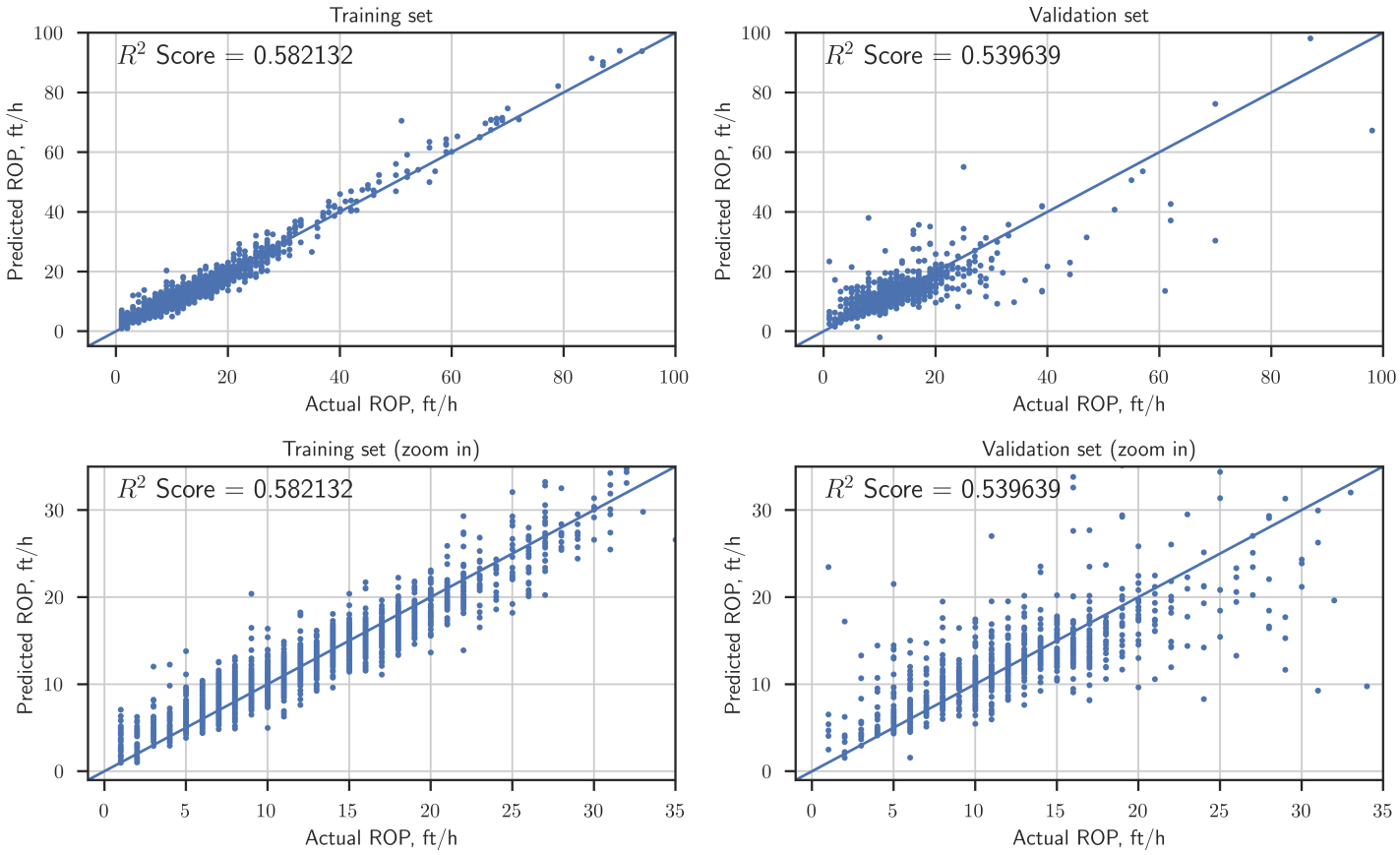

Figure 5.10: Correlation plot for ROP modeling with textual information, random train/validation splitting

Direct comparisons between Figure 4.7 and Figure 5.10, Figure 4.8 and Figure 5.11, Figure 4.9 and Figure 5.12, and Figure 4.10 and Figure 5.13 show that adding textual information only provides small improvements in the quality of ROP predictions. This contradicts the hypothesis from the end of Chapter 3 that adding lithological information would improve the quality of ROP predictions. Even with drillers' remarks and lithological information proxies from the mud-shakers, it is not possible to improve the ROP results enough to make the models usable in production.

An explanation for this contradiction is that the lithological proxies used are not sufficient, and direct measurements of the geological properties are necessary for accurate predictions. Another possible explanation is that the properties of the training dataset and the validation dataset are significantly different from each other. The final possible explanation is that BERT outputs are not

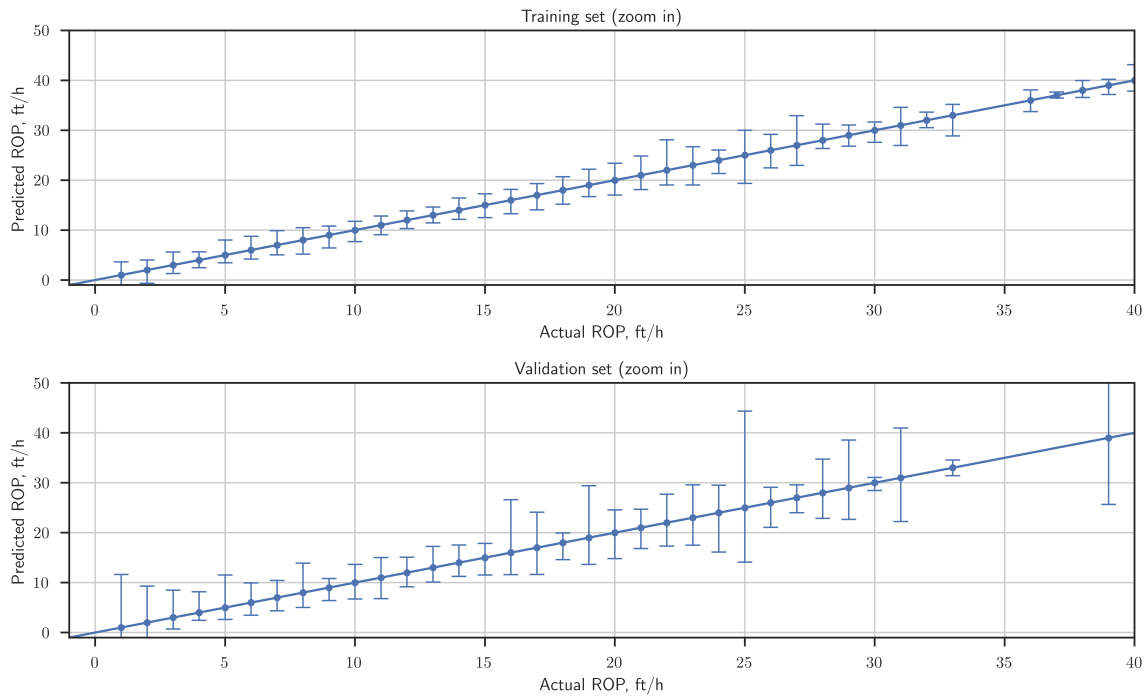

Figure 5.11: 95% confidence interval plot for ROP modeling with textual information, random train/validation splitting

useful to the downstream DPDBN, therefore there are only minor differences between using BERT and not using BERT. This explanation is rather unlikely, as demonstrated before, BERT is capable of understanding the remarks/comments in the drilling record, and a simple feed-forward neural network was enough to differentiate between remarks/comments. On the other hand, DPDBN, which is many times more powerful than standard feed-forward neural network, should be able to "understand" the outputs from BERT.

However, Figure 5.14 and Figure 5.15 provide a completely different picture when using BERT. Figure 5.14 shows that using BERT also provides small improvements in tripping prediction accuracy. However, Figure 5.15 shows that BERT helps bring significant improvements in problem prediction accuracy. Without using BERT, the accuracy of predictions if problems do indeed happen is only 50%, which is no better than a random guess. However, with BERT, the ratio of True positive to False positive improved to nearly 2:1. This results in a model that is accurate enough to use in production, which can give drillers ample preparation for possible future problems.

The results in Figure 5.15 can be explained by the fact that the drillers tend to make remarks/comments about unusual observations encountered when drilling. A future problem will likely

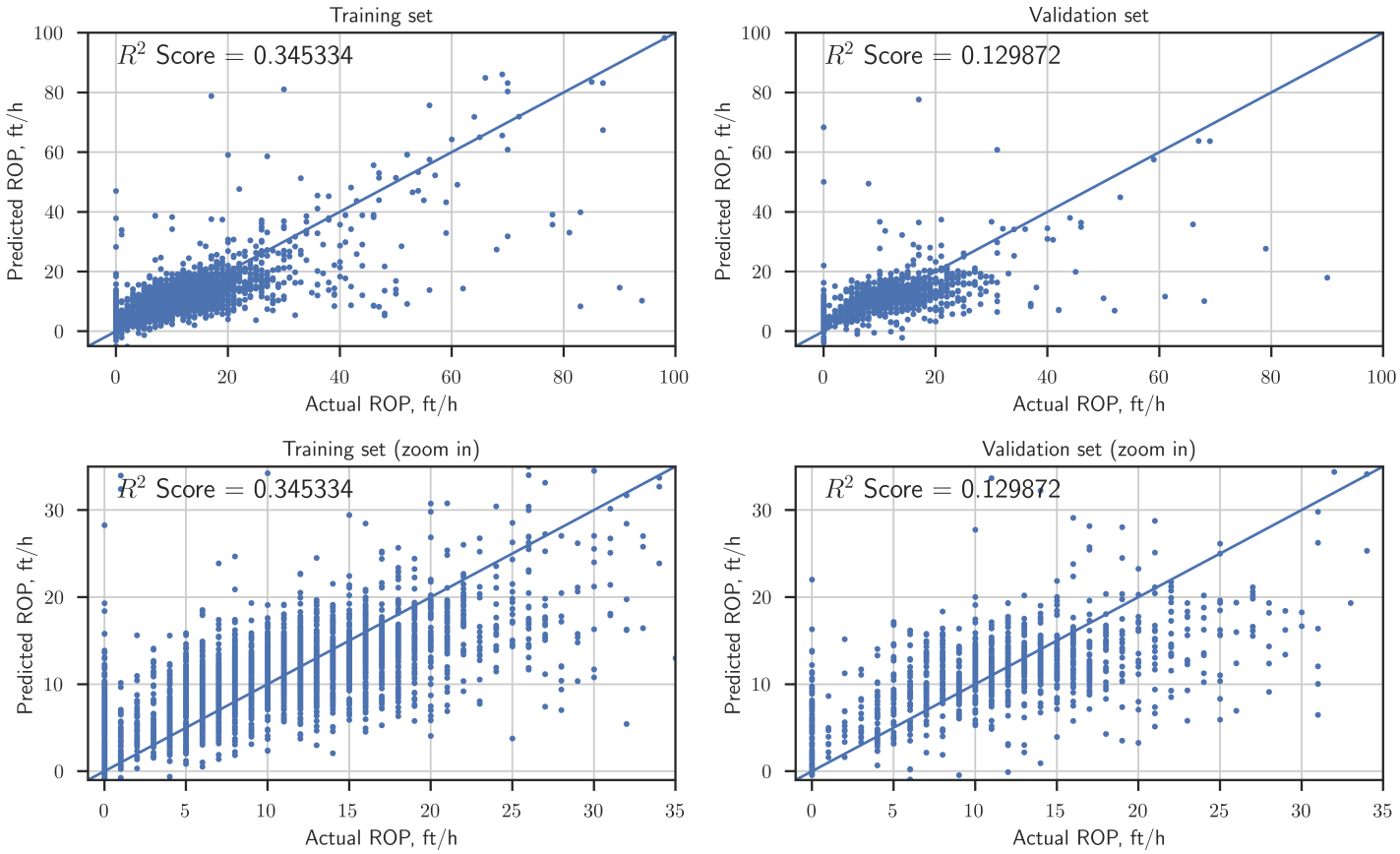

Figure 5.12: Correlation plot for ROP modeling with textual information, well-by-well train/validation splitting

correlate to unusual observations in the past (however the opposite is not true). BERT is able to pick out those observations, and in conjunction with numerical drilling records, DPDBN can give accurate forecast about possible future problems.

However, this only mystifies the problem of why BERT does not help in ROP modeling. BERT can give valuable information about the drilling process or geological information that is not available in standard numerical drilling records. This, in theory, should bring significant improvement in prediction quality. Unfortunately, that was not the case in this study.

Another observation is that by adding outputs from BERT, the prediction quality will improve in most cases. This shows that DPDBN is capable of processing very large inputs, and only pick out the most relevant features while ignoring the rest, which is a vital property when processing data like the inputs with BERT which are composed of more than one thousand features.

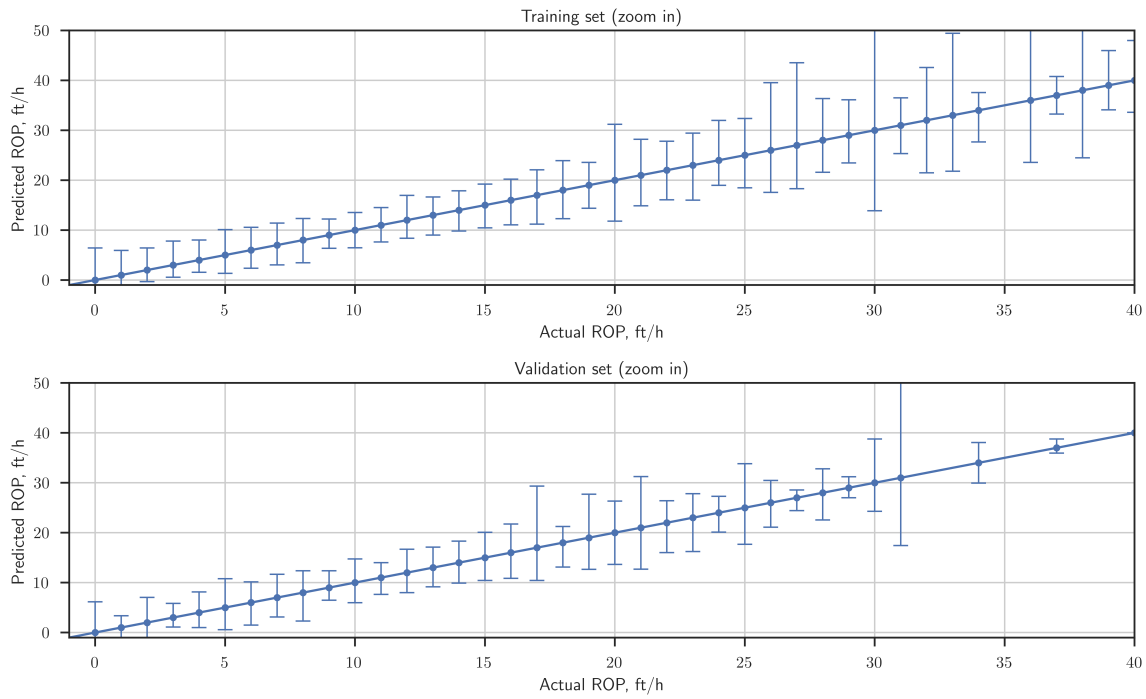

Figure 5.13: 95% confidence interval plot for ROP modeling with textual information, well-by-well train/validation splitting

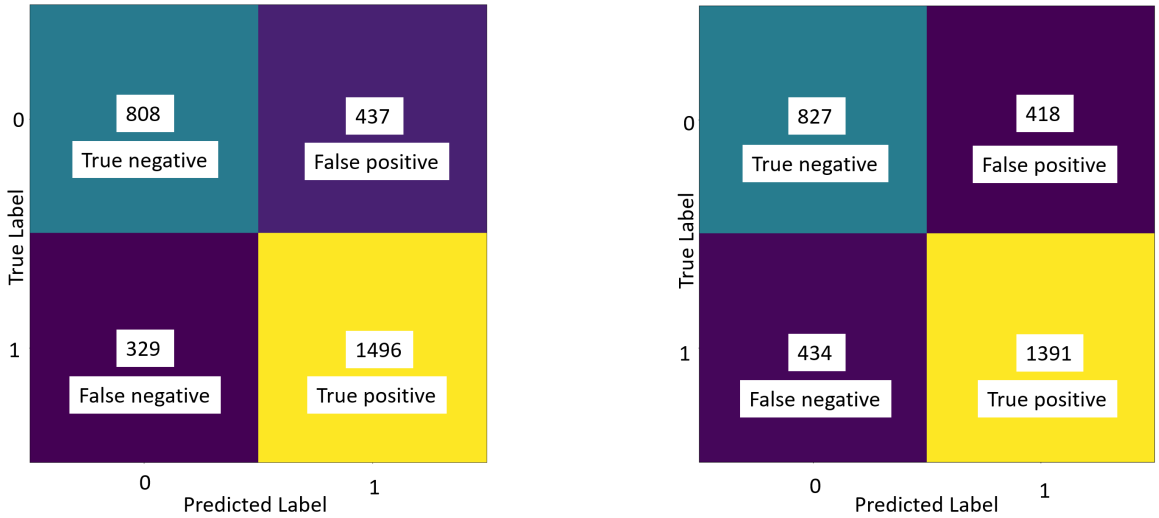

Figure 5.14: Confusion matrix for tripping predictions: left uses textual information, right do not. One indicates tripping did happen, zero indicates tripping did not happen.

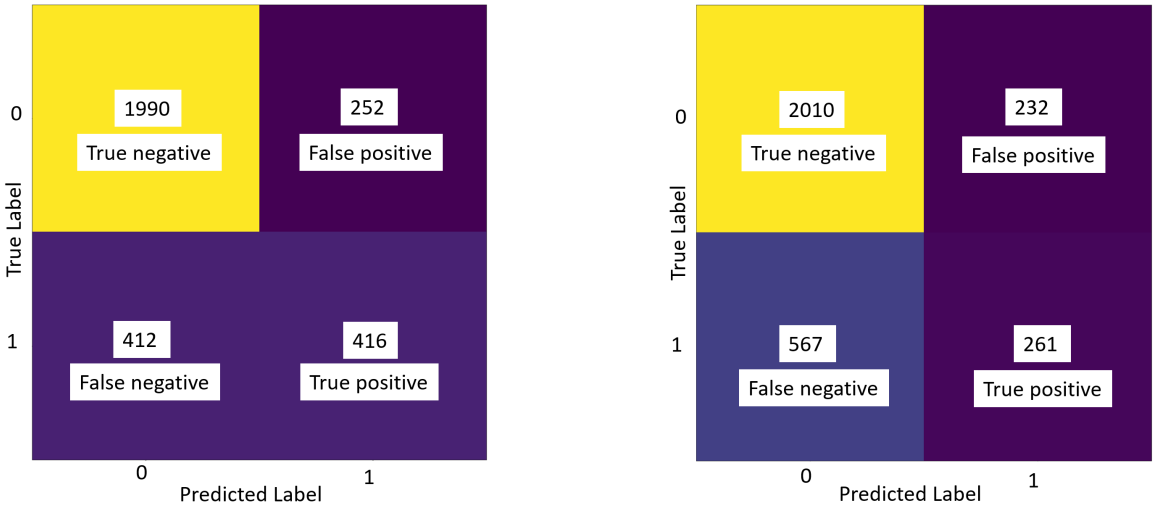

Figure 5.15: Confusion matrix for problem predictions: left uses textual information, right do not. One indicates problem did happen, zero indicates problem did not happen.

Chapter 6

Conclusion and Future Works

Although this study failed to developed an accurate ROP predicting deep neural network model, it outlines in detail the procedures on how to process, develop, and train a deep neural network model on geothermal drilling records. Due to the similarities between drilling in the geothermal industry and the oil and gas industry, this procedure can be rapidly adapted to use in oil and gas industry.

Due to the fact that the dataset used in this study is daily-averaged, it is hard to pinpoint the reason for the low performance of ROP prediction. As there many different drilling operations throughout a single drilling day, daily-averaged records cannot capture all of these operations. If higher resolution data were available, then better ROP predictions may be achievable.

This study also shows that using Bidirectional Encoder Representations from Transformers (BERT) can effectively process and encode textual information into a form that can be incorporated with normal numerical records. This enables the use of textual data in drilling optimization without costly manual preprocessing. If a sufficiently powerful neural network is used in modeling, then the encoded textual information will provide benefits when used with conventional numerical data.

Because all drilling records in this study are recorded in English, a BERT model pre-trained on the English language was used. However, there are also pretrained BERT models in other languages. If the drilling records are written in an unfamiliar language, then a suitable pretrained BERT can easily encode them into usable forms. This results in not only savings from not manually preprocessing the textual information, but also from not hiring foreign language translators to translate the records.

However, most pretrained BERT models are only trained on the general materials, and these models do not have much experience with specialized context (geothermal and geology in this case).

If there is a specialized BERT model trained on geology and geothermal materials, then the quality of the resulting numerical representation forms of textual information could be improved significantly.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015, cs.CV/1512.03385. xi, 22

[2] M. Grant Bingham. A new approach to interpreting rock drillability. *The Oil and Gas Journal*, 1964. 3

[3] Chiranth Hegde, Cesar Soares, and Ken Gray. Rate of penetration (ROP) modeling using hybrid models: Deterministic and machine learning. Presented at Unconventional Resources Technology Conference as URTeC-2896522, July 2018. 3

[4] A.T. Bourgoyne Jr. and F.S. Young Jr. A multiple regression approach to optimal drilling and abnormal pressure detection. In *Society of Petroleum Engineers Journal 14*, pages 371–384, 1974. 4

[5] Hamed reza Motahhari, G. Hareland, and J.A. James. Improved drilling efficiency technique using integrated PDM and pdc bit parameters. In *Journal of Canadian Petroleum Technology 49*, pages 45–52, 2010. 4

[6] Chiranth Hegde and Ken Gray. Evaluation of coupled machine learning models for drilling optimization. In *Journal of Natural Gas Science and Engineering 56*, pages 397–407, 2018. 5

[7] Yuanjun Li and Robello Samuel. Prediction of penetration rate ahead of the bit through real-time updated machine learning models. Presented at SPE/IADC International Drilling Conference and Exhibition as SPE/IADC-194105, March 2019. 5

[8] Ammar M. Alali, Mahmoud F. Abughaban, Beshir M. Aman, and Sai Ravela. Hybrid data driven drilling and rate of penetration optimization. *Journal of Petroleum Science and Engineering*, 200:108075, 2021. 5

[9] Omid Hazbeh, Saeed Khezerloo ye Aghdam, Hamzeh Ghorbani, Nima Mohamadian, Mehdi Ahmadi Alvar, and Jamshid Moghadasi. Comparison of accuracy and computational performance between the machine learning algorithms for rate of penetration in directional drilling well. *Petroleum Research*, 2021. 5

[10] H. Basarir, L. Tutluoglu, and C. Karpuz. Penetration rate prediction for diamond bit drilling by adaptive neuro-fuzzy inference system and multiple regressions. *Engineering Geology*, 173:1–9, 2014. 5

[11] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. 10

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020, cs.CL/2005.14165. 20, 39

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 21

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, cs.LG/1502.03167. 21

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014, cs.CV/1409.4842. 22

[16] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016, cs.CV/1608.06993. 23

[17] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks, 2017, cs.CV/1707.01629. 26

[18] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012, cs.NE/1207.0580. 27

[19] Xavier Gastaldi. Shake-shake regularization, 2017, cs.LG/1705.07485. 28

[20] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach, 2019, cs.CL/1907.11692. 39

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 40

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013, cs.CL/1301.3781. 40

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014, cs.NE/1412.3555. 40

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014, cs.CL/1409.3215. 41

[25] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015, cs.CL/1508.04025. 43

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017, cs.CL/1706.03762. 44

[27] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016, cs.CL/1508.07909. 49

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019, cs.CL/1810.04805. 51