



**Prediction of Incoming Work-In-Progress in Semiconductor Fabrication
Foundry using Long Short-Term Memory Recurrent Neural Network**

Tin Tze Chiang

Master of Science

2018

Prediction of Incoming Work-In-Progress Prediction in Semiconductor Fabrication
Foundry using Long Short-Term Memory Recurrent Neural Network

Tin Tze Chiang

A thesis submitted

In fulfilment of the requirements for the degree of

Master of Science

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2018

DECLARATION

I declare that the work in this thesis was carried out in accordance with the regulations of Universiti Malaysia Sarawak. It is original and is the result of my work, unless otherwise indicated or acknowledged as referenced work. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Name : Tin Tze Chiang
Metric No. : 15020052
Faculty : Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak
Dated : April, 2018

ACKNOWLEDGEMENT

The funding for this project is made possible through the research grant obtained from UNIMAS under the Special FRGS 2016 Cycle [Grant No: F08/SpFRGS/1533/2017].

I would like to express my sincere gratitude to my supervisor Dr. Chiew Kang Leng for our weekly meetings and his great supervision. His continuous guidance, support and encouragement have helped me to overcome the challenges encountered during the research process. His advices were crucial in completing this thesis.

I would like to extend my appreciation X-FAB Sarawak Sdn. Bhd. for their support in this research by providing the environment and resources to extract and compute for the relevant data for this project.

Finally, I would like to thank my parents, who has taught me the value of learning and not to give up on tough times and always showed me unwavering love. This work is dedicated to them.

ABSTRACT

Semiconductor process is one of the most complex processes in the manufacturing industries. In order to meet the high demands of semiconductor components, cycle time reduction in semiconductor manufacturing is crucial. One of the factors that can increase cycle time is the equipment preventive maintenance activity. Preventive maintenance activity is crucial to maintain or increase the operational efficiency and reliability of the equipment and minimises unanticipated failures due to faulty parts. Despite its criticality, preventive maintenance activity requires significantly long hours, thereby increasing cycle time. Therefore, this activity is usually performed when the equipment group is forecasted to have low incoming Work-in-Progress. However, current statistical forecasting approach has low accuracy because it lacks the ability to capture the time-dependent behaviour of the Work-in-Progress. In this work, we present an approach to Work-in-Progress arrival forecasting by using LSTM neural network. The LSTM forecasting model is trained using the historical incoming Work-in-Progress of an equipment group. The LSTM forecasting model is then required to forecast the incoming Work-in-Progress of three consecutive weeks to the same equipment group. The results of this study are compared with the results of the statistical method practised by the manufacturing production control engineers in the sponsoring company of this research. The comparison showed that the result proposed in this study is superior to the statistical method in the sponsoring company. The proposed model achieved value accuracy of 100%, 100%, 85.7% and trend accuracy of 66.7%, 100%, 50%, with respect to the three consecutive weeks of comparison.

Keyword: Semiconductor, cycle time, preventive maintenance activity, Work-In-Progress, forecasting, statistical method, LSTM

***Peramalan Ketibaan Kerja-dalam-Kemajuan di FAB Semikonduktor dengan
Menggunakan Pembelajaran Mesin
ABSTRAK***

Proses semikonduktor adalah salah satu proses yang paling kompleks dalam industri pembuatan. Untuk memenuhi permintaan komponen semikonduktor yang tinggi, pengurangan kitaran masa (CT) dalam pembuatan semikonduktor adalah penting. Salah satu faktor yang boleh meningkatkan CT ialah aktiviti penyelenggaraan pencegahan (PM). Aktiviti PM adalah penting untuk mengekalkan atau meningkatkan kecekapan operasi, kebolehpercayaan peralatan, dan meminimumkan kegagalan peralatan yang tidak dijangkakan. Aktiviti PM boleh meningkatkan CT sebab ia memerlukan masa yang panjang. Oleh itu, aktiviti PM biasanya dilaksanakan apabila ketibaan Kerja-Dalam-Proses (WIP) ke kumpulan peralatan diramalkan rendah. Tetapi, cara ramalan statistik semasa kurang tepat sebab ia tidak dapat memfaktorkan hubungan WIP terhadap masa. Dalam penyelidikan ini, satu cara peramalan ketibaan WIP menggunakan rangkaian saraf LSTM adalah dipersembahkan. Model peramalan LSTM dilatih menggunakan sejarah ketibaan WIP ke salah satu kumpulan peralatan. Kemudian, ia diuji dengan meramalkan ketibaan WIP tiga minggu yang berturutan ke kumpulan peralatan tersebut. Keputusan peramalan dibandingkan dengan keputusan peramalan kaedah syarikat penajaan penyelidikan ini. Perbandingan ini menunjukkan kaedah yang dicadangkan adalah lebih unggul berbandingkan cara syarikat penaja. Kaedah yang dicadangkan mencapai ketepatan nilai 100%, 100%, 85% dan ketepatan trend 66.7%, 100%, 50% untuk ketiga-tiga minggu perbandingan yang berkenaan.

Kata-kunci: *Semikonduktor, kitaran masa, aktiviti penyelenggaraan pencegahan, Kerja-Dalam-Kemajuan, peramalan, cara statistik, LSTM*

TABLE OF CONTENTS

	Page
DECLARATION.....	i
ACKNOWLEDGEMENT.....	ii
ABSTRACT.....	iii
<i>ABSTRAK</i>	iv
TABLE OF CONTENTS	v
LIST OF TABLES.....	viii
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xii
CHAPTER 1 INTRODUCTION.....	1
1.1 Problem Statement.....	6
1.2 Motivation.....	7
1.3 Research Objectives.....	7
1.4 Research Scope	8
1.5 Research Significance.....	8
1.6 Thesis Outline.....	9
CHAPTER 2 LITERATURE REVIEW.....	10
2.1 Research Background.....	10
2.2 Statistical Models	15
2.2.1 ARIMA.....	16
2.2.2 Exponential Smoothing	17

2.2.3	Holt-Winters Method	18
2.3	Machine Learning Models	19
2.3.1	Neural Networks	20
2.3.2	Gaussian Process	21
2.3.3	Support Vector Regression	22
2.3.4	K-Nearest Neighbours	23
2.4	Applications of Statistical Models and Machine Learning Models in Forecasting	25
2.5	Neural Networks Models	27
2.5.1	Feedforward Neural Network	32
2.5.2	Recurrent Neural Network	34
2.5.3	Deep Learning	36
2.6	Optimizers of Neural Network	37
2.6.1	Stochastic Gradient Descent	38
2.6.2	RMSprop	39
2.6.3	Adaptive Moment Estimation (Adam)	40
2.7	Regularization in Neural Network	41
2.8	Evaluation Metrics to Measure the Forecasting Results	43
2.9	Critical Summary	44
CHAPTER 3 METHODOLOGY		46
3.1	Motivation	46
3.2	Proposed Framework	48

3.3	Data Source.....	50
3.4	Data Acquisition	51
3.5	Feature Engineering.....	54
3.5.1	Data Formatting	54
3.5.2	Data Scaling	56
3.6	Long Short-Term Memory Model.....	56
3.7	Parameter Size Identification	63
3.8	Model Forecasting and Evaluation	67
3.9	Summary.....	73
CHAPTER 4 RESULTS AND ANALYSIS		75
4.1	Software Tools	75
4.2	Experiments Setup	77
4.3	Results	79
4.4	Discussion	96
4.5	Summary.....	100
CHAPTER 5 CONCLUSION.....		102
5.1	Summary of Research Contributions	102
5.2	Conclusion and Future Works.....	103
REFERENCES.....		105

LIST OF TABLES

		Page
Table 2.1	Common activation functions	28
Table 3.1	Current statistical WIP forecasting steps	47
Table 3.2	Data formatting for neural network	55
Table 3.3	Data formatting for neural network using $N = 20, n = 7, f = 7$	55
Table 3.4	Dimension of the data format in LSTM	62
Table 3.5	Modified reserved datasets with $\bar{x}$ as dummy value	68
Table 4.1	Parameter size selection results	79
Table 4.2	Value accuracy of forecasted result for 100 epochs, 4 batch size and 256 hidden neurons	84
Table 4.3	Trend accuracy of forecasted result for 100 epochs, 4 batch size and 256 hidden neurons	85
Table 4.4	Value accuracy of forecasted result for 100 epochs, 8 batch size and 256 hidden neurons	86
Table 4.5	Trend accuracy of forecasted result for 100 epochs, 8 batch size and 256 hidden neurons	87
Table 4.6	Value accuracy of forecasted result for 100 epochs, 12 batch size and 256 hidden neurons	88
Table 4.7	Trend accuracy of forecasted result for 100 epochs, 12 batch size and 256 hidden neurons	89
Table 4.8	Value accuracy of forecasted result for 100 epochs, 12 batch size and 512 hidden neurons	90
Table 4.9	Trend accuracy of forecasted result for 100 epochs, 12 batch size and 512 hidden neurons	91
Table 4.10	Summary table of value accuracy and trend accuracy	91
Table 4.11	Value accuracy of forecasted result for Fab method	94
Table 4.12	Trend accuracy of forecasted result for Fab method	95
Table 4.13	Forecast result comparison between LSTM and Fab	95

Table 4.14	Comparison of the modeling approaches of the proposed method and the FC method	96
Table 4.15	Advantages of PM planning using the proposed model over FC method	97

LIST OF FIGURES

	Page
Figure 1.1	2
Figure 1.2	3
Figure 1.3	5
Figure 2.1	11
Figure 2.2	14
Figure 2.3	14
Figure 2.4	27
Figure 2.5	33
Figure 2.6	35
Figure 2.7	41
Figure 2.8	42
Figure 2.9	42
Figure 3.1	49
Figure 3.2	52
Figure 3.3	53
Figure 3.4	58
Figure 3.5	59
Figure 3.6	65
Figure 3.7	65
Figure 3.8	65
Figure 3.9	66
Figure 3.10	66
Figure 3.11	71

Figure 3.12	Second example of graph plots for actual IWIP and predicted IWIP	72
Figure 3.13	Third example of graph plots for actual IWIP and predicted IWIP	73
Figure 4.1	Supervised-learning result for 100 epochs, 4 batch size and 256 hidden neurons	81
Figure 4.2	Supervised-learning result for 100 epochs, 8 batch size and 256 hidden neurons	81
Figure 4.3	Supervised-learning result for 100 epochs, 12 batch size and 256 hidden neurons	82
Figure 4.4	Supervised-learning result for 100 epochs, 12 batch size and 512 hidden neurons	82
Figure 4.5	Incoming WIP forecast using 100 epoch, 4 batch size and 256 hidden neuron	92
Figure 4.6	Incoming WIP forecast using 100 epoch, 8 batch size and 256 hidden neuron	92
Figure 4.7	Incoming WIP forecast using 100 epoch, 12 batch size and 256 hidden neuron	93
Figure 4.8	Incoming WIP forecast using 100 epoch, 12 batch size and 512 hidden neuron	93
Figure 4.9	Incoming WIP forecast using FC method	96

LIST OF ABBREVIATIONS

AC	Autocorrelation
AIC	Akaike Information Criterion
ANN	Artificial neural network
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ASR	Automatic speech recognition
BIC	Bayesian Information Criterion
BPTT	Back-propagation-through-time
CMP	Chemical mechanical planarization
CT	Cycle time
DBN	Deep-belief network
DNN	Deep neural network
DPML	Day per Mask Layer
FNN	Feedforward Neural Network
GP	Gaussian Process (GP)
IWIP	Incoming Work-In-Progress
kNN	<i>K</i> -Nearest Neighbour
LSTM	Long Short-Term Memory
MA	Moving average
MAE	Mean absolute error
MES	Manufacturing execution system
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
MRE	Mean relative error

MSE	Mean squared error
PAC	Partial Autocorrelation
PM	Preventive maintenance
RMSE	Root mean squared error
RNN	Recurrent neural network
RNN	Recurrent neural network
SAS	Simple Exponential Smoothing/Single Exponential Smoothing
SGD	Stochastic Gradient Descent
SRN	Simple recurrent network
SSE	Sum of squared errors
SVC	Support Vector Classification
SVM	Support Vector Machine
SVR	Support Vector Regression
TAT	Turn-around-time
WIP	Work-In-Progress

CHAPTER 1

INTRODUCTION

The semiconductor process is one of the most complex processes in the manufacturing industries with unique characteristics such as extremely long manufacturing process, complex series of hundreds of sequential processing steps, process steps dedication at designated tools, strict process window time frame, and dynamic product mix-run environment (Rozen & Byrne, 2016). These processes need to be completed almost flawlessly as defects in the manufactured items can bring about costly impacts in today's world of technologies where electronic devices play vital roles in daily living.

In general, semiconductor fabrication consists of seven major process steps. These process steps are photolithography, etching, chemical mechanical planarization (CMP), oxidation, ion implantation and diffusion. Briefly described, photolithography process is the process to transfer the pattern from a photomask to the surface of a wafer. By forming a layer of photoresist on the top of the wafer, the wafer is then sent to the etching process for the photoresist to be developed.

An etching mask is applied in this process to define the parts to remain and remove. At the deposition process, various layers of different materials is deposited on the wafer as part of the fabrication process. CMP process is a process used to plane the wafer surface using chemical slurry. This process is necessary because processes such as etching and deposition modify the wafer surface, leading to a non-planar surface. Oxidation is a process to convert silicon on the wafer into silicon dioxide, which is a layer to be used as masks for ion implantation process.

Ion implantation and diffusion process are used to introduce a controlled amount of impurities into the wafers to change their electrical properties. Ion implantation introduce

the dopant impurities into the crystalline silicon using electric fields and the diffusion process then moves the impurity atoms in the wafer at high temperatures. These steps are repeated until the required design is completed. The sequence of steps for the wafer to move through is defined by manufacturing execution system (MES). The defined steps are commonly called a process flow. Figure 1.1 further illustrates a non-exhaustive example of process flow of semiconductor fabrication process (Quirk & Serda, 2001).

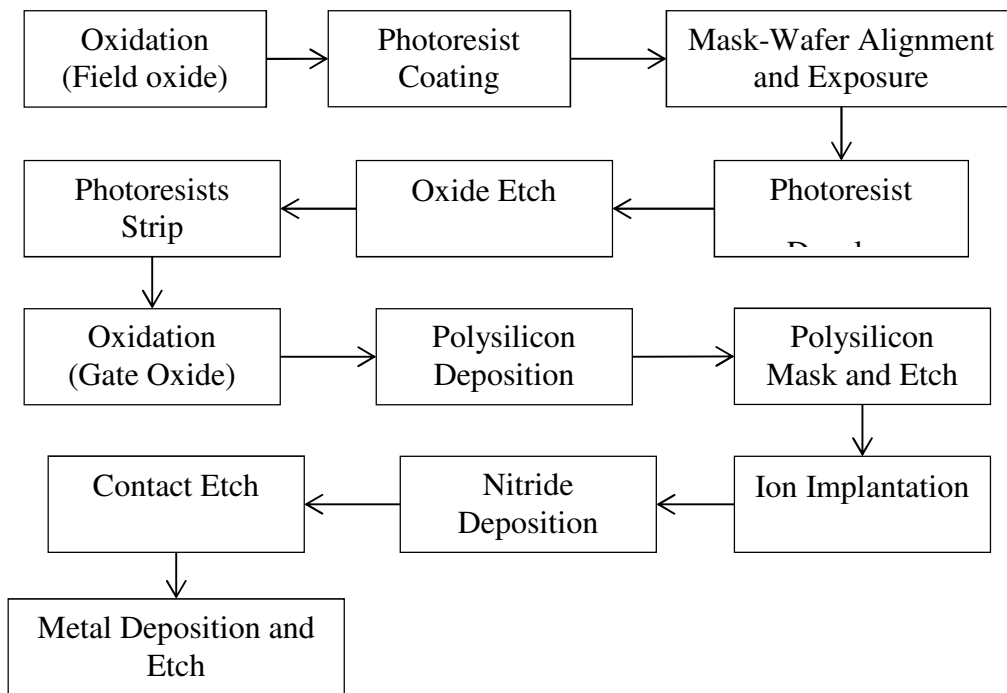


Figure 1.1: Example of semiconductor fabrication process flow

One of the main goals of semiconductor manufacturing plant, or commonly known as fab, is to reduce cycle time (CT) because reduced CT results in lower inventory costs, improved quality, and better response to demand changes and shorter time-to-market for new products (Rozen & Byrne, 2016). In the context of semiconductor manufacturing, CT is the total time it takes for a lot, which is a batch of wafers stored in a single cassette pod serve as the transportation means for the wafers production line, to be processed, plus the total time spent on waiting. Lots that are waiting to be processed in the production line are

commonly referred to as Work-in-Progress (WIP) in the manufacturing industries. The measurement unit for CT is Days per Mask Layer, (DPML), which is the number of days committed to process one photolithography layer.

One of the factors that can contribute to long waiting time, thereby increasing CT, is the equipment preventive maintenance activity. Preventive Maintenance (PM) is an activity that takes the entire machine off-line, or partially off-line, to carry out prescribed maintenance activity for maintaining or increasing the operational efficiency and reliability of the tool and minimizing unanticipated failures due to faulty parts (Ramirez-Hernandez et al., 2010). An unhealthy machine can cause various defects and chemical contaminations on the wafers. An affected wafer may need a rework by re-entering a sequence of chemical process steps to remove and fabricate the affected layers. Major defects or contamination could lead to scrapped wafer. Both of these activities incur losses to the company in both time and revenue. Most importantly, it affects the trust of the customers for potential future businesses.

In a fab environment, a set of machines is usually allocated for each process step to allow the lots to flow through in shorter time. The set of machines that perform the same wafer fabrication process step are commonly grouped logically and termed as equipment group. Figure 1.2 illustrates the logical grouping of six machines into an equipment group called Group A.

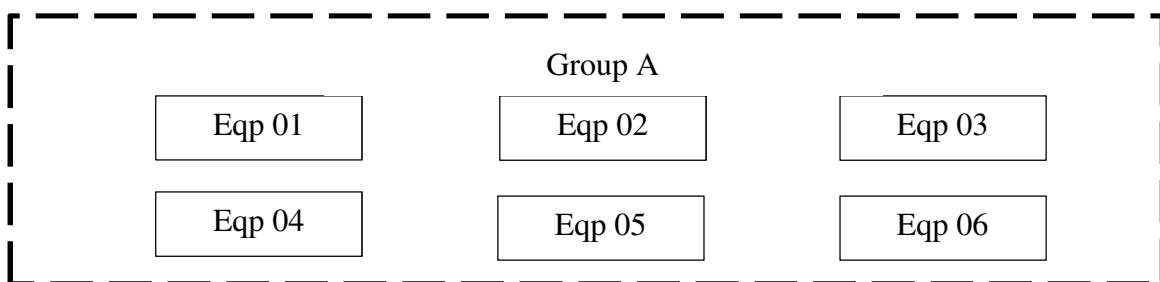


Figure 1.2: A logical group of equipment into an equipment group

Although PM activities are vital, PM downtime can be costly because it takes significantly long hours. As the complexity of the semiconductor manufacturing equipment increases, PM activities have also become more complex and would require longer time to complete. If one of the machines is taken offline for PM activities, the equipment group will experience insufficient capacity to process the incoming WIP. This will cause the production line to suffer WIP bottleneck that results in non-linear distribution of the WIP across the fab. Non-linear WIP distribution across the fab is highly undesirable because it will cause operational losses to the equipment at subsequent process steps as they become frequently idle due to the lack of WIP to process. In addition, WIP bottleneck also brings negative impact to the CT. This is because the CT duration will increase as the total waiting time of the impacted lots increases.

In the terminology of semiconductor manufacturing, the total wafers that each machine completed is called total moves of that machine, and the total wafers completed by the machines of an equipment group is called the total moves of that equipment group. Consider the example of logical group of six machines presented in

Figure 1.2, the total moves of the six machines in equipment group A on a particular day is the total moves of equipment group A for that day. The completed wafers from equipment group A are then moved to their designated process steps in the form of cassette pods. With this wafer movement, the completed wafers from equipment group A become the WIP of the next equipment groups. Figure 1.3 illustrates the typical relation of moves and WIP among equipment groups. In Figure 1.3, S_1 , S_2 , S_3 and W represents four different equipment groups in a fab. S_1 , S_2 and S_3 represents three equipment groups that have completed wafers designated to equipment group W . The wafers completed from S_1 , S_2 and S_3 are the moves of S_1 , S_2 and S_3 , respectively. These completed wafers from S_1 , S_2 and S_3 designated to be processed next at W will become the WIP of W .

It is also possible that the completed wafers from W becomes the WIP of W again.

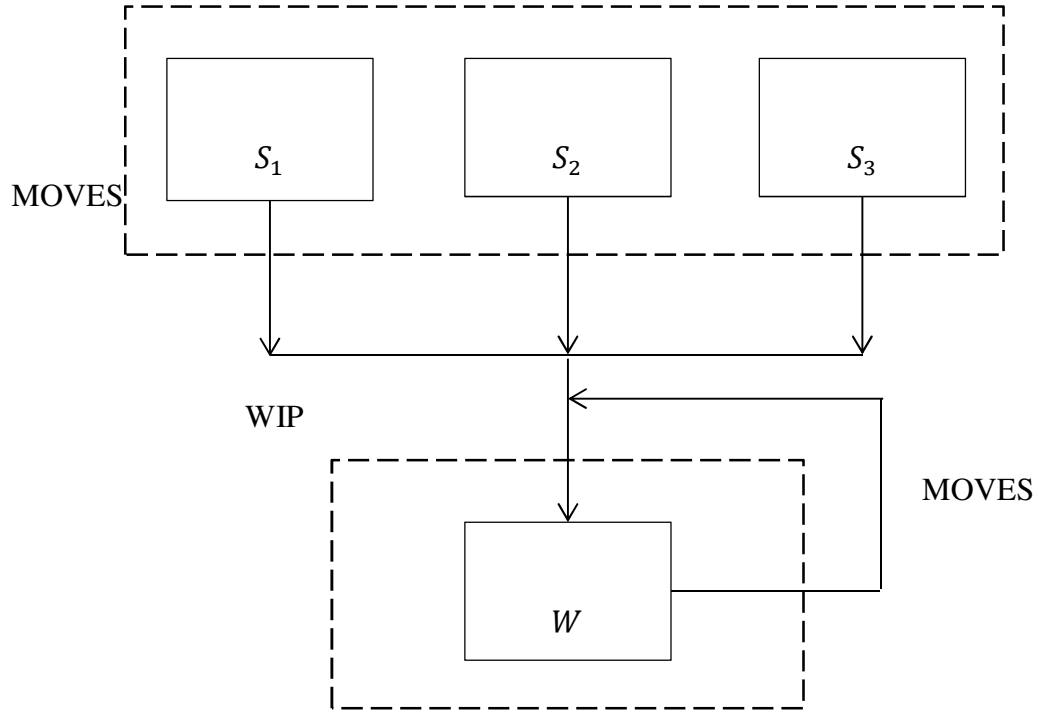


Figure 1.3: Typical relation of moves and WIP among fab equipment groups

In order to prevent long CT during PM activities is to perform the PM when the equipment group of the target machine is expected to have low incoming WIP. Referring to Figure 1.3, if one of the machines in W is scheduled for PM activity, it is desirable that the expected WIP arriving at W to be low. It is therefore essential to forecast the arrival of WIP to assist in PM activities planning for machines in W .

In semiconductor manufacturing, the WIP arrival forecasting commonly done by manufacturing production control engineers. A commonly to used approach to forecast the WIP arrival is through statistical aggregation based on the turn-around-time (TAT) of the wafers per operation step in the process flow. This is accomplished by first determine the product of interest to forecast, followed by retrieving the complete process flow of the product. For each of the process steps in the retrieved process flow, the TAT for an operation step is calculated, followed by calculating the rate of wafer flow in the

production. With both TAT and flow rate calculated, the CT for each operation step can be estimated. If a forecast of lot arrival for the next 24 hours is required, the engineer will first sum up the CT of each operation step of a lot of the product until it reaches 24 hours, and from there, examine the sum of estimated wafer to arrive at the step of interest.

1.1 Problem Statement

Although statistical forecasting method is commonly used to forecast the WIP arrival to an equipment group, the current statistical forecasting method contains the following limitations.

- i. Statistical forecasting approach lacks the ability to capture the time-dependent behavior of the WIP in the production lines. This is because statistical approach assumed that an observed series of data is caused by random process and each occurrence is independent from each other. However, such assumption is not true for WIP behavior in the production lines which is time series in nature.
- ii. Statistical forecasting approach is product-specific. Each product that is present in the production line needs to be modelled separately to create its forecasting model. The reason for this specificity is because each product has its own process steps sequence, and the statistical forecasting model is dependent on these process steps to perform forecasting. As a result of the separate modelling, statistical forecasting approach indirectly assumed that each product has dedicated resources such as manpower and equipment to cater to its production activity in the production environment. This assumption is not true as both manpower and equipment are shared among all the products in the production environment.

- iii. The assumptions of non-dependency in the time series data and non-resource sharing production environment reduced the accuracy of statistical forecasting model. The forecast results of most of the statistical forecasting models are not within the accuracies level where they are sufficient for critical decision making in the production environment, but can only serve as general guidelines or references.

1.2 Motivation

Recently, a new approach to arrival forecasting was introduced by using a type of recurrent neural network (RNN), called the Long-Short Term Memory (LSTM) neural network. LSTM has been proven to achieve higher accuracy in traffic flow prediction over statistical approach by various research works (Vlahogianni et al., 2005; Lv et al., 2015; Ma et al., 2015; Tian & Pan, 2015; Duan, Lv, & Wang, 2016; Fu et al., 2016). Besides traffic flow prediction, LSTM is also used in research such as log-driven information technology system failure prediction to discover long-range structure in historical data (Zhang et al., 2016), gesture recognition (Zhu et al., 2017), voice conversion (Lai et al., 2016), and aircraft engines excess vibration events predictions (ElSaid et al., 2016). Motivated by the cited works, this research work aims to investigate the efficiency of LSTM in predicting the incoming WIP arrival for an equipment group of interest as a mean to assist in PM activities planning. With high prediction accuracy, PM activities planning could be improved to avoid major obstacle to WIP movement in the production line and thus avoid negative impact to the CT of the fab.

1.3 Research Objectives

The main objective of this research is to develop an incoming WIP arrival forecasting model that can minimize occurrence of long CT due to PM activities and

overcome the limitation of the existing forecasting method. To achieve this, the specific objectives are defined as follows:

- i. To develop a generic forecasting model utilizing machine learning approach that can model the time-dependent behavior in the incoming WIP of all products in the production line.
- ii. To develop a forecasting model capable of forecasting the incoming WIP within the acceptable accuracy¹ of the fab to assist in PM activities planning.

1.4 Research Scope

This work is a joint-research with X-FAB Sarawak Sdn. Bhd. To provide a better presentation of the dissertation, X-FAB Sarawak Sdn. Bhd. is referred as the Fab. The work of this research covers the prediction of the incoming WIP arrival at the equipment group of interest selected by the Fab. The number of days to be forecasted is set to seven days as per the Fab requirement. The results of the developed forecasting model in this research work will be compared with the existing forecasting method of the Fab to assess its performance and accuracy. The purpose of the assessment is to determine its potential use in the actual production line.

1.5 Research Significance

The work of this research focuses on developing an incoming WIP arrival forecast model. With a forecasting model that can provide more accurate WIP arrival, the Fab manager will be able to plan the equipment's PM activities ahead to minimize the occurrence of bottlenecks in the production line due to PM activities that could bring negative impact on CT. This has direct positive impact on the revenue and customers trust

¹ Acceptable accuracy is set at ninety percent accuracies based on the Fab's requirement

of a Fab as lower CT allows more deliverables and on time shipment.

1.6 Thesis Outline

The thesis of this research is organized as follows. Chapter 2 presents the theoretical fundamentals of artificial neural network and recurrent neural network that are necessary to lay the foundations for the proposed solution of this research. Chapter 3 presents the proposed solution and the methodology of the research to develop the forecasting model. The architecture of Long-Short Term Memory (LSTM) model as the proposed recurrent neural network for the forecasting model will be presented. In Chapter 4, the experimental set-up and the results of the proposed forecasting model will be presented and discussed. The obtained results from the proposed forecasting model will be compared with the results of the current forecasting model of the Fab. Finally, Chapter 5 presents the concluding remarks and potential improvement for the future research.

CHAPTER 2

LITERATURE REVIEW

In this chapter, the background of this research is first presented in the next section, followed by the literature reviews of existing works in the subsequent sections.

2.1 Research Background

In the research domain of forecasting in semiconductor manufacturing, majority of the research works focus on forecasting the cycle time of the fab. For instance, Wang et al. (2018a) proposed an automatic factor selection to improve the prediction accuracy of CT forecasting of wafer lots. The authors presented that the ANN input factors for CT forecasting model in the past research works are either selected manually, or empirically. According to the authors, such approach seems arbitrary and unreliable to select input factors since it depends on artificial experience. Another approach to input selection uses the global situation of the whole fab such as the total WIP of the whole fab. According to the author, such approach is too complex to represent the wafer flow of the fab since it is influence by the interactions among the equipment properties, wafer properties, product mix and production control policies. Hence, in the authors work, input factor selection are accomplished through analyzing the collected data without artificial experience to improve the accuracy, scalability and comprehensibility of the CT forecasting models. Wang et al. (2018a) obtained 108 factors that are highly correlated with the CT of a wafer lot for their work. The results indicated that the proposed approach has higher accuracy than linear regression and Back Propagation Network in CT forecasting, especially in large scale data analysis.

In another research, Wang et al. (2018b) attempted short-term cycle time forecasting in re-entrant manufacturing systems. According to the authors, most previous

studies focus on estimating the whole CT in long-term time scales, which predicted the output time at the at the moment the wafers enter the production stage. However, considering the long production cycle (60,000 – 90,000 minutes) and dynamic production environment, such long-term prediction hardly meet the needs of decision making in production control. Based on the long-term forecasting comparison, the authors justified that CT forecasting in short time scales can provide more timely advice on production control, such as rebalancing work in process, changing dispatching rules, and job prioritizations.

In estimating the short-term CT of lots, the authors first established the foundation that semiconductor wafer fabrication systems (SWFSs) is a typical multiple re-entrant manufacturing system that fabricates layers of integrated circuits (IC). As such, the wafers repeat a series of operations to prepare one layer of ICs. In the studies that the authors conducted, for a typical 300 millimetre (mm) wafer fabrication, the wafer has 400-1000 processing operations, and each wafer repeatedly goes through deposit film, pattern film, etch film, wafer testing, and cleaning process for 20 – 40 times. Thus, the authors defined the short-term prediction interval for their work to be the CT of each re-entrance measured by the starting time difference of first two operations in adjacent continuous re-entrances.

Figure 2.1 is an extraction of the re-entrance modeling presented by Wang et al. (2018b).

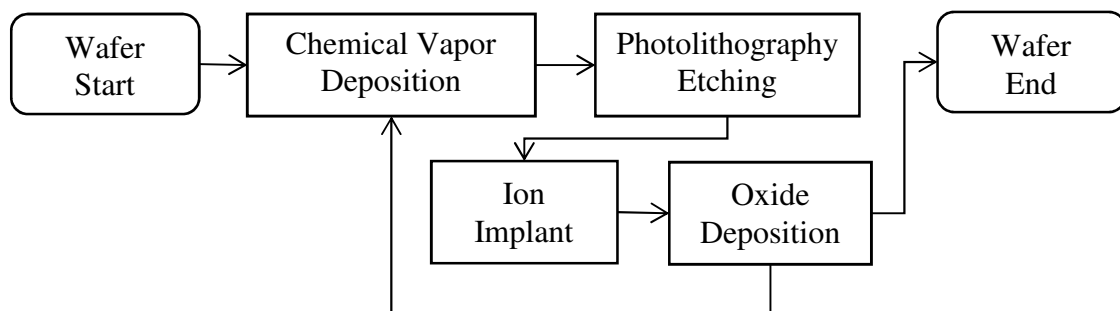


Figure 2.1: Semiconductor fabrication operation steps re-entrance modeling

Compared with long-term CT predictions, short-term CT forecast remains a difficult task due to two challenges. The first challenge is due to machine dedication constraint. The machine dedication constraint prescribes that once the first layer of wafer is operated on a specific machine, the remainder layers should be fabricated by the same machine. This constraint typically occurs for the photolithography operations. The second constraint is due to mask setup constraint. This constraint also occurs in the photolithography operations. In the photolithography process, since the required circuit patterns in the same re-entrance of two wafers of the same kind are the same, exposure conditions need identical mark, which is called mask setup constraint. In order to ensure that circuit patterns from different layers can be precisely connected together, the photolithography machine will calibrate the mask reticles to match the previous circuits. Therefore, the change of reticle mask to cater to wafers carrying various pattern requirements is a time-consuming task, and the correlation between the layer and the wafer need to be fully considered in the CT forecast of each re-entrance.

Scholl et al. (2011) presented their work on an implementation of a simulation-based short-term lot arrival forecast in a mature 200mm semiconductor fab, conducted in Infineon Technologies, Dresden. The authors' work forecast the lot arrival to the defect density measurement (DDM) work centre in Infineon Technologies. The problem domain of the authors' research is similar to this research where the intention of the lot arrival forecast is to avoid PM activities when the WIP is expected to be high.

The simulation system was developed by Infineon Technologies called D-SIMForecaster. In general, the forecasting simulation system is made up of three major software components. The first component is a component that queries production databases for relevant data, and processes the data into a consistent dataset. The second component is used to create and manage the simulation scenarios based on the dataset

prepared by the first component. Lastly, the third component exports the scenario into a file format that the simulation engine can understand, and import reports from the simulation run to be shown to the user. The simulation engine that performs the actual forecasting is a proprietary simulation engine created by Applied Material, called AutoSched AP (ASAP) (AutoSched AP, 2011). The simulations done by the authors resulted in high accuracy with 2% deviation from reality. The time horizon for the forecast was 5 days, and three subsequent horizons were used to evaluate the forecast.

The proposed model of Scholl et al. (2011) is very specific to Infineon Technologies and is impractical to apply in other fab. This is mainly due to the fact that the core of the simulation engine used in the research work to perform the arrival forecast is a proprietary simulation engine. As such, the forecasting method used is undocumented by the company for public references. In addition, operating methods and products modeled are specific to Infineon Technologies, which differs from other fab. It is also important to note that Infineon Technologies is a fab with fully-automated wafer transportation system using robotic systems as oppose to human operators in other fabs. The lot arrival time of a fully-automated wafer delivery system between each operation steps is highly consistent compare to human-operated delivery system. These factors made the comparison difficult across other fabs that do not have the same system and facilities.

Due to the lack of similar research works done in the domain of semiconductor fabrication, a cross reference to similar research problem in a different domain is necessary. From the literature reviews, vehicle traffic arrival forecasting exhibits the closest similarity to the forecast of WIP arrival in the fab. Consider the comparison of the following two models presented in Figure 2.2 and Figure 2.3. Figure 2.2 presents the scenario geometry of a typical traffic arrival modeled by Head (1995), while Figure 2.3 presents a typical WIP arrival scenario to an equipment group in a fab.

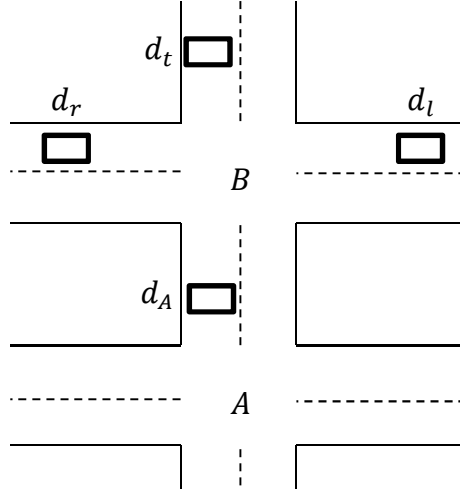


Figure 2.2: Geometric layout of traffic flow prediction scenario by Head (1995)

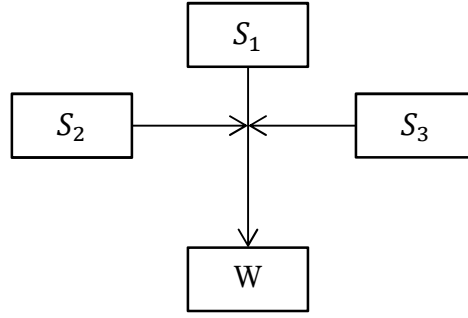


Figure 2.3: Typical scenario of WIP arrival to an equipment group in a fab

In Figure 2.2, d_r , d_t , d_l and d_A each represents a traffic detector deployed at their designated location, while A and B each denotes the two intersections of the road. It is desired to predict the traffic flow approaching intersection A at detector d_A where the actual traffic flow can be measured, so that the quality of the prediction can be assessed in real-time. In Figure 2.3, S_1 , S_2 and S_3 each denotes the equipment group that supplies lots to the equipment group W . S_1 , S_2 and S_3 are analogous to the three roads interconnected at intersection B that suppliers traffic to intersection A in Head's model. The lots that are supplied by S_1 , S_2 and S_3 becomes the WIP for W . The traffic in Head, L.'s model is therefore analogous to the WIP of W in this work. Similar to the work of Head (1995),

this research work would like to predict the total WIP that would arrive at W for a period of time in the future.

According to Head (1995), in general, traffic flow is a time-space phenomenon. Many of the subsequent traffic flow prediction works have also model the traffic flow prediction as a form of time series problem. A non-exhaustive of related works includes Dougherty et al. (1996), Williams & Hoel (2003), VanderVoort et al. (2005), Xie et al. (2007), Huang et al. (2014), Abadi et al. (2015), Tian & Pan (2015), Fu et al. (2016) and Shao & Soong (2016). The models used by the listed research works can be categorized into two categories. The first category uses statistical models, while the second category is uses machine learning models. In the next sections, the example of models from both categories will be introduced. Section 2.2 introduces relevant statistical models used in time series forecasting research, Section 2.3 introduces some of the machine learning models used in time series forecasting research, and Section 2.4 presents the comparison of the models from both categories in terms of their performance in time series forecasting.

2.2 Statistical Models

Statistical models are commonly known as parametric models which refer to models with fixed structure based on some assumptions and that parameters can be computed with empirical data (Fu et al., 2014), commonly through a series of mathematical analysis steps on the given time series data. The main characteristics of a time series is the dependency between the observations that are close to each other, the goal of statistical time series analysis is therefore to find such dependency and formulate them with mathematical equations that defines the forecasting model. In this section, we introduce three of such statistical forecasting models. These models are Autoregressive Integrated Moving Average (ARIMA), Exponential Smoothing and Holt-Winters method.

2.2.1 ARIMA

The acronym ARIMA stands for Autoregressive Integrated Moving Average. This model was introduced by Box and Jenkins in 1976 (Box et al., 2008). The model consists of three parts, which are autoregressive (AR), a differentiation, and moving average (MA). In the model, AR is represented with the parameter p , calculus differentiation is represented with the parameter d , and MA is represented with the parameter q . The complete notation of the model is then represented as ARIMA (p, d, q). The standard ARIMA model (Collobert & Weston, 2008; Goodfellow et al., 2013; Huval et al., 2013) of the process $Y(t)$, which is represented as ARIMA (p, d, q) is formulated as

$$\left(1 - \sum_{j=1}^p \phi_j B^j\right) (1 - B)^d Y(t) = \theta_0 + \left(1 - \sum_{j=1}^q \theta_j B^j\right) a(t) \quad (2.1)$$

where

- p = order of AR
- ϕ_j = coefficient of the AR component of ARIMA
- B^j = back shift operator
- d = order of calculus differencing to transform the data from non-stationary to stationary
- $Y(t)$ = the random process modelled on an observed time-series data
- θ_0 = deterministic trend term
- q = order of MA
- θ_j = coefficient of the MA component of ARIMA
- $a(t)$ = white Gaussian noise with zero mean and variance

The ARIMA model requires that the time-series data modeled to be stationary. A time-series is said to be stationary if the mean, variance and auto-covariance of the time

series are constant and independent of time factor, t . If a time-series is non-stationary, then a d order of differencing is applied to the time series to produce a more stabilized variances. The Autocorrelation (AC) and Partial Autocorrelation (PAC) analysis in the form of graphs are used to perform analysis for the order of d . In addition, AC and PAC are also used to determine for the orders of p and q in the ARIMA model. After the potential orders of p , d and q are identified through the AC and PAC analysis, the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) is then used to select the best order of ARIMA(p , d , q) by choosing the model that minimizes the criterion. Lastly the Maximum Likelihood Estimation (MLE) method is used to estimate for the remainder of the coefficients in the ARIMA's equation. A good ARIMA (p , d , q) model will produce residuals with the properties of zero mean distribution and zero time correlation.

2.2.2 Exponential Smoothing

Exponential smoothing forecasting method is one of the time series prediction algorithms based on statistical approach. The algorithm takes advantage of all historical data, and distinguishes them through the smoothing factor to let the recent data make a greater impact on the predictive value than long-term data (Ren et al., 2011). The advantage of this method is that it could save a lot of data and decrease the amount of the data that needs to be considered in the forecasting (Tang et al., 2008). The simplest form of exponential smoothing method is commonly called the Simple Exponential Smoothing or Single Exponential Smoothing (SES). This method is suitable for forecasting data with no trend or seasonal pattern. Assuming the observed time series as $x_1, x_2, x_3, \dots$, the formula of the SES model can be expressed mathematically as

$$s(t) = \alpha x(t) + (1 - \alpha)s(t - 1) \quad (2.2)$$

where $s(t)$ is the exponential smoothing value at time t , $s(t - 1)$ is exponential smoothing value of time at $t - 1$, $x(t)$ is the data value at time period t and α is a smoothing coefficient where $0 < \alpha < 1$.

The smoothing coefficient α represents the allotment value of the new and old data where its value shows different period factor to act as different function in the forecasting process. When the value of α is smaller, such as in between of 0.1 to 0.3, the influence of recent change is less, and the exponential smoothing value would represent long-term trend of time series. Whereas when the value of α is bigger, such as in between of 0.6 to 0.8, the model is more sensitive to the recent changes of the time series, and therefore the exponential smoothing value represents the recent trend of the time series (Tang et al., 2008).

2.2.3 Holt-Winters Method

The Holt-Winters method is actually a type of exponential smoothing method. However, unlike the SES method describe in Section 2.2.2, the Holt-Winters method can handle time series data with trend and seasonality. There are two approaches in the Holt-Winters method for forecasting, one is by multiplicative approach, and the other by additive approach. Holt-Winter method consists of three smoothing equations and one forecasting equation. Let L_t represents level, B_t represents the trend, S_t represents the seasonal component, t represents time, and let α , β , and γ represents the three smoothing parameters in the Holt-Winters method, the three smoothing equations for level, trend, and seasonal are expressed respectively as

$$L_t = \alpha \left(\frac{Y_t}{S_{t-s}} \right) + (1 - \alpha)(L_{t-1} + B_{t-1}) \quad (2.3)$$

$$B_t = \beta(L_t - L_{t-1}) + (1 - \beta)B_{t-1} \quad (2.4)$$

$$S_t = \gamma \left(\frac{Y_{t-p}}{L_{t-p}} \right) + (1 - \gamma) S_{t-p} \quad (2.5)$$

where p is the seasonal cycle period.

The difference between the multiplicative and the additive methods is in their forecasting equation. The equations below express the additive and multiplicative forecasting equations, respectively.

$$F_t = B_t + L_t + S_t \quad (2.6)$$

$$F_t = (B_t + L_t) \cdot S_t \quad (2.7)$$

Hyndman & Athanasopoulos (2012) made suggestions on the criterion to select an additive or multiplicative model. Hyndman et al. suggested that the additive model is preferred for time series with nearly constant seasonal variations throughout the whole period. In contrast, multiplicative model is suitable for varying seasonal variations.

2.3 Machine Learning Models

Machine learning models are also referred to as non-parametric models, where such models have no fixed structure and parameters (Fu et al., 2016). It is also known as data-driven model. Machine learning models are very popular in the areas of pattern recognition, classification and prediction. In the area of prediction, machine learning methods are capable of finding out the new values that follows from the last observed data point. Machine learning models have gained much attention from researches to solve time series problem because of their ability to address stochastic and nonlinear nature of time series problem compared to parametric models (Tian & Pan, 2015). In this section, we review some of the machine learning models used in time series forecasting research. The selected machine learning models for review are Neural Networks, Gaussian Process, Support Vector Regression and K -Nearest Neighbour.

2.3.1 Neural Networks

A neural network is a network of nodes, called neurons, organized into input layer containing input nodes, intermediate layers called hidden layers with hidden nodes and an output layer containing output nodes. Information moves from input nodes, pass through the hidden nodes and ends on the output node. The standard neural network has only one hidden layer, which is also described as the simplest architecture of neural network. This simple architecture is termed as feedforward neural network. The standard neural network can be represented mathematically as

$$m(x) = \alpha_0 + \sum_{j=1}^H \alpha_j \varphi(w_j^T x') \quad (2.8)$$

where x' is the input vector x that is augmented with 1, i.e., $x' = (1, x^T)^T$, w_j is the weight vector for the j th hidden node, $\alpha_0, \alpha_1 \dots \alpha_H$ are the weights for the output node and H is the number of hidden nodes. The function φ is the output of the hidden node. The number of hidden nodes, H , controls the complexity of the neural network model. When $H = 0$, the neural network reduces to a linear model. The weights of hidden nodes are generally estimated using optimization algorithm. The most popular optimization algorithm is backpropagation algorithm (Rumelhart & McClelland, 1987; Taieb, 2014). Typically, the weights are initialized with random values that are near to zero. The backpropagation algorithm will then update the weights to minimize the prediction errors. The error function minimized by the neural networks is nonconvex which can have multiple local minima. Therefore, the final solution will depend on the initial chosen value. Due to this randomness, neural networks are often trained for multiple times using different randomized initialization values, and the outputs of the neural network are averaged to obtain the final predictions.

2.3.2 Gaussian Process

Gaussian Process (GP) is a machine learning model developed by Rasmussen & Williams (2006) that is widely used in time series forecasting. GP is a probability distribution over functions f . More formally, according to Rasmussen & Williams (2006), GP is a collection of random function variables $f = \{f_1, f_2, \dots, f_N\}$, any infinite number of which have a joint Gaussian distribution.

A GP is fully determined by a mean m function and a kernel/covariance function k . Let a particular finite subset of the random function variables f that follows the same expression as $f = \{f_1, f_2, \dots, f_N\}$ and the corresponding inputs (indices) $X = \{x_1, x_2, \dots, x_N\}$, GP can be expressed as

$$f(x) \sim GP(m, k) \quad (2.9)$$

and the prediction distribution p is expressed as

$$p(f(x_1), \dots, f(x_n) | x_1, \dots, x_n) = \mathcal{N}(m, K) \quad (2.10)$$

where $\mathcal{N}(m, K)$ denotes a Gaussian distribution with K represents the full covariance matrix of the function values $f(x_1), \dots, f(x_n)$, and m is the corresponding mean vector.

To simplify the notation and calculation, the mean $m(x)$ of GP is taken to be zero, and thus Equation (2.9) and (2.10) are written as

$$f(x) \sim GP(0, K) \quad (2.11)$$

$$p(f(x_1), \dots, f(x_n) | x_1, \dots, x_n) = \mathcal{N}(0, K) \quad (2.12)$$

In time series regression, given a training data set D of N observation, where $D = \{(x_i, y_i) | i = 1, \dots, N\}$, the goal is to predict a new y_* given x_* using $f(x)$ such that $y_i = f(x_i) + \delta_i$ where δ_i is a Gaussian noise with mean zero and variance σ^2 . The prior

distribution of the observed target y is given by

$$y \sim \mathcal{N}(0, K) \quad (2.13)$$

The predictive distribution of y_* can be computed by conditioning on the training data set to get $p(f(x_*)|x_*, D)$. The joint distribution described by Rasmussen et al. over y and predictions of x_* is therefore given by

$$\begin{bmatrix} y \\ f(x_*) \end{bmatrix} \sim \mathcal{N} \left(0, \begin{bmatrix} K(X, X) & K(X, x_*) \\ K(x_*, X) & K(x_*, x_*) \end{bmatrix} \right) \quad (2.14)$$

where X is $(n \times m)$ matrix of input in the training data.

2.3.3 Support Vector Regression

The Support Vector Regression (SVR) can be regarded as an extension of Support Vector Machine (SVM). SVM is an optimization technique developed by Vapnik et al. (1995) that attempts to find a hyperplane in the original input space to separate a given training set correctly and leave as much distance as possible from the closest instances to the hyperplane on both sides (Ojemakinde, 2006). In SVM literature, when SVM is used to solve classification problems, it is called Support Vector Classification (SVC) and when it is used to solve regression problems, it is called Support Vector Regression (SVR).

The concept of SVR is to estimate the nonlinear function between input vector x and output variable y . Using a nonlinear mapping $x \rightarrow \Phi(x)$ where Φ denotes a transformation function, the relationship between x and y can be converted to linear regression problem. Let $f(x)$ denotes the function that describe the linear regression relationship between $\Phi(x)$ and y , w denotes a weight vector, and b denotes the bias, the mathematical expression can be written as

$$f(x, w) = w \cdot \Phi(x) + b \quad (2.15)$$

The objective of SVR is therefore to find an optimal w , b and the appropriate parameters in $\Phi(x)$ that gives the function $f(x, w)$ to have the most error, ε , deviated from the actual targets for all the training data, and at the same time it is as flat as possible. The flatness of the $f(x, w)$ is achieved by seeking for a small w by minimizing the $\frac{1}{2} \|w\|^2$. These can be expressed mathematically as

$$\begin{aligned} & \text{minimize } \frac{1}{2} \|w\|^2 \\ & \text{subject to } \begin{cases} y - \langle w, \Phi(x) \rangle - b \leq \varepsilon \\ \langle w, \Phi(x) \rangle + b - y \leq \varepsilon \end{cases} \end{aligned} \quad (2.16)$$

In the event that the function $f(x, w)$ could not be found, slack variables ξ_i, ξ'_i are introduced for measurements of higher and lower error level respectively to allow for error levels greater than ε , arriving to the mathematical expressions

$$\begin{aligned} & \text{minimize } \frac{1}{2} \|w\|^2 + C \sum_{i=1}^I (\xi_i + \xi'_i) \\ & \text{subject to } \begin{cases} y - \langle w, \Phi(x) \rangle - b \leq \varepsilon \\ \langle w, \Phi(x) \rangle + b - y \leq \varepsilon \\ \xi_i, \xi'_i \geq 0, n = 1, 2, \dots, N \end{cases} \end{aligned} \quad (2.17)$$

where C measures the trade-off between generalization ability and accuracy in the training data and N is the total number of the training data point.

2.3.4 K -Nearest Neighbours

The K -Nearest Neighbour (kNN) model is both a nonlinear and nonparametric model where the prediction for a data point is obtained by averaging the target outputs of the K nearest neighbour points. In other words, the prediction for input x is computed as

$$m(x) = \frac{1}{K} \sum_{k=1}^K y_{NN}(x, k) \quad (2.18)$$

where $y_{NN}(x, k)$ is the output of the k -th nearest neighbour of the input vector x in the data set. The key parameter K has to be selected with care since it is controlling the bias, or variance, of the trade-off of the estimates. A large K will lead to smoother fit and therefore a lower variance but at the expense of higher bias, and vice versa for a small K . Despite its simplicity, kNN model yields good results in practice, and is often used to provide a benchmark to more complex models such as neural networks.

The most commonly used metric to compute the nearest neighbors of the input vector x is the standard Euclidean distance expressed as

$$d(x_i, y_i) = \sqrt{\sum_{i=1}^n ((x_i) - (y_i))^2} \quad (2.19)$$

where d denotes the distance between two points, x and y , in the Euclidean space, and i denotes the subscript index of the points in the Euclidean space, and n denotes the total point to be calculated. This allows the target function to be either discrete-valued or real valued, and therefore, the kNN method can be used for both classification and regression problems. Consider the problem is regression related, a training algorithm for continuous-valued target function can be described in two steps. Firstly, for a given training example $(x, f(x))$, the examples are added to the list of training examples, and secondly, in order to predict a given query instance with input vector x_q , the following function is used

$$f(x_q) = \frac{\sum_{i=1}^k f(x_i)}{k} \quad (2.20)$$

where $x_1, \dots, x_k$ denote the k instances from training examples that are nearest to x_q .

With kNN model, the potential problem that can arise is that distance between the neighbors may be dominated by some irrelevant attributes in the data. When there are many such irrelevant attributes in the data, the phenomena called curse of dimensionality

might occur. Another problem that could arise is the high computation cost. This is because for each new query instance, the distance of the new query instance to all training samples must be calculated.

2.4 Applications of Statistical Models and Machine Learning Models in Forecasting

Time series forecasting has been widely studied in statistics and econometrics. It has also been influenced by linear statistical models such as Autoregressive Integrated Moving Average (ARIMA) model (Gooijer & Hyndman, 2006). However, due to the fact that time series in real world phenomena typically behave nonlinearly (Kantz & Schreiber, 2004), it became increasingly clear in the late 1970s and 1980s that linear models are not being adapted to many real-world applications (Gooijer & Hyndman, 2006). Nonlinear time series, such as bilinear model (Poskitt & Tremayne, 1986) and the threshold autoregressive model (Tong & Lim, 1980) have been proposed. However, according to (Gooijer & Hyndman, 2006; Taieb et al., 2012; Fischer et al., 2018), nonlinear time series analysis and forecasting is still in its infancy compare to the development of linear time series. According to Schwartz & Rivera (2009), real-life semiconductor manufacturing processes are highly stochastic and nonlinear.

In the last two decades, machine learning have emerged and established themselves as serious contenders to classical statistical models in the forecasting domain (Zhang, 1998; Palit & Popovic, 2005; Ahmed et al., 2012). These models, which are black-box in nature and data-driven, are also referred to as nonparametric models where there is no fixed structure and parameters. Machine learning models assume that the data has been generated from some unknown data generating process and different learning algorithms are used to approximate it (Taieb, 2014). These models use only historical data to learn the stochastic dependency between the past and the future (Taieb et al., 2012). Various

research works have demonstrated that machine learning models performs better than classical statistical models in forecasting, for instance, the use of artificial neural network (ANN) (Werbos, 1974; Lapedes & Farber, 1987; Werbos, 1988; Abhisheka et al., 2012; Oancea, 2014), Support Vector Machine (Ojemakinde, 2006), K-nearest neighbour (Hastie et al., 2009; Alpaydin et al., 2010), and Long Short-Term Memory (LSTM) (Fu, 2016). According to Taieb (2014), forecasting is primary dominated by ANN.

Ahmed et al. (2012) presented a large scale comparison study of time series forecasting for Multilayer Perceptron (MLP), Bayesian neural networks, radial basis functions, generalized regression neural network or kernel regression, K-nearest neighbour regression, CART regression trees, support vector regression, and Gaussian processes machine learning models by using M3 competition data. The M3 competition is the latest in a sequel of M forecasting competitions which consists of 3003 business-type time series. It has become an important benchmark for testing and comparing forecasting model (Ahmed et al., 2012). According to the authors, such extensive study has not been done for regression or time series problems. The results of the comparison showed that the overall ranking of the models from best to worse are multilayer perceptron, Gaussian processes, followed by Bayesian neural networks and support vector regression with close similarity, and generalized regression neural networks and K-nearest neighbor regression are almost the same, CART regression trees, and in the last spot is radial basis functions. This research has contributed significant reference to the selection of machine learning models for solving time series forecasting.

In the next section, a detail discussion on the fundamentals of neural network, and the discussion on the types of neural networks that are available in general are presented. Next, gradient descent optimization algorithms are introduced. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to

optimize neural networks (Ruder, 2016). Subsequently, the regularization method that is used in neural network to avoid overfitting during its training phase is presented. Lastly, the evaluation metrics that are commonly used in forecasting research is presented.

2.5 Neural Networks Models

A neuron is the fundamental information processing unit of a neural network.

Figure 2.4 depicts the model of a neuron.

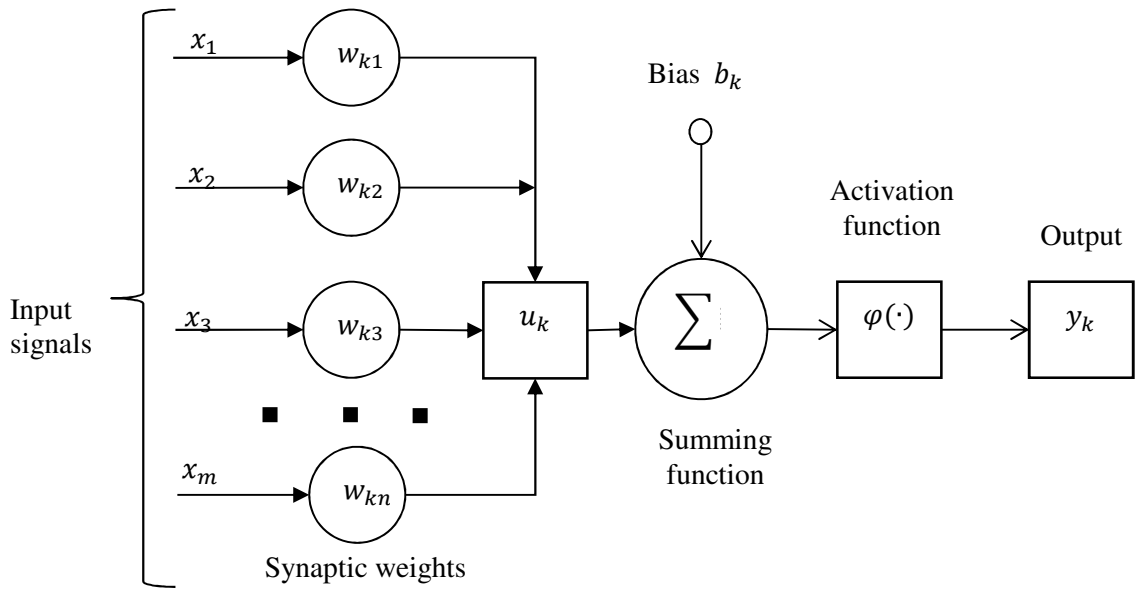


Figure 2.4: A neuron

It consists of a set of synapses x_i which has its own weight w_{ki} , a summing function that adds up the weighted input signals according to the respective synapses, an external bias b_k and an activation function $\varphi(\cdot)$ that limits the amplitude of the output signal to a permissible range. To formulate the neuron mathematically, let $x_1, x_2, \dots, x_m$ be the input signals, $w_{k1}, w_{k2}, \dots, w_{km}$ are the respective synaptic weights of neuron k , m be the total number of input signals, bias denoted by b_k , u_k as the linearly combined output from the input signals, $\varphi(\cdot)$ as the activation function, and y_k is the output signal of the neuron k , the neuron k can be expressed mathematically as

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (2.21)$$

$$v_k = u_k + b_k \quad (2.22)$$

$$y_k = \varphi(u_k + b_k) \quad (2.23)$$

By modelling the bias b as one of the synapse where

$$w_{k0} = b_k \quad (2.24)$$

we can formulate the combination of Equations (2.22) to (2.23) as

$$v_k = \sum_{j=0}^m w_{kj} x_j \quad (2.25)$$

$$y_k = \varphi(v_k) \quad (2.26)$$

The activation function $\varphi(\cdot)$ scales the activation of a neuron into an output signal.

The most common activation functions are listed in Table 2.1. Note that most of the activation functions are non-linear. This allows neural networks to model non-linear equations.

Table 2.1: Common activation functions

Function	Formula	Values range
Linear	$\varphi(v) = mv, m \in \mathbb{R}_+$	$(-\infty, \infty)$
Sigmoid	$\varphi(v) = \frac{1}{1 + e^{-\alpha v}}, \alpha \in \mathbb{R}$	$(0, 1)$
Logistic	$\varphi(v) = \frac{1}{1 + e^{-v}}$	$(0, 1)$
Hyperbolic tangent	$\varphi(v) = \frac{e^{\alpha v} - e^{-\alpha v}}{e^{\alpha v} + e^{-\alpha v}}, \alpha \in \mathbb{R}$	$(-1, 1)$
Exponential	$\varphi(v) = e^{-\alpha v}, \alpha \in \mathbb{R}$	$(0, \infty)$

Neural networks must first be trained with relevant data, called training set, to build the model that best characterizes the problem. When an input vector x is fed into the neural network, a predicted value of $\hat{y}$ is computed at the output layer using a cost function. The two widely used cost functions are sum of squared errors (SSE) and root mean squared error (RMSE). These two cost functions can be expressed mathematically as

$$SSE = \sum_{n=1}^N (y_n - \hat{y}_n)^2 \quad (2.27)$$

$$RMSE = \sqrt{\frac{\sum_{n=1}^N (y_n - \hat{y}_n)^2}{N}} \quad (2.28)$$

where y denotes the actual value, $\hat{y}$ denotes the predicted value, n denotes the subscript index, and N denotes the total number of values to compare. The gradient descent based methods are commonly used to minimize the cost function in neural network. The intuitive of this approach is that for a given weight configuration, the weights are iteratively adjusted by moving in the direction of the negative slope until the cost function reaches the minimum. Each iteration of the gradient descent method is called an epoch, and the number of input vector x used for each iteration can be either a single vector or a batch of vectors where the maximum size of the input batch is not greater than the size of the training set. An efficient algorithm based on this method that is widely used to compute the gradients for all weights in the neural network is the backpropagation algorithm (Werbos, 1988; Parascandolo et al., 2016).

To illustrate backpropagation algorithm mathematically, let index variable i represents neuron in input layer, j represents neuron in hidden layer, and k represents neuron in output layer in the context of this research. The adjustment of the weight in the gradient approach described can be represented mathematically as

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} \quad (2.29)$$

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n+1) \quad (2.30)$$

where η is the learning-rate parameter of the algorithm and E denotes the cost function.

This method is also known as the stochastic gradient descent.

To illustrate the backpropagation algorithm, consider a neuron j being fed by a set of function signals produced by its preceding layer of neurons, the induced local field $v_j(n)$ produced at the input of the activation function of neuron j is therefore

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n) \quad (2.31)$$

where m is the total number of inputs applied to the neuron j , excluding the bias. The synaptic weight w_{j0} equals the bias b_j applied to the neuron j . Therefore, the function signal $y_j(n)$ produced at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \quad (2.32)$$

Let e denotes the error signal from neuron j . The mathematical definition for e is

$$e_j(n) = d_j(n) - y_j(n) \quad (2.33)$$

where d is the desired output of neuron and y denotes the function signal produced at the output of neuron j in the output layer.

The algorithm applies a correction $\Delta w_{ji}(n)$ to synaptic weight $w_{ji}(n)$, which is proportional to the partial derivative $\partial E(n)/\partial w_{ji}(n)$. This gradient, which defines the direction of search in the weight space for the synaptic weight w_{ji} , can be expressed using chain rule as (Haykin, 2009)

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \quad (2.34)$$

where

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \quad (2.35)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \quad (2.36)$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n)) \quad (2.37)$$

Equation (2.34) can therefore be written as

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))y_i(n) \quad (2.38)$$

Δw is therefore

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad (2.39)$$

where

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)) \quad (2.40)$$

$\delta_j(n)$ is the local gradient required to change the synaptic weights.

In the event that neuron j is an output neuron, the Equation (2.40) remains the same to calculate for the gradient. In the event that neuron j is a hidden neuron, the equations below represent its local gradient derivatives.

$$\begin{aligned} \delta_j(n) &= \frac{\partial E(n)}{\partial v_j(n)} \\ &= \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial E(n)}{\partial y_j(n)} \varphi'_j(v_j(n)) \end{aligned} \quad (2.41)$$

Using chain rule for partial derivative $\frac{\partial E(n)}{\partial y_j(n)}$, the following equation is obtained

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \quad (2.42)$$

With neuron k as an output neuron, $e_k(n)$ and $\partial e_k(n)/\partial v_k(n)$ are therefore

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)) \end{aligned} \quad (2.43)$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n)) \quad (2.44)$$

Differentiating Equation (2.44), we obtain

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \quad (2.45)$$

Finally,

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j(n)} &= - \sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n) \\ &= - \sum_k \delta_k(n) w_{kj}(n) \end{aligned} \quad (2.46)$$

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (2.47)$$

where j is the hidden neuron.

2.5.1 Feedforward Neural Network

A feedforward neural network (FNN) is a type of neural network where each layer only receives input from its previous layer. It is said to have no feedback connections. Such neural network is a directed graph, where information is always traveling forward. The simplest type of FNN made up of three layers of neurons, which are input layer containing the input neuron, the hidden layer containing the hidden neuron and the output layer containing the output neurons. This type of FNN is commonly known as the Multilayer Perceptron (MLP), and is depicted in Figure 2.5 as a directed graph. In Figure 2.5, each node represents the neurons and the edges represent connectivity among neurons, while the arrows at the edges represent directions for which the signal flows.

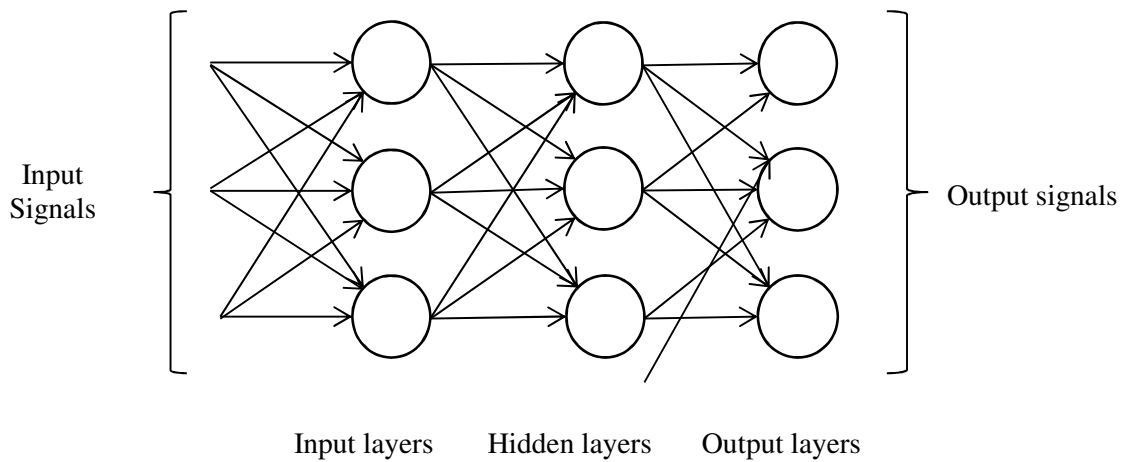


Figure 2.5: Feedforward Neural Network

The input neurons, which forms the input layers, are neurons that takes input signals as vector. Input neurons do not perform any computation on the input vector; they only pass the input vector to the subsequent layer of neurons, which are the hidden neurons. The hidden neurons, which form the hidden layers, are the basis of the neural networks. These neurons receive signals from the input neurons or previous hidden neurons should the neural network has multiple hidden layers and performs transformation on them according to the Equations (2.21) and (2.22). These neurons act as feature detectors that gradually discover the salient features in the given inputs. The computed signals are then passed to subsequent layer, which is either a hidden neurons or an output neurons. The output neurons, which forms the output layers, receive signals from their preceding neurons and likewise performs transformation on the signals using Equations (2.21) and (2.22). The resulting values represent output of the entire neural networks.

An argument in favour of MLP's simplicity and yet expressive capability is given by the universal approximation theorem, which states that a MLP with a single hidden layer and sufficient amount of neurons can approximate any measurable function regardless of the activation function used and the dimension of the inputs (Hornik et al.,

1989). However, the number of hidden neurons required is unknown and can be impracticably large. MLPs' hidden layers are often extended to multiple layers to create an architecture termed deep neural network (DNN), which will be described in Section 2.5.3. The advantage of such approach is that it allows the MLP to partition the input space into exponentially more linear regions, allowing MLP to represent highly structured and complex function more easily (Montufar et al., 2014). This is usually needed in the area of pattern recognition task such as speech recognition (Mohamed et al., 2012).

In a FNN, all samples are processed independently of each other. Due to the lack of context information, FNNs cannot directly process sequential input such as audio, video and time series related data. The often used approach to provide context information to the input is to use a fixed-size window to concatenate the current feature vector with previous and future vectors. However, this approach presents substantial shortcomings, such as unknown optimal window size, fixed context availability, and increased dimensionality which will impose more data, longer training time and larger models. Recurrent neural network (RNN) described in the following sections, are more suitable to represent sequential inputs.

2.5.2 Recurrent Neural Network

The RNN distinguishes itself from the FNN in that it is a neural network with feedback connection, where neurons feed their output signal back to the input of all the other neurons, except the neurons in the input layer. One of the simplest and most popular RNN architecture is the Elman's simple recurrent network (SRN) (Elman, 1990) depicted in Figure 2.6 where the layers' representation are simplified by texts.

The SRN resembles a simple feedforward network, but with addition of a context layer through the introduction of the feedback connection. The context layer is a copy of

the hidden layer's activation in previous time step. It acts as the network's memory of previous activity. Therefore, in a RNN, the hidden layer has two sources of input, which are the input layer and the context layer. The context layer's information allows the hidden neurons to have some information of their prior activations, which enables the network to perform learning tasks that extend over time.

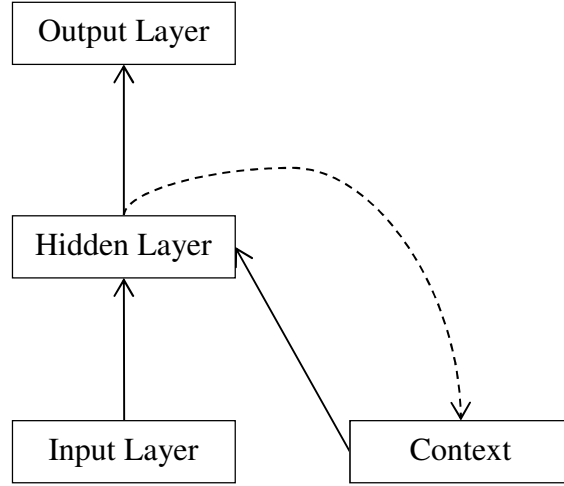


Figure 2.6: Recurrent Neural Network

For a sequence input of vectors $\{x_1, x_2, \dots, x_T\}$, a simple RNN computes a sequence of hidden activities $\{h_1, h_2, \dots, h_T\}$ and output vectors $\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T\}$ as

$$h_t = f(W^{xh}x_t + W^{hh}h_{t-1} + b^h) \quad (2.48)$$

$$\hat{y}_t = g(W^{h\hat{y}}h_t + b^{\hat{y}}) \quad (2.49)$$

for all time steps $t = 1, \dots, T$, where the matrices W^{**} denote the weights connecting two layers, b^* are bias terms, and f and g are activation functions.

The effects of the introduction of feedback connection in the RNN that provides the context information are profound. In principle, the information from previous time steps can circulate indefinitely inside the RNN where hidden layers also act as a memory. This creates an internal state presented by h_t vectors for each hidden layer, which is the result of the inputs history up to time step t . Through the h_t vectors of the hidden layers, the output

of the network $\hat{y}_t$ at time step t becomes a function of all received inputs $\{x_1, x_2, \dots, x_t\}$. RNNs are therefore very well suited to analyse sequential inputs.

In terms of internal memory, RNNs are far more expressive than FNNs. As stated in Section 2.5.1, FNN can approximate any non-linear function on a compact domain to arbitrary precision. The equivalent result to MLP's approximation theorem is that a RNN with a sufficient amount of hidden units can approximate any dynamical system. According to Siegelmann & Sontag (1995) and Hammer (2000), RNN is capable of computationally as expressive as a Turing machine.

2.5.3 Deep Learning

Deep learning can be described as a sub field of machine learning that studies on deep neural networks (DNNs). DNN are neural networks that capable of learning complex and hierarchical representations from raw data. This capability distinguishes them from the hand crafted models made through an essential features engineering step. A neural network is called a deep neural network (DNN) when it has several stacked hidden layers, where the depth increases with the number of hidden layers. DNN has achieved significant performance especially in the areas of image classification (Krizhevsky et al., 2012), visual recognition (Graves et al., 2013) and natural language processing (Yoon Kim, 2014).

In classic machine learning, the typical approach to solve a problem such as classifications is to design discriminative features and then feed to a general purpose classifier, such as the Support Vector Machine model described briefly in Section 2.3.3. For more complex tasks such as computer vision, automatic speech recognition (ASR) and natural language processing, good features that are sufficiently expressive are very difficult to design. In addition, when the number of features increases, and thus increases the dimensions of the training data, the curse of dimensionality causes the available data to

become sparse. Without using distributed representations, it is not trivial to generalize the regions of the space that have no training instances, often resulting in poor generalization for complex tasks (Bengio et al., 2010; Bengio et al., 2015).

Deep learning has changed the way these problems are approached. The multiple stacked hidden layers in the DNN model provides computations that can be used to automatically discover the increasingly more complex features and allow for their composition. By learning and combining multiple levels of representations, the number of distinguishable regions in a DNN grows almost exponentially with the number of parameters, with the potential to generalize to non-local regions unseen during training (Bengio & LeCun, 2007).

With the success that it has achieved, deep learning approach still has limitations. From theoretical perspective, DNN is not well understood due to its non-convex property. Despite numerous efforts, a proof of convergence to good local minima in the gradient descents has never been found. Thus, most of the research made in this fields are experimentally driven and empirical (Zeiler & Fergus, 2014). From a practical perspective, the need of DNN for large amounts of training samples to derive meaningful context information does not provide them the ability to generalize when trained on small and medium datasets. Big Data is therefore one area that DNN is expected to see potential results. Examples of using DNN in the areas of forecasting using Big Data on deep learning has been demonstrated by Lv et al. (2015) and Zhang et al. (2016).

2.6 Optimizers of Neural Network

The gradient descent based methods are commonly used to optimize the neural network by minimizing the cost function. Recall from Section 2.5, cost function needs to be minimized in order to reduce the learning error of the neural network. Gradient descent

is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks (Ruder, 2016). In this section, three optimizers are reviewed. These optimizers are Stochastic Gradient Descent (SGD), RMSprop and Adaptive Moment Estimation (Adam).

2.6.1 Stochastic Gradient Descent

One of the gradient descent based methods is the stochastic gradient descent (SGD) (Haykin, 2009). SGD is an optimization algorithm with less intensive memory usage because it does not require the entire training dataset to be loaded into the computer's main memory to perform the calculation, but rather only requires one data point at a time. In addition, SGD also converges faster to the minimum point of the slope over the traditional gradient descent method because each update of the neural network weights can often be made without performing iterating over the entire dataset. This is especially beneficial when training over dataset with high data redundancy. Recall from Equations (2.29) and (2.30), $\Delta w_{ji}(n)$ denotes the weight update, η denotes the learning rate, and n is the index of training iterations. The learning rate η controls the amount of weight update per iteration.

Choosing the proper learning rate and the schedule to change the value of the learning rate as learning progresses are one of the challenges in using SGD. A standard method is to use a small constant learning rate that gives stable convergence in the first few training epochs and subsequently halve the value of the learning rate as convergence begins to slow down. The next approach is to evaluate the performance of the model on a reserved data set after each training epoch and reduces the learning rate when the changes between epochs are below a certain threshold. Another commonly used schedule to change the learning rate is to reduce the learning rate at each iteration i as $\frac{a}{b+i}$ where a and

b denotes the initial learning rate and the learning rate when the reduction begins, respectively.

With the careful tuning of the learning rate, SGD can at times still lead to slow and poor convergence to the minimal. A technique of introducing momentum to the SGD was introduced by Polyak (1964) to assist the convergence acceleration. The use of momentum can accelerate the gradient descent by accumulating the velocity of the convergence in the direction of steady error reduction across iterations. This technique introduces a new term, m , which denotes the momentum coefficient is added to the weight update in the Equation (2.29), leading to a new equation expressed as

$$\Delta w_{ji}(n+1) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} + m\Delta w_{ji}(n) \quad (2.50)$$

where m is typically within the range of $0.9 \leq m < 1$.

2.6.2 RMSprop

RMSprop is an unpublished, adaptive learning rate method proposed by Tieleman & Hinton (2012) in their Coursera Class, Lecture 6e. RMSprop uses a moving average of the magnitude of the recent gradients to normalize the current gradients. The normalization is performed over the root mean squared gradients (Parascandolo et al., 2016). Let η denotes the learning rate, and n is the index of training iterations, $r(n)$ denotes the running average term to be added to the weight update equation, the RMSprop can be expressed as

$$r(n) = \gamma r(n-1) + (1-\gamma) \left(\frac{\partial E(n)}{\partial w_{ji}(n)} \right)^2 \quad (2.51)$$

$$\Delta w_{ji}(n+1) = \frac{-\eta}{\sqrt{r(n)}} \frac{\partial E(n)}{\partial w_{ji}(n)} \quad (2.52)$$

where γ is the decay term, which is typically set to 0.9, that controls the contribution of new gradients to the running average $r(n)$.

2.6.3 Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation (Adam) (Kingma & Ba, 2014) is another method that computes adaptive learning rates for each parameter. In addition to storing the decaying average of the past squared gradients like RMSprop, Adam also keeps an exponential decaying average of past gradients, which is similar to the concept of momentum in SGD (Ruder, 2016). Let v_n denote the decaying average and m_n the exponential decaying average of past gradients, v_n and m_n can be expressed as

$$m_n = \beta_1 m_{n-1} + (1 - \beta_1) \frac{\partial E(n)}{\partial w_{ji}(n)} \quad (2.53)$$

$$v_n = \beta_2 v_{n-1} + (1 - \beta_2) \left(\frac{\partial E(n)}{\partial w_{ji}(n)} \right)^2 \quad (2.54)$$

As v_n and m_n are initialized as vector of 0's, Kingma & Ba (2014) observed that they are biased towards zero, especially during the initial iterative steps when β_1 and β_2 are close to 1 (Kingma & Ba., 2014). The authors therefore compute the bias correction for the first and second moment as

$$\hat{m}_n = \frac{m_n}{1 - \beta_1^n} \quad (2.55)$$

$$\hat{v}_n = \frac{v_n}{1 - \beta_2^n} \quad (2.56)$$

Let ϵ denotes the error due to the bias, these bias corrections are then used to update the parameters, similar to RMSprop, which yields the Adam update rule as

$$\Delta w_{ji}(n+1) = w_{ji}(n) - \frac{\eta}{\sqrt{\hat{v}_n} + \epsilon} \hat{m}_n \quad (2.57)$$

The authors also proposed the default values of 0.9 for β_1 , 0.999 for β_2 and 10^{-8} for ϵ for stable performance of Adam (Ruder, 2016).

2.7 Regularization in Neural Network

With the cost function optimizers introduced in the previous section to correct the learning error during training phase of the neural network, in this section, regularization is introduced as a technique to handle overfitting during training phase.

According to Srivastava et al. (2014), overfitting is a serious problem that often occurs in neural network during the training phase. Although neural network can be used to discover very complicated relationships between the inputs and outputs, when limited training data is provided, many of these complicated relationships are actually caused by sampling noise. This sampling noise will only exists in the training set but in in real test data even if it is drawn from the same distribution (Srivastava et al., 2014). Srivastava et al. (2014) therefore introduced regularization using dropouts to overcome this problem and have proven its effectiveness. This method works by dropping out units temporarily from the network, along with all its incoming and outgoing connection, depicted in Figure 2.7. The unit selection for dropping out is random.

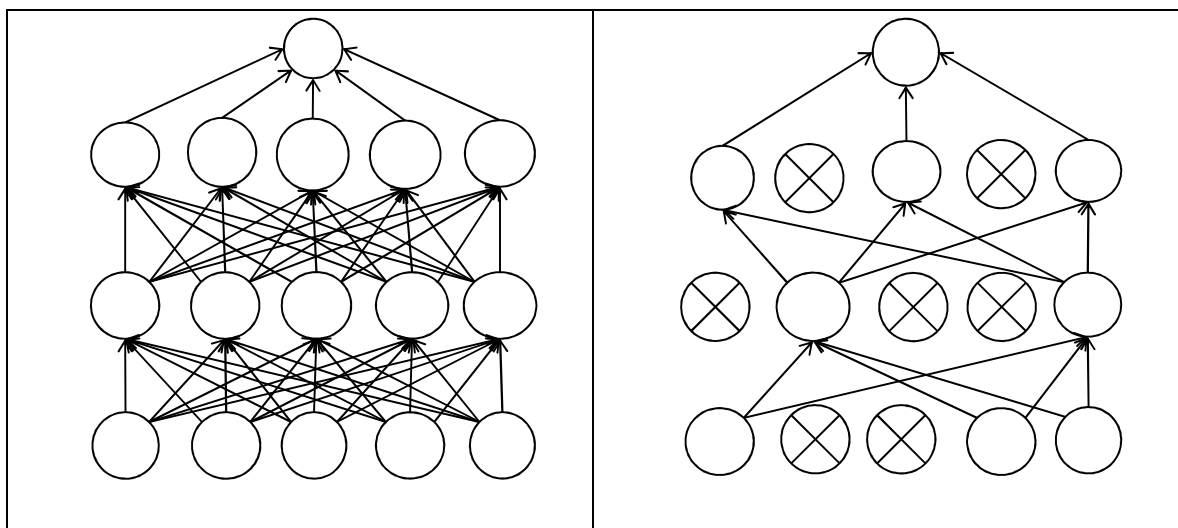


Figure 2.7: Dropout with random unit selection

Dropout's technique applied during training phase differs from testing phase. During training phase, each neuron is presented with probability p and it is connected to

units in the next layer with weights w . During testing phase, each neuron is always present, and its weighted connections to the units in the next layer are present with probability. Figure 2.8 and Figure 2.9 depicts the presents of the neuron with its connection to the units in the next layer in training and testing phase, respectively.

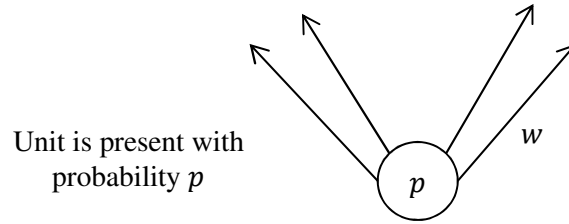


Figure 2.8: Dropout at training phase

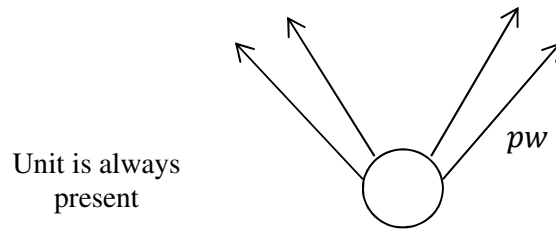


Figure 2.9: Dropout at testing phase

In the simplest case, each unit can be set independently with a fixed probability p for retention. According to Srivastava et al., $p = 0.5$ seems to be close to optimal for a wide range of networks and tasks, but for input units, the optimal probability of retention is usually closer to 1 than to 0.5.

With the combination of appropriate optimizers and regularization during training phase of the neural network, the learning capability of the neural network can be optimized by reducing learning error due to sampling noise. In the next section, various evaluation metrics to evaluate performance of the trained neural network model will be introduced. These evaluation metrics allows the evaluation of the performance of a trained model by measuring the accuracy of the forecasted results.

2.8 Evaluation Metrics to Measure the Forecasting Results

For each identified model, the forecasted results will be used to evaluate their forecasting performances and accuracy. The commonly used evaluation metrics to measure model's performance on forecasting accuracy (i.e. the amount of error between the actual value and the forecasted value) are mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) and mean relative error (MRE). These evaluation metrics are expressed in the equations below, respectively.

$$MSE = \frac{1}{n} \sum_{j=1}^n (P_j - \tilde{P}_j)^2 \quad (2.58)$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_j - \tilde{P}_j)^2} \quad (2.59)$$

$$MAE = \frac{1}{n} \sum_{j=1}^n |P_j - \tilde{P}_j| \quad (2.60)$$

$$MRE = \frac{1}{n} \sum_{j=1}^n (P_j - \tilde{P}_j) \quad (2.61)$$

where n is the total number of values to be compared, and j is the index subscript notation, and P_j and $\tilde{P}_j$ are the actual and forecasted values, respectively.

To evaluate the model's trend accuracy, the common approach is to compare the trend of the predicted value with the trend of the actual value. If the predicted point is higher than its previous point, the trend is consider upward, otherwise it is consider downward. If the predicted point moves in the same trend as the actual point, then the prediction for that point is consider as accurate, otherwise it is considered inaccurate. The evaluation is based on score accumulation, which awards one score for accurate prediction and zero score for inaccurate prediction. The score awarding starts from the second point

since there is no trend created yet at the first point. The total scores obtained are then averaged and convert to percentage to represent the model's prediction accuracy.

2.9 Critical Summary

In this chapter, various statistical model and machine learning models were introduced. Various types of neural network were then discussed. Neural network is a type of non-parametric model which is driven by data. Non-parametric model are inherently susceptible to overfitting. Therefore, regularization techniques were introduced by Srivastava et al. (2014) to address this problem. Gradient descent which is one of the most popular algorithms to perform optimization and by far the most common way to optimize the cost function in neural network was discussed.

From the discussion of the types of neural network, namely feedforward neural network, recurrent neural network and deep neural network, recurrent neural network shows the most promising capability to model sequential inputs, which time series is a type of, without requiring large dataset. Its internal memory structure that creates a form of context information to remember information from previous time steps distinguishes itself from feedforward neural network as a better model to use in time series related problem. Specifically, Long Short-Term Memory (LSTM), which is a type of recurrent neural network, has shown great performance in solving time series forecasting and sequence pattern recognition problem in recent years. LSTM has shown great performance in the area of traffic arrival forecasting. The main reason for the success of LSTM in traffic arrival forecasting is its ability to remember data for longer time span compared to other ANN. LSTM is also capable of predicting time series when there are time lags of unknown size and bound between important events because it is less sensitivity to gap between time steps.

Among other machine learning models researched in the area of traffic arrival forecasting is the stacked auto-encoders (Lv et al., 2015), which is deep learning based neural network, and deep-belief network (DBN) (Huang et al., 2014). Fundamentally, deep learning emphasizes the use of several layers stacked on top of each other to create deep architectures. The applications of deep learning in traffic forecasting are currently still lesser compare to LSTM due to its requirement for larger amount of dataset and complexity in training the model.

With the outstanding ability of LSTM in the area of time series forecasting compare to other neural network model, this work intends to forecast the incoming WIP to a tool group of interest. As presented in Section 2.1, the WIP arrival forecast is time series problem. Hence, LSTM is a suitable model to be used in this work. Therefore, in the next chapter, we present the incoming WIP forecast framework that uses the LSTM as the forecasting model. The methodology required to create the forecasting model will be presented in detail.

CHAPTER 3

METHODOLOGY

In the previous chapter, the architectural concepts of the general types of neural networks have been laid down. From the discussion in previous chapter, the RNN neural network was reviewed to be the most suitable for time series forecasting which exhibit the nature of sequential inputs format. Besides its capability to handling time steps information with the presence of context information as compared to MLP, it requires lesser data as compared to DNN. RNN therefore is the suitable type of neural network for this research. Equipped with this understanding, we proceed to discuss the proposed methodology of this research to create the forecasting model based on RNN model in this chapter. In this chapter, the motivation of this research is presented first, followed by the presentation of the selected RNN model which is used in the proposed framework to create the forecasting model. The proposed framework is presented next, and each of the elements in the framework will be discussed in further detail in the sections that follows. Lastly, a summary section is presented to conclude this chapter.

3.1 Motivation

One of the ways to prevent long cycle time (CT) during preventive maintenance (PM) activities is to perform the PM when the tool group is expected to have low incoming Work-In-Progress (WIP). The WIP arrival expectation is the result from the WIP arrival forecasting commonly done by manufacturing production control engineers.

The existing solution in the Fab² uses a basic statistical forecasting approach to forecast the WIP arrival for all tool groups for the next 7 days. The forecast is run once a

² The problem domain is based on X-Fab Sarawak Sdn. Bhd., which has been abbreviated as the Fab in Section 1.4

week, at the beginning of each week. The calculation steps of statistical forecasting approach are briefly described in Table 3.1.

Table 3.1: Current statistical WIP forecasting steps

Step	Description
1	Given the process flow name of product P is F , retrieve all process steps for F
2	For each process step s in F , get the Turn-Around-Time of s , TAT_s . This value is pre-defined in the MES
3	Let L_F denotes the total photolithography layers in F , $DPML_P$ denotes the Day-Per-Mask-Layer (DPML) committed for P , TAT_F denotes the total Turn-Around-Time (TAT) for F , the run rate for F , RR_F is $RR_F = \frac{L_F \times DPML_P}{TAT_F}$
4	CT for a process step s , CT_s therefore $CT_s = RR_F \times TAT_s$
5	For each lot, sum the next n steps of s until the CT reaches 24 hours. The last s would be the forecasted destination step of the lot after 24 hours from current hour
6	To forecast the destination step of the lot for the next D day, sum the next n steps of s until the CT reaches $D \times 24$ hours. The last s would be the forecasted destination step of the lot for the next D day

The existing forecasting method only caters to products with the number of wafers orders dominating the total WIP in the production line. This is because the calculation requires the number of operation steps and their respective TAT to calculate the forecasted arrival steps. The forecasted results are therefore not accurate because the number of wafers considered in the calculation differs from the actual number of wafers in the production line. In addition, the method could not predict the trend of the incoming WIP to a particular tool group accurately because it does not include any algorithms to capture the time-dependency relation between the data. This limits the ability of the fab managers³

³ Fab managers refers to a group of personnel assigned the management responsibility to oversee various aspect of the fab to ensure that the fab's production line performs smoothly

to create a better PM activities schedule that could minimize negative impact to the production line. This problem therefore provides a research opportunity to create a forecasting model with better accuracy to assist fab managers to carry out more effective PM activities planning that can minimize the impact on CT.

The daily incoming WIP to a tool group can be modeled as a form of time series data by indexing and arranging the observed incoming WIP according to the sequence of their observation time. This is because the daily incoming WIP is a sequence of values observed sequentially over a period time. Recall from Section 2.1, the incoming WIP forecast to a tool group is similar to traffic arrival forecast, where the objective is to ensure that there is enough capacity for the traffic to flow through with minimum obstruction for a given time-frame in the future in order not to create any bottlenecks in the traffic flow. The amount of WIP arriving to a tool group is analogical to the amount of vehicles arriving to a road junction or a group of inter-links of interest in vehicular traffic modeling. This research also requires multi-step-ahead forecasting approach as the research problem requires forecasting the incoming WIP multiple days ahead from the last observation in order to plan for PM activities.

3.2 Proposed Framework

The proposed framework consists of five stages as depicted in Figure 3.1. The first stage is the data acquisition of the incoming WIP as the input for the forecasting model. The second stage is the feature engineering of the incoming WIP data to the format and scales required to train the neural network. The third stage is the parameter size identification in the neural network, and lastly, the evaluation of the forecasting performance of the model with the selected parameters sizes.

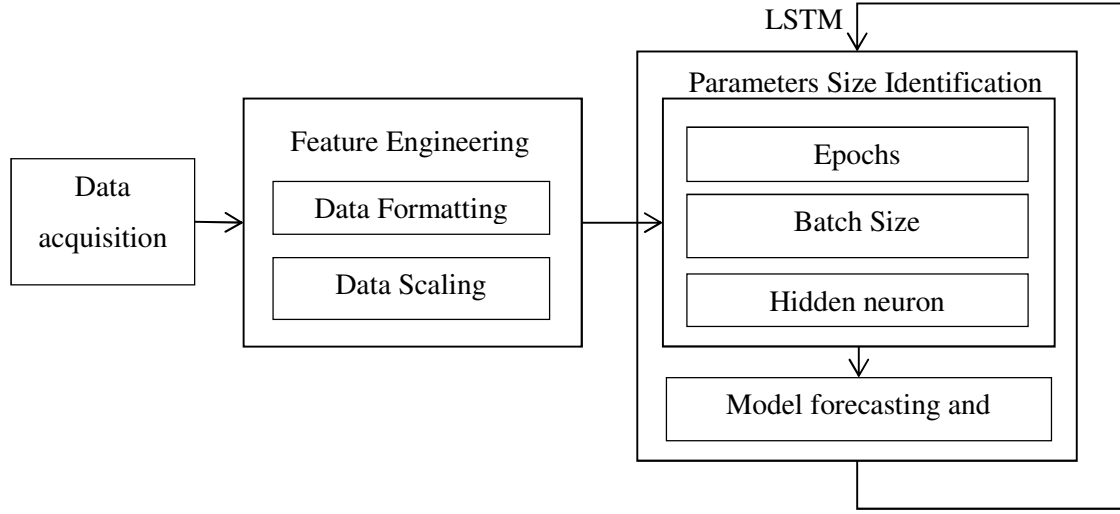


Figure 3.1: Proposed framework

In data acquisition step, the process will extract the necessary information from the data store to form the dataset. The detail of the dataset formation will be discussed in Section 3.5. Next, the dataset is sent to the feature engineering step to prepare the data into structures and values required to train and test the neural network. The feature engineering steps consist of data formatting, data scaling and lag size variations. Every individual step in feature engineering will be discussed in Section 3.6.

Once the features have been engineered, the parameters size of neural network can then be tested to identify the best size for each of the parameters. The proposed method needs to identify the size for three parameters which are the epochs, the batch size and the hidden neuron. Note that the parameter size affects the forecasting performance of the neural network as it determines the capability of the neural network to model the given set of time series data. The description of the identifications steps for these parameters will be discussed in Section 3.7. Finally, the neural network is required to forecast seven days ahead from the last observed incoming WIP, and its forecasted value will be measured against the actual value to evaluate for its forecasting performance. The details of the

forecasting and evaluation steps will be presented in Section 3.8.

3.3 Data Source

The incoming WIP is the main input data for the forecasting model. Let $IWIP$ denotes the incoming WIP for a particular tool group, $IWIP$ can be calculated as

$$IWIP = (WIP_{t24} - WIP_{t1}) + \sum_{t=1}^{n=24} (MOVE) \quad (3.1)$$

where WIP is the actual WIP recorded at hour t , $MOVE$ denotes the total number of wafers completed at the tool group for the past 24 hour and $IWIP_{t24}$ denotes the $IWIP$ at the 24th hour. $IWIP$ is the WIP difference between the start (i.e., first hour) of a manufacturing day and the last hour (i.e., 24th hour) of the same manufacturing day. A wafer is considered as completed at a particular tool group when it is moved to the next tool group. The information required to calculate for the incoming WIP of one manufacturing day are therefore the first and the last recorded hourly WIP, and the total wafers moved by that tool group for that manufacturing day. The collection of the incoming WIP over a period of time will form a set of historical data that will be used for the model to train and predict the future values. Each historical incoming WIP is called a lag, and the collection of these historical data is called lags in the time series forecasting domain.

By collecting the $IWIP$ for each manufacturing day for a period of time as historical observation, a sequential observation of lags for the $IWIP$ is created. With the use of the production data, these observed lags of the $IWIP$ would represent the actual WIP condition in the production fab for the observed series of manufacturing days. It therefore forms the fundamental time series data that can be used to train the neural

network to model the *WIP* condition of the production environment. Such data that is time series in nature would require its sequence to be preserved by following the date of the manufacturing day it represents when it is sent to feature engineering step. In the data acquisition process, the sequence of the *IWIP* will be preserved when the data is being acquired from the production environment.

Due to the fact that a wafer is a single entity with the smallest quantity value of one, the value of *MOVE* and *WIP* will always be the whole number. However, for *IWIP*, it will be integer number instead of whole number, as it is possible for *IWIP* to be negative value. *IWIP* can be negative values because in the event that the *WIP* supply to tool group *A* experiences interruption, such as the tool group that supplies *WIP* to tool group *A* is experiencing tool errors that caused low *WIP* throughput, the subsequent *WIP* supplying to tool group *A* will be low. Low *WIP* supply has direct impact on the total *MOVE* that tool group *A* can generate as well. Hence, it is possible for *IWIP* to have negative values.

3.4 Data Acquisition

The data sources required to calculate *IWIP*, which are *WIP* and *MOVE* discussed in Section 3.3, needs to be acquired from the production environment of the semiconductor manufacturing plant. In every manufacturing plant, there is a computerized system called the Manufacturing Execution System (MES) that defines the workflow of the manufacturing process. This database-driven system is the core system of a manufacturing plant as it serves as the primary platform to setup, monitor and execute the manufacturing processes.

With such criticality to the production environment, it is of utmost important to maintain the stability and performance of the MES at all the time. One of the safeguards

taken by the managing department of the MES is to prevent direct database access from custom written programs to prevent any unintentional logical error that could jeopardize the stability and the performance of the MES. For example, database tables locking and inefficient query methods when retrieving information. In order to serve the needs of custom written programs, it is a standard practise of the managing department to retrieve and store the data in the MES to a secondary production database environment through technologies such as database replications. The custom written programs will then use the secondary production database environment as the primary data source for further analysis and decision making. Figure 3.2 describes this setup.

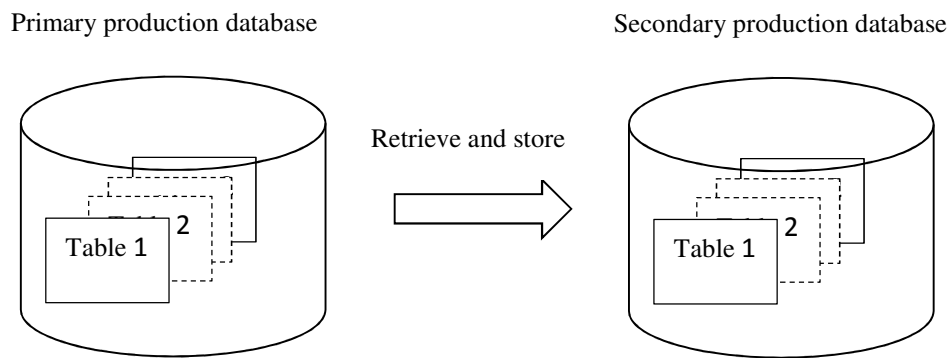


Figure 3.2: Storing production data to secondary database

Following the standard practice of accessing the production environment's data, the database retrieval mechanism of our custom programs will therefore query against the secondary production database to acquire the required data and store them in a local database on a virtual machine setup for the purpose of this research. To get the data required to calculate for *WIP*, our custom program will be scheduled to retrieve the data every day to get the actual *WIP* from the production floor and save to the local database. The custom program will filter the data by tool group in order to get the actual *WIP* of the tool group to be forecasted. To get the data to calculate for *MOVE*, our custom program

will be scheduled to run every day to retrieve the total wafers completed for the past 24 hours and store in the local database and then, further sum the total wafers for the tool group to be forecasted. The *IWIP* calculation for the tool group to be forecasted is then calculated after both *WIP* and *MOVE* data is ready. The calculated *IWIP* will be stored in one of the tables in the local database to form the historical data, which will be used in the next step of the framework to create the necessary feature and as the input to the neural network. Figure 3.3 describes this setup.

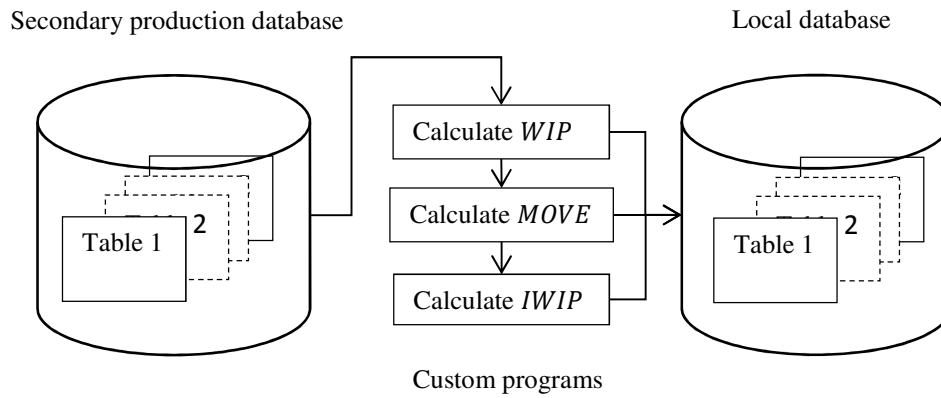


Figure 3.3: Storing processed data into local database for forecasting

As describe in Section 3.3, the sequence of the *IWIP* data needs to be preserved before furthering its use for feature engineering. In order to do so, the data is stored in a single column format beginning from the earliest lag to the latest according to date of its manufacturing day. This arrangement preserves the sequence of the *IWIP* as a dataset. Storing the observed lags in a column structure is necessary as the feature engineering step will need this structure to create the data properly before feeding into the neural network.

Various training datasets for the neural network can be generated once and stored in the tables of local database for permanent use. Having these data stored in the tables of the local database eases their retrieval for the next step in the framework, which is feature engineering. In addition, the storing of the calculated *IWIP* in a centralized location and

isolated from both primary and secondary production database environment allows us to create a forecasting program that the unintended logical error will not affect the production environment.

3.5 Feature Engineering

Multi-step-ahead time series forecasting consist of predicting several future observations of a given sequence of historical observations (Taieb, 2014). In time series research, there is no concept of input and output features. Instead, we determine the part of the time series data that will represent the output to be predicted and use feature engineering to construct all of the inputs that will be used to train the forecasting model to make predictions for the future time steps. These feature engineering processes involves data formatting and data scaling. In this section, we first discuss the data formatting of the time series data of the incoming WIP in Section 3.5.1, followed by the data scaling step in Section 3.5.2.

3.5.1 Data Formatting

Data formatting is important because in time series forecasting, neural network model requires supervised-learning type of training method, where the neural network is trained with the actual values as its guidance, to create the suitable internal states that can model the time series.

In time series, the term “lags” is commonly used to denote values observed at prior time steps to the prediction. Generally, the time series data are separated into training and testing set where the training set contains the lags, while the testing set contains the actual values of future time steps. The data formatting required by neural network for both training and testing dataset are represented in Table 3.2.

Table 3.2: Data formatting for neural network

Set	Training dataset	Testing dataset
1	$(x_1, x_2, x_3, x_4, \dots, x_n)$	$(x_{n+1}, x_{n+2}, x_{n+3}, x_{n+4}, x_{n+5}, x_{n+6}, x_{n+7})$
2	$(x_2, x_3, x_4, x_5, \dots, x_{n+1})$	$(x_{n+2}, x_{n+3}, x_{n+4}, x_{n+5}, x_{n+6}, x_{n+7}, x_{n+8})$
$\vdots$	$\vdots$	$\vdots$
$N - (n + f) + 1$	$(x_{N-n}, x_{(N-n)+1}, x_{(N-n)+2}, x_{(N-n)+3}, \dots, x_N)$	$(x_{N+1}, x_{N+2}, x_{N+3}, x_{N+4}, x_{N+5}, x_{N+6}, x_{N+7})$

Through data formatting, the time series data is separated into smaller training sets, where each subsequence training and testing set shifts a day forward while maintaining the dataset size. Consider N as the total time series data size, x denotes individual lag in the dataset, n is the size for each training set, and f is the size of the testing set which would be the number of days the neural network is required to forecast. In this research, since the number of days required to forecast is seven, therefore the testing set has a size of seven days. For example, assume that $N = 20$, $n = 7$ and $f = 7$, a total of $(20 - 14) + 1 = 7$ sets of training dataset and testing pairs to train the neural network. Table 3.2 can be written as Table 3.3.

Table 3.3: Data formatting for neural network using $N = 20$, $n = 7$, $f = 7$

Set	Training dataset	Testing dataset
1	$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	$(x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14})$
2	$(x_2, x_3, x_4, x_5, x_6, x_7, x_8)$	$(x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15})$
3	$(x_3, x_4, x_5, x_6, x_7, x_8, x_9)$	$(x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$
4	$(x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$	$(x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17})$
5	$(x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11})$	$(x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18})$
6	$(x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$	$(x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19})$
7	$(x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13})$	$(x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20})$

With this data format, it signifies the model that the *IWIP* of the subsequent f days ahead are affected by the immediate previous n days of *IWIP*. Note that the formatted data set can be randomized so that the lag sequence is only maintained within the each pair of training and testing dataset, while the lag sequences between each subsequent pairs of

training and testing dataset are random.

With the formatted data given in this research, set 1 to $(N - (n + f) + 1) - 3$ will be used in Section 3.7 while the remaining three sets from $(N - (n + f) + 1) - 3$ to $(N - (n + f) + 1)$ will be used in Section 3.8.

3.5.2 Data Scaling

In order for this dataset to be usable for the neural network model, the training dataset needs to be scaled to the range of the activation function of the neuron so that the activation value can be calculated accurately to decide if the neuron should be activated. In this research, the activation function that will be used is the hyperbolic tangent activation function. By using hyperbolic tangent activation function, the training dataset will therefore be scaled to the range of 0 to 1. The data scaling is accomplished using Equation (3.2).

$$y = \frac{x - \min}{\max - \min} \quad (3.2)$$

where x is a lag point in the time series, $\min$ is the smallest lag value in the time series and $\max$ is the largest lag value in the time series.

Besides being the default activation function of the LSTM, hyperbolic tangent is used in this research because it matches characteristic of the *IWIP* that is integer number in nature, as explained in Section 3.4. Therefore, hyperbolic tangent function is suitable to be used as the activation function in this research.

3.6 Long Short-Term Memory Model

Recall from Section 2.5.2, recurrent neural network (RNN) is a type of neural network that is well suited to analyse sequential data. This makes RNN suitable for this work because incoming WIP data is time series data. In this work, a type of RNN called

the Long Short-Term Memory (LSTM) will be used to develop the forecasting model.

LSTM is a type of RNN that can be used to learn time-dependent relation and discover long term dependencies in sequential data more accurately than other RNNs and DNN (Sak, 2014). The reason for the increased accuracy in LSTM over other RNNs is because the exploding gradient and vanishing gradient phenomena inherently experienced in RNN was addressed in LSTM. Although the RNN prevails in modelling sequential based data compared to MLP, prior to the development of LSTM, the presence of these two phenomena cause previous variants of RNN to suffer in the inability to record information for longer period of time, and therefore, unable to capture long term dependencies in the training examples.

In order to understand the exploding and vanishing gradient and how it hinders RNN from recording information over longer period, an understanding of the training algorithm used to train RNN is first required. The training algorithm commonly used to train a RNN is the back-propagation-through-time (BPTT). This algorithm is an extension of the standard back-propagation algorithm introduced earlier in Chapter 2. The intuitive of BPTT is to unfold the RNN over time and then apply the back-propagation to the network for training as if the network is a MLP. Let x be the input layer of RNN, s be the hidden layer of RNN, y be the output layer of RNN, t be the time subscript, V be the weight matrix from input layer to hidden layer, U be the weight matrix from a previous hidden layer to the next hidden layer, and W be the weight matrix from hidden layer to output layer. Figure 3.4 depicts an unfolded RNN over time steps, $T = 3$.

In the unfolded RNN, the recurrent weight is duplicated spatially for an arbitrary number of time steps, T . Therefore, for each neuron that sends activation along a recurrent connection has at least T number of copies. In accordance to Equation (2.50), the errors are propagated backward as

$$\delta_j(t-1) = \sum_h^m \delta_h(t) u_{hj} \varphi_j' (s_j(t-1)) \quad (3.3)$$

where j is the index subscript for hidden layer, h is the index subscript for the previous hidden layer, m is the total hidden layer, t is the time subscript, δ is the local gradient required to change the synaptic weights, m is the total hidden layer, u is the linear combination of the input signals to the neuron, φ' is the differentiated activation function and s denotes hidden layer. This allows the calculation of the error assessed at time t for the neuron's output calculated on the basis of an arbitrary number of previous time steps.

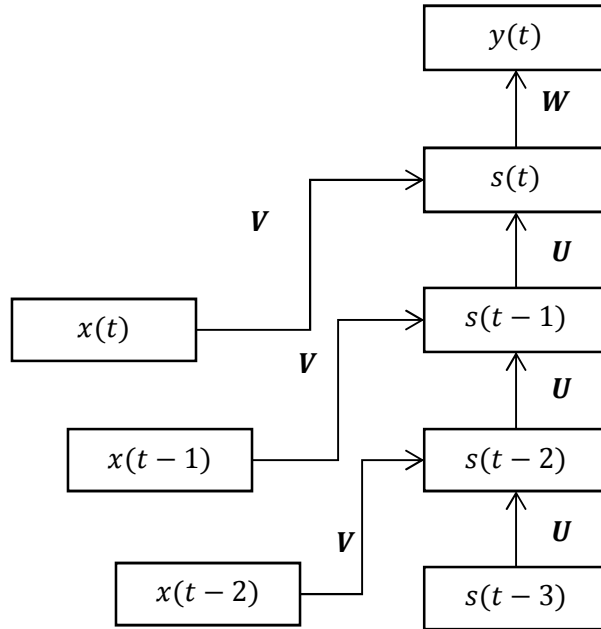


Figure 3.4: An unfolded RNN over time steps

It is obvious that the larger T we choose, the greater the requirement of the memory would be since both past errors and activations need to be stored. However, in practise, a large T is often useless due to the vanishing or exploding gradients problem which was formally identified by Hochreiter in 1991 (Hochreiter, 1991). The cause of this problem is due to the reason that for most activation functions like sigmoid and hyperbolic tangent that have the gradient in the range of $(-1, 1)$, when chain rule in the backpropagation

algorithm multiplies the gradients to compute updates of weights, the weights updates either decreases rapidly or grows out of bound as it propagates along the layers.

The solution to this problem is the introduction of forget gate into the neural network to avoid long-term dependencies (Hochreiter & Schmidhuber, 1997). The forget gate is used during the training phase to decide when information from previous cell state should be forgotten. In general, LSTM has three gates, namely, the input gate, the forget gate, and the output gate. Figure 3.5 depicts the LSTM cell.

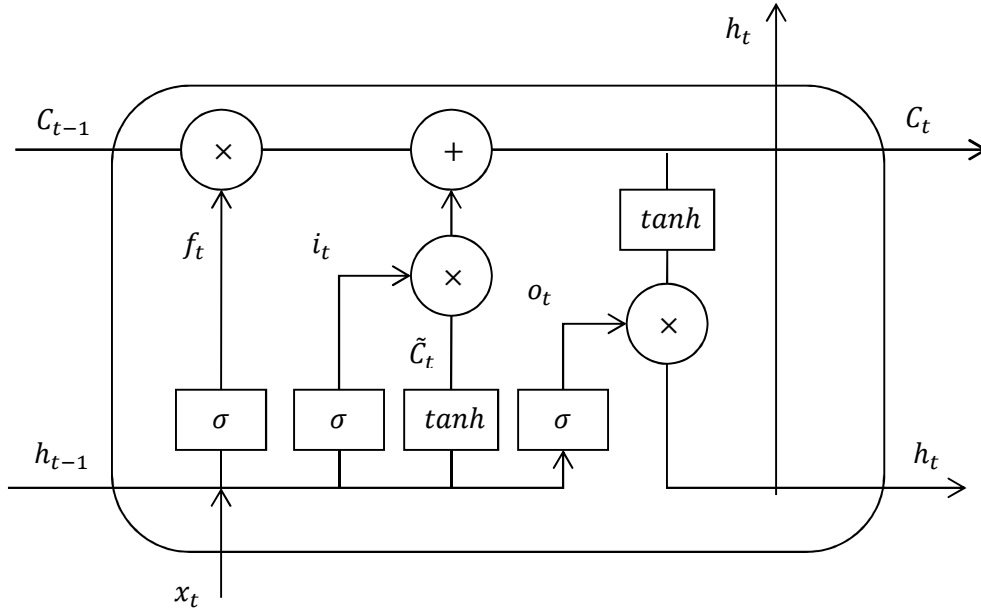


Figure 3.5: Long Short-Term Memory's cell

The key feature of LSTM is its gated memory cell and each cell has the above mentioned three gates. These gates are used to control the flow of information through each cell. Let time be denoted as t . At time t , the input to a LSTM cell is x_t , and its previous output is h_{t-1} . The cell input state is $\tilde{C}_t$, the cell output state is C_t and its previous state is C_{t-1} . Input gate at time t is i_t , forget gate is f_t and output gate is o_t . According to the structure of the LSTM cell, C_t and h_t will be transmitted to the next cell in the network. To calculate C_t and h_t , we first solve the following 4 equations:

Input gate:

$$i_t = \sigma(W_i x_t + W_i h_{t-1} + b_i) \quad (3.4)$$

Forget gate:

$$f_t = \sigma(W_f x_t + W_f h_{t-1} + b_f) \quad (3.5)$$

Output gate:

$$o_t = \sigma(W_o x_t + W_o h_{t-1} + b_o) \quad (3.6)$$

Cell Input:

$$\tilde{C}_t = \tanh(W_c x_t + W_c h_{t-1} + b_c) \quad (3.7)$$

where:

W = weighted matrices

b = bias vector

σ = sigmoid function

$\tanh$ = hyperbolic tangent function

Sigmoid function, σ , is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (3.8)$$

Hyperbolic tangent function, $\tanh$, is defined as:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \quad (3.9)$$

Following Equation (3.9), we calculate the cell output state using the following equation:

$$C_t = (f_t * C_{t-1}) + (i_t * \tilde{C}_t) \quad (3.10)$$

Lastly, the hidden layer output is calculated using the following equation:

$$h_t = o_t * \tanh(C_t) \quad (3.11)$$

The memory cell in LSTM is the main component that allows LSTM to remember the long term dependencies relation in a sequential data such as the time series data and thus, it has

better performance in solving problems related to forecasting compare to other RNNs.

In LSTM, the restriction in training batch data preparation experienced in MLP and DNN has also been removed, and a new feature that allows time element to be included into the training batch data was introduced. The MLP and its variants in DNN have a restriction for the training batch data preparation where the mapping function of the input and the output is fixed and static (Sutskever et al, 2014). This would translate to the need of the dimensionality of the inputs and outputs to be the same. In our research context, the output will have two dimensions, denoted as X and Y , where X represents the number of rows of the output, while Y represents the single set of seven *IWIP* values forecasted. Therefore, the maximum dimension allowed for the inputs to the MLP is restricted to two as well.

The LSTM is less restrictive compare to MLP because the input dimension of LSTM does not need to be the same as the output dimension. In addition, a new dimension that represents time sequence is introduced in LSTM. The significance of this new dimension is that, instead of mapping inputs to outputs alone, the LSTM is capable of learning a mapping function for the inputs over time to an output with the use of the sequence dimension as time step notation. According to Felix et al. (2001), the introduction of sequence as a dimension in LSTM unlocks the neural network's capability to solve many time series tasks unsolvable by feed-forward network such as the MLP. Using the example from Table 3.3, Table 3.4 depicts this inclusion of time steps as dimension Y of the training and testing dataset. Following this format, dimension X consists of a series of lags, dimension Y consists of the number of time steps to be forecasted, and dimension Z consists of the number of features used in the forecast. The flexibility of this approach allows the time step to be factored into the training dataset which are the inputs to the LSTM, creating three dimensional training dataset, while

allowing the testing dataset to maintain in two dimensional data structure, which is the output of the LSTM for the future steps forecasting.

Table 3.4: Dimension of the data format in LSTM

Set	Training dataset			Testing dataset	
	X	Y	Z	X	Y
1	$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	7	1	$(x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14})$	1
2	$(x_2, x_3, x_4, x_5, x_6, x_7, x_8)$	7	1	$(x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15})$	1
3	$(x_3, x_4, x_5, x_6, x_7, x_8, x_9)$	7	1	$(x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$	1
4	$(x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$	7	1	$(x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17})$	1
5	$(x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11})$	7	1	$(x_{12}, x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18})$	1
6	$(x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$	7	1	$(x_{13}, x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19})$	1
7	$(x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13})$	7	1	$(x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20})$	1

In this research, each row in dimension X will contain a set of seven *IWIP* points that corresponds to the number of *IWIP* points to be forecasted. These seven *IWIP* points are grouped into a single set of value. The number of *IWIP* points to be forecasted is represented in dimension Y as time steps to be forecasted. Dimension Z will have the value of one in the training dataset, which correspond to the single set of seven *IWIP* points in dimension X . By putting seven *IWIP* points in both the training and testing dataset, we are effectively telling LSTM that each future seven *IWIP* points are related to its immediate previous seven *IWIP* points. Following the same approach of Felix et al. (2001), the number of *IWIP* points in both training and testing dataset is set to be the same number as the number of points to be forecasted. We then use this data formation to request the LSTM to discover the time-dependent relation by training it with the given *IWIP* datasets. The discovering capability of the LSTM in its supervised-learning is dependent on the size of its three internal parameters, which are the epoch, the batch size and the hidden neuron. In Section 3.7, we will discuss the role of these three parameters and the steps to identify their sizes.

3.7 Parameter Size Identification

In this section, we discuss the steps that will be taken to identify the size of the parameters of the LSTM model. As a non-parametric model, neural network model does not have a fixed structure. In a RNN with one hidden layer, the ability of the neural networks to discover important relationship in the training data during the supervised-learning is affected by the number of hidden neuron, the batch size used per epoch and the number of epoch. The combination of the sizes of these three parameters that results in stable supervised-learning and delivers the lowest forecast error is desired. In this section, the procedures used to identify the sizes of the three parameter sizes and the analysis of the stability of the supervised-learning will be presented.

The arrangement of the sequence of parameters to be examined follows the order of epoch, batch size and hidden neuron. The number of hidden neuron to use should be tested last because it is easy for the model to suffer overfitting/over-learning when the number of hidden neuron increases. Batch size should be tested after epoch as larger batch size allows the number of epoch to be reduced while increasing the prediction accuracies. Each parameter being examined will have a list of pre-defined sizes to be tested.

When one of the parameters is being examined, the remaining two parameters will be fixed to their current sizes in their respective list. This is to control the variation across the examinations. For each combination of the parameters, the model will be tested with that combination to measure its performance in terms of its forecasting error and the stability of its supervised-learning using the data sets allocations specified in Section 3.5.1. The metric used to measure the forecasting error of the model is the root mean-squared-error (RMSE) defined in Equation (2.66). RMSE is a frequently used evaluation metric because it measures the difference between the values predicted by a model and the actually observed values.

For each parameter size combination to be tested, the model will be experimented N number of times with the same parameters setting, and N number of RMSE will be obtained to represent the performance of the model for each experiment. The reason running multiple experiments for each parameter size combination is because internally, neural network uses randomization to the weights and the states of its neurons. Hence, forecasting error differs between experiments. Therefore, multiple experimental runs is recommended to allow for the selection of the neural network model with internal settings to produce the lowest RMSE.

It is common to assume that low RMSE at the end of the supervised-learning directly translate to sufficient capability of the model to perform forecast. However, this assumption is misleading because a model that has over-learned during the supervised-learning can deliver result with very low RMSE at the end of the training. An over-learned model will perform poorly in the actual forecast. Therefore, it is necessary to also measure the stability of the supervised-learning of the model given a particular combination of the three parameters. During each epoch in the supervised-learning, the model will be required to perform two forecasts, one uses a reserved set from the training set, and other uses a reserved set from the testing set. With two forecasts performed, two RMSE will be generated. The RMSE generated by using training set is the training error, while the RMSE generated by using the reserved testing set is the testing error. To measure the stability of the supervised-learning, the RMSE for both training error and testing error of each epoch are collected and plotted in a single graph. With y-axis presenting the RMSE, and x-axis representing the number of epoch, Figure 3.6 – Figure 3.10 show examples of the curves that exhibited from the supervised-learning. The combination of parameter sizes that allows the model to exhibit curve pattern similar to Figure 3.6 is the desired selection.

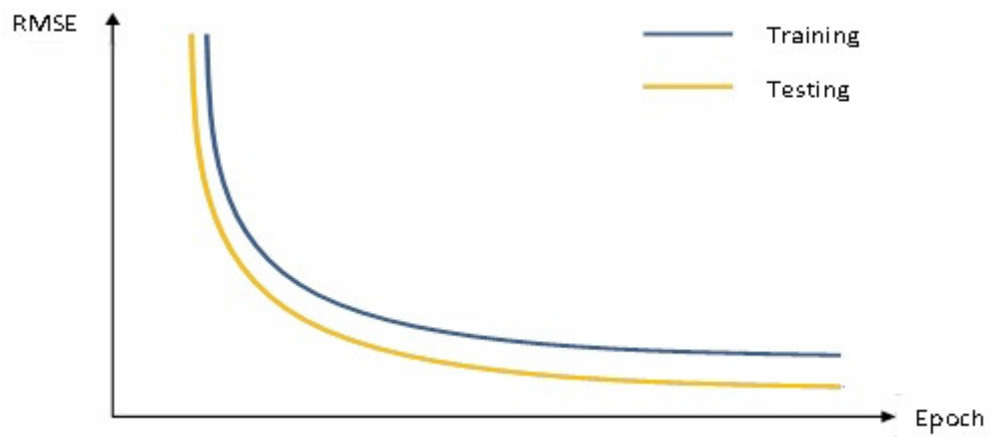


Figure 3.6: RMSE curves when the model is well learned

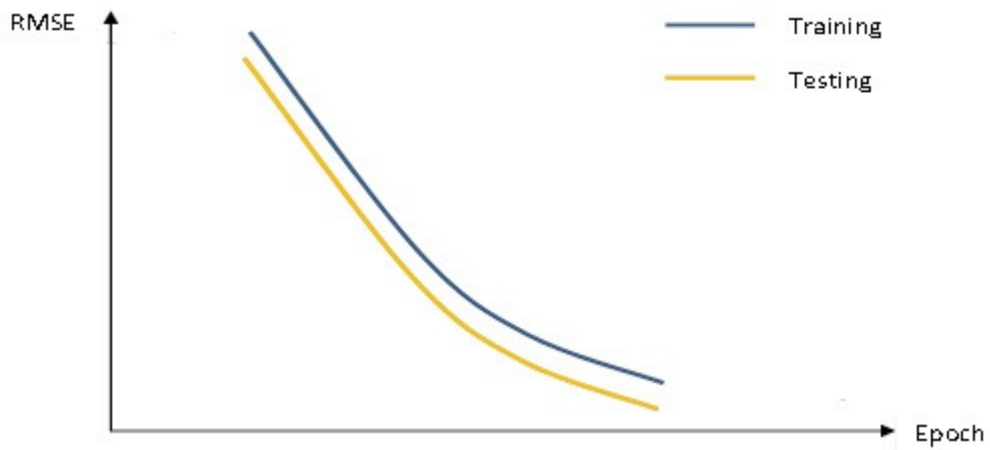


Figure 3.7: RMSE curves when the model is under-learned

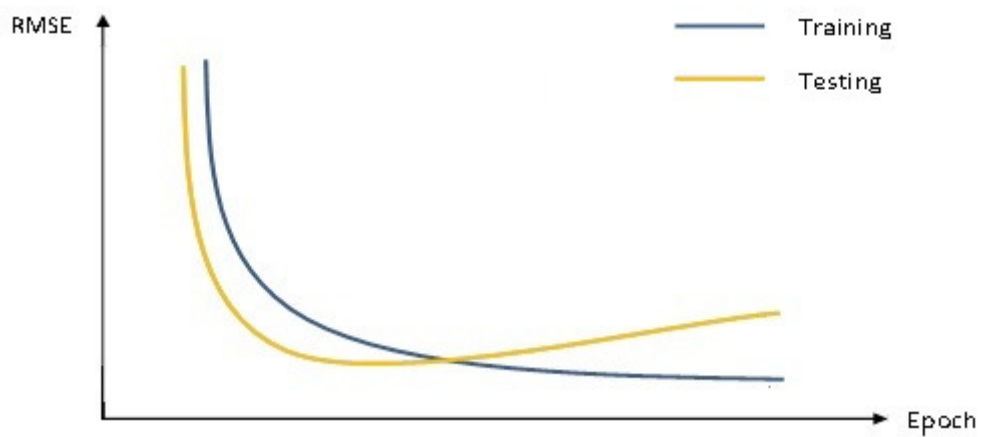


Figure 3.8: RMSE curves when the model is over-learned (overfitting)

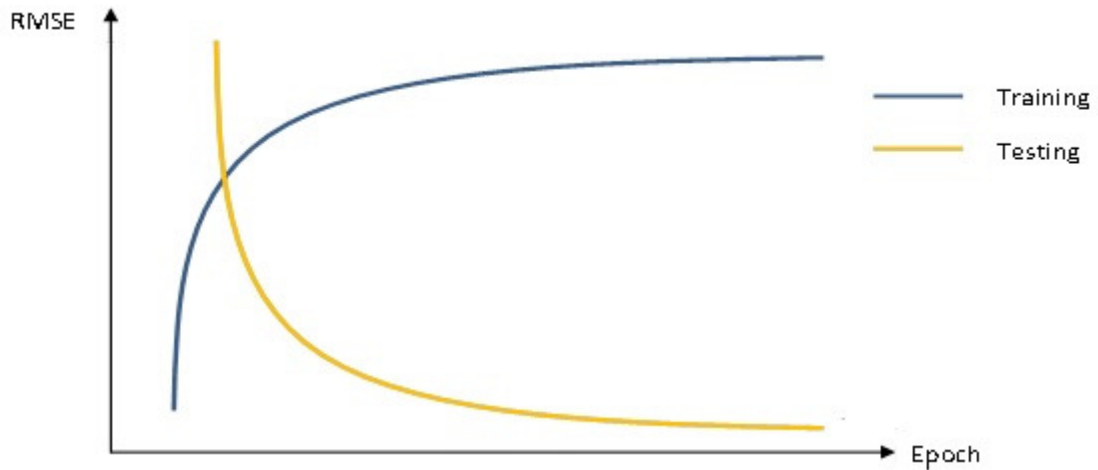


Figure 3.9: RMSE curves when the model experiences learning error

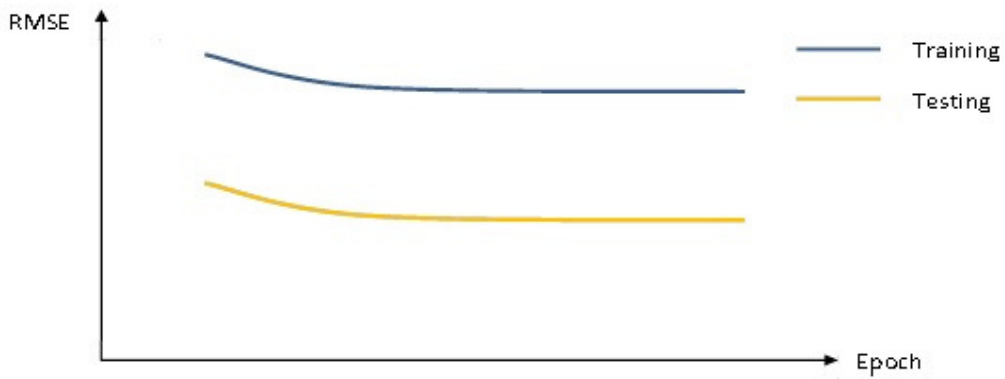


Figure 3.10: RMSE curves when the model failed to learn

The combination of the three parameters' sizes that allows the model to perform both stable supervised-learning and lowest RMSE in its forecast error will be selected to be used in Section 3.8. Should none of the sizes in the array fits the good learning curve, the sizes in the array will be re-adjusted, either by reducing or increasing the value of the sizes.

For each of the experiment conducted on the model, the model and its final adjusted internal weightage are saved separately. With this, there will be two files created to store the configuration information of the neural network for each experiment, and a total of $N \times 2$ files will be created, where N denotes the number of experimental runs. Storing the

configuration information of the neural network ensures reproducibility and this is necessary for the model forecasting and evaluation to be carried out next, which will be discussed in Section 3.8.

3.8 Model Forecasting and Evaluation

After the parameters size are identified, with the neural network's information outputted to files from Section 3.7, we will be able to re-create the exact model and its weights for further evaluation. Namely, we will evaluate the neural network using its outputted configurations that has the lowest RMSE with stable supervised-learning, as these indicates that the prediction of the neural network will have smaller error when measured with the actual value. The forecasting test that will be conducted requires the neural network to forecast seven steps ahead from the last observed lag in the dataset. With the reserved data sets defined in Section 3.5.1, this test can be conducted.

By using the reserved data sets in this section which differ from the data sets used the supervised-learning in Section 3.7, we ensure that the forecasting test to be conducted mimics the real world scenario where the actual values ahead of the last lag observed by the model in its supervised-learning are unknown to the model. This eliminates the possibility that the model returns the time series it has memorized as its forecasting results.

In real world scenario, the actual values ahead of the last observed lag are unknowns during the forecasting. Therefore, in order to mimic the real world scenario, we create a set of dummy values to represent the unknown values ahead of the last observed lag. The actual values are preserved for model evaluation purposes while the dummy values are supplied to the neural network for forecasting. By using the dummy values, it is easier to distinguish the purpose of the seven lag points used in each of the testing dataset in Section 3.7 and this Section 3.8. In Section 3.7, the seven lag points are used to measure

the performance of the neural network model in terms of its forecasting accuracy for the parameters size being tested. In this Section 3.8, the seven dummy values are used to represent the next seven days' *IWIP* that are unknown at the moment of the forecasting is performed. In essence, we are asking the neural network, "What is the forecast for the next seven days, given the last observed historical lags of the past N days presented by the training set?" In addition to the use for forecasting purpose, the dummy values also serve the purpose of filling in the gap of the unknown *IWIP* ahead in the testing dataset so that data formation is intact and follows the structure defined in Section 3.5.1 to allow the neural network to execute the forecasting properly.

The dummy values to fill in the gap of the dataset can be any numerical values. In this work, the average values of the time series will be used. To maintain the consistency of the numerical type of the time series data, the calculated average value will be rounded up to the nearest integer value. The average value will be calculated from the first lag until the last lag prior to the first day of seven days gap. This average value also represents the average *IWIP* for L days of lags. Let x denotes a single lag, L denotes the total lag size of the time series data, and $\bar{x}$ denotes the average *IWIP*, the average *IWIP* is calculated as

$$\bar{x} = \frac{\sum_{i=1}^L x_i}{L} \quad (3.12)$$

For ease of illustration, we use the last three data sets from Table 3.4 as the reserved datasets to illustrate the modification performed on the duplicated copies of the same datasets. Table 3.5 shows the end result of the modification.

Table 3.5: Modified reserved datasets with $\bar{x}$ as dummy value

Set	Training dataset	Testing dataset
5	$(x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11})$	$(\bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x})$
6	$(x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$	$(\bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x})$
7	$(x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13})$	$(\bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x}, \bar{x})$

The $\bar{x}$ denotes the average *IWIP* calculated using Equation (3.12). Each of them has the same value as one another. Each of the testing dataset in Table 3.5 created using repeated $\bar{x}$ serves as the dummy set. The purpose of having the dummy set is to represent the *IWIP* ahead that are unknown during forecast and to maintain the structure of the testing set. The structure of both training and testing dataset must be maintained for the forecasting model to be able to evaluate them. Once the input data for the neural network is ready, the neural network is re-created from neural network information files, and the input data will be fed into the model. After the neural network model has performed the forecasting, the forecasted values will be stored in a dedicated table in the local database for evaluation purposes later. The results can also be used for future comparison purposes when the neural network is tested using different sets of parameter sizes. For the evaluation of the forecasted result, two accuracies measurement will be used, which are value accuracy and trend accuracy. We first explain the value accuracy measurement, followed by trend accuracy measurement.

For value accuracy, the values of the forecasted result are compared with the actual values to measure the residual errors. The evaluation metric which is commonly used to measure the residual error for prediction problem is the Root Mean Squared Error (RMSE) (refers to Section 2.8). In addition to RMSE, confusion matrix will be used to create an error matrix for the predictions. The use of RMSE and confusion matrix allows us to represent the performance of the forecasting model from different perspective. RMSE is a good representation of the forecasting model performance as a whole for the total seven days forecasting while confusion matrix is a good representation of the forecasting model performance for individual day forecasting. Confusion matrix also allows us to define the upper and lower limits of the RMSE for the forecasting model so that the accuracy of the model can be quantified.

To properly assess the accuracy of the forecasted results, accuracy is defined as within the ten percent difference from the actual *IWIP*. Let y denotes the actual *IWIP* for one of the forecast days and $\tilde{y}$ denotes the forecasted *IWIP* for the same day, the accuracy required is therefore expressed as

$$(y - (y \times 0.10)) \leq \tilde{y} \leq (y + (y \times 0.10)) \quad (3.13)$$

For each of the individual day, one point will be assigned if the forecasted *IWIP* meets the defined accuracy, otherwise zero point will be assigned. The total points collected by the model are then summed and converted to percentage as the representation of its accuracy, in addition to its RMSE. Let ACC denotes the accuracy score of an individual day where ACC is either zero or one, F denotes the total days to be forecasted, f denotes the subscript of each individual day within F , the value accuracy of the model calculated in percentage, V_{ACCP} , can be expressed as

$$V_{ACCP} = \frac{\sum_f^F ACC_f}{F} \times 100 \quad (3.14)$$

For trend accuracy, the pattern of the trend of the forecasted seven incoming WIP is compared with the pattern of the actual seven incoming WIP to calculate the number of days with the correct trend. In trend accuracy measurement, each of the incoming WIP is quantify into unit of point, and a set of trend is measured using two intermediate points. The first set of trend is measured from the first point to the second point, while the second trend is measured from the second point to the third point.

To measure the trend accuracy, we first define the type of trend in this work. Recall from Section 2.8, if a point is higher than its previous point, the trend is consider upward, otherwise it is consider downward. A score of one point is assigned to a trend set of the forecast result when the predicted trend is the same as the actual trend, and zero

point if the trend differs. Let ACC denotes the accuracy score of each trend set where ACC is either zero or one, T denotes the total days trend set, t denotes the subscript of each individual day within T , the trend accuracy of the model calculated in percentage, T_{ACCP} , can be expressed as

$$T_{ACCP} = \frac{\sum_t^T ACC_t}{T} \times 100 \quad (3.15)$$

To aid the visualization of the predicted incoming WIP pattern against the actual incoming WIP pattern, a graph plot containing the actual $IWIP$, the forecasted $IWIP$, and the permitted upper and lower accuracy limits will be prepared. Figure 3.11 shows an example of this visual aid.

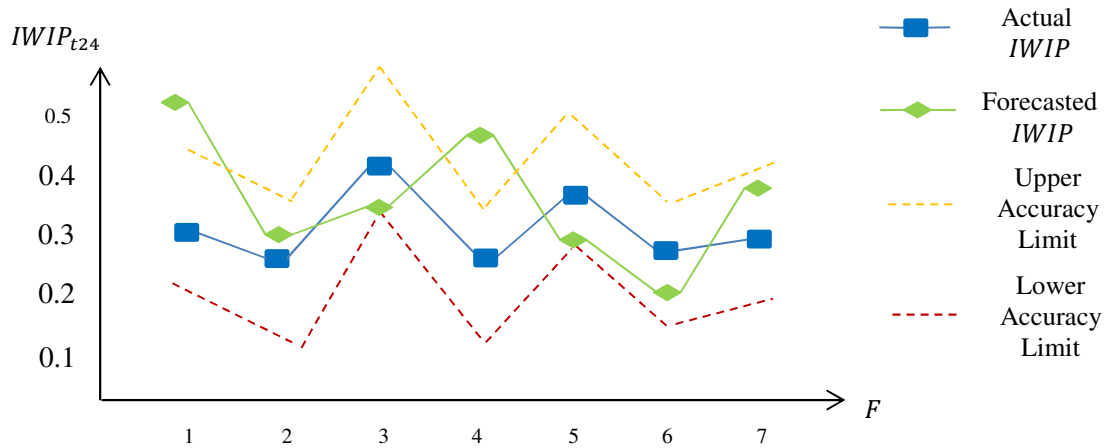


Figure 3.11: First example of graph plots for actual IWIP and predicted IWIP

In Figure 3.11, the Y-axis represents the $IWIP$ at the 24th hour of each manufacturing day and the X-axis represents the forecast day. The square represents the actual $IWIP$ while the diamond represents the predicted $IWIP$. The dotted line above the graph of the actual $IWIP$ represents the acceptable upper ten percent deviation limit of the predicted $IWIP$ from the actual $IWIP$, while the dotted line below the graph of the actual $IWIP$ represents the acceptable lower ten percent deviation limits of the predicted $IWIP$.

from the actual *IWIP*. The values of upper and lower limits are obtained using Equation (3.16).

In the given example of Figure 3.11, four trend sets from the forecast results correctly follow the actual trend. These trend sets are set 1, 2, 5 and 6. In addition, there are two days of forecasted *IWIP* that are out of the accuracy range limit (i.e., the upper accuracy limit), which are the first and the fourth day. Figure 3.12 shows another example of the visual aid. In Figure 3.12, only one trend set was wrongly forecasted, which is set 5, and only one forecasted *IWIP* is out of the accuracy range limit (i.e., the upper accuracy limit), which is the first day.

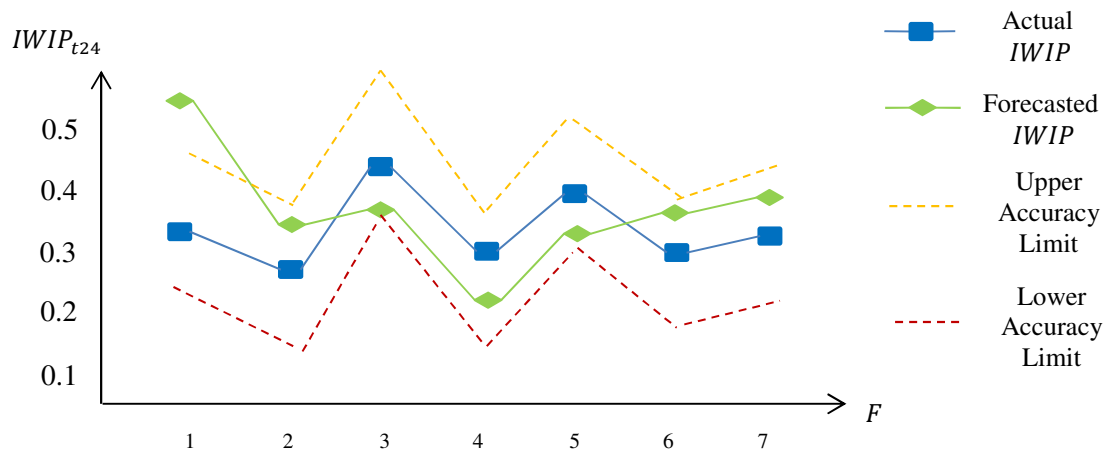


Figure 3.12: Second example of graph plots for actual *IWIP* and predicted *IWIP*

In addition to visualization purpose, the graph plot is also necessary to determine if the result is usable in practice. Consider Figure 3.13 that has the same trend as Figure 3.12. Although the trend accuracy of both results is the same, the model that produced the results in Figure 3.12 is preferable over the model that produced in Figure 3.13. This is because the trend in Figure 3.13 is not obvious and almost resembles a straight line. Trend that resembles a straight line could not represent the actual incoming *WIP* accurately because the peaks of the incoming *WIP* are indistinctive. Distinctive peaks of the

incoming WIP are important in identifying days ahead that has low incoming WIP so that PM planning can be scheduled accordingly.

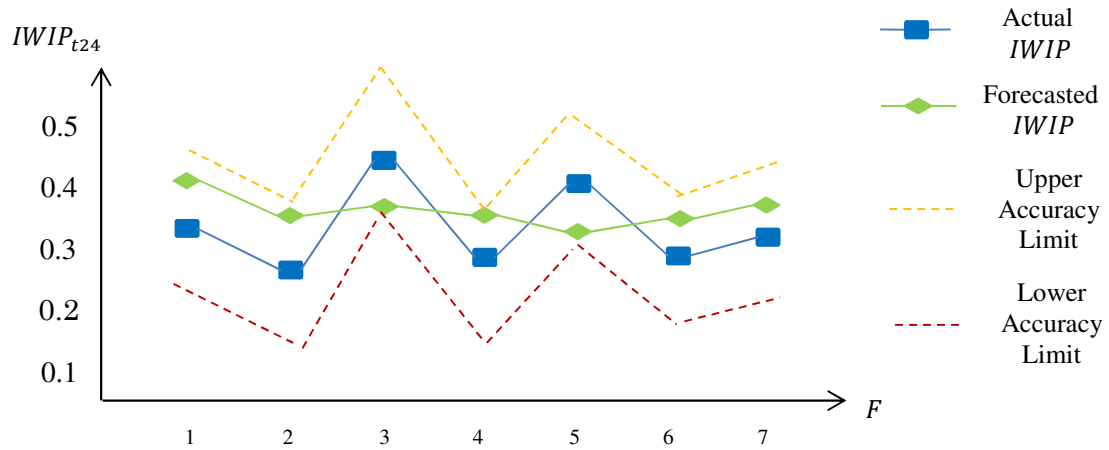


Figure 3.13: Third example of graph plots for actual IWIP and predicted IWIP

Therefore, visualization is also necessary in evaluating the forecasting results as it helps to assess the need to repeat the parameter sizes identification of the forecasting model.

3.9 Summary

In this chapter, the proposed methodology for this research work was presented. The motivation of this research work was first laid down followed by the presentation of the proposed framework. The data sources that are required to create the fundamental input data for the forecasting model were then presented, followed by the presentation of individual steps of the proposed framework. We then present the data acquisition step to gather the defined data sources, calculate the fundamental data and stores them to a central data storage location, followed by the steps to engineer the time series features. The steps to engineer the time series features consist of data formatting and data scaling. LSTM, which is a one of the RNN that has reached state of the art in sequence prediction, was selected as the neural network to create the forecasting model. The internal mechanisms of

LSTM that contributed to its distinguish performance in time series forecasting were presented. Having selected LSTM as the neural network to use for the forecasting model, the next step in developing the forecasting model was to identify the parameters sizes that give the optimum configuration to the LSTM neural network to perform an accurate forecast. The parameters that require size identifications are the epoch, the batch size and the hidden neuron. The parameter size identification methodology was discussed. After the size for each parameter has been identified, the LSTM is evaluated for its forecast accuracy using the selected parameters sizes. The forecast result is evaluated for its values accuracy and trend accuracy. The metrics used for each accuracy evaluation were discussed. In the next chapter, the implementation of the proposed methodology will be presented and its experimental results will be discussed.

CHAPTER 4

RESULTS AND ANALYSIS

In the previous chapter, the architectural concepts of neural networks and the detailed framework were laid down. In this chapter, we present the experiment implementation of this research work. The software tools required are first presented, followed by the experimental setup. Next, the results obtained from the experiments are presented, and lastly, discussion is made on the obtained results.

4.1 Software Tools

The popularity of machine learning has led to creation of many new programming libraries, tools and framework to extend the capabilities of the existing programming languages to implement, optimize and alter the neural network models. One of such programming languages is Python. Python was first released in 1991 as a general purpose programming language that emphasizes on the readability of the codes and syntax that allows software programmers to express concepts in fewer lines than other programming languages such as C and Java. Python has gained popularity as the programming language for machine learning with the creation of stable and matured programming libraries such as Scikit-learn (Pedregosa et al., 2017), TensorFlow (Abadi et al., 2016) and Keras (Chollet, 2017).

Scikit-learn is an open source machine learning library built upon another open source library for Python called Scientific Python (SciPy) (Eads & Rosten, 2008) that is used for scientific and technical computing. SciPy contains software modules to cater the common tasks in science and engineering such as linear algebra, integration, interpolation, special functions and optimization. Scikit-learn extended the capabilities of SciPy by providing a range of supervised and unsupervised machine learning algorithms software

modules with Python application programming interface (API).

TensorFlow is another open source Python programming libraries that was developed by Google for the purpose of conducting machine learning and deep neural network research. It was released as open source in November 2015. TensorFlow processes numerical computation by means of data flow graphs where graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays that flow between them. The multidimensional data arrays are also called tensors. Its architecture allows computation to be deployed in one or more CPU or GPU in a desktop, server or mobile device without rewriting code.

Keras, on the other hand, is a high-level neural network API written in Python that was designed to enable fast experimentation with neural network without having to handle the complexity of the numerical operation in neural network in order to develop a functional neural network model during experimentations. Keras is capable of running on top of TensorFlow, which makes it a good choice as programming library in neural network research. Keras provides the LSTM neural network package in its prebuilt library, which removes the need to rewrite the LSTM from scratch in a research work.

In this research, Python will be used as the programming language together with Scikit-learn and Keras as the machine learning programming libraries. In order to use Scikit-learn and Keras, another three open source libraries for Python need to be installed. These three open source libraries are NumPy (Walt, Colbert & Varoquaux, 2011), Matplotlib (Hunter, 2007) and Pandas (McKinney, 2011). Briefly, NumPy is a programming library for Python that adds support for handling large, multi-dimensional arrays and matrices, and in addition, provides a large collection of high-level mathematical functions to operate on these arrays. While Matplotlib is a graph plotting open source library for Python and Numpy, and lastly, Pandas is a programming library written for

Python to extend its capability in data manipulation and analysis.

For local database, MySQL will be used as the database system to provide the centralized data storing facility. MySQL is a very popular open source relational database management system (RDBMS) that are well suited for less mission critical systems. MySQL is especially popular on Linux operating system due to its ease of installation with only a few lines of commands. Its suitability for less mission critical tasks also makes it easier to configure and manage. The open source tool that provides visual environment for configuring, managing and querying the MySQL database is called MySQL Workbench. MySQL provides API to interface with Python as well. This makes it well suited as the database system for this research.

4.2 Experiments Setup

Since all experiments are based on the real data from the Fab, permissions need to be granted so that proper technical procedures and resources can be allocated to support the data acquisition of this research. In total, we have collected 221 days of *WIP* and *MOVE* data. A virtual machine was setup for this research with the purpose to host the required local MySQL database as the centralized data location to store the data acquired from the production environment. We also used this virtual machine to install the software tools introduced in Section 4.1 for the forecasting model development. The specifications of virtual machine are with 4 Intel Xeon CPUs, 8 GB RAM and 10GB hard disk size, with Ubuntu 16.04 as the operating system.

Based on the collected 221 days of data, 200 data points of *IWIP* lags were reserved for supervised-learning to allow the LSTM model to discover the time-dependent relation in the dataset, while the remaining 21 data points of *IWIP* lags are reserved to evaluate the forecasting performance of the LSTM model. With 200 data points reserved

for supervised-learning, four different sizes of dataset were created to train the LSTM model. The four dataset sizes used were 50 data points, 100 data points, 150 data points and 200 data points.

The 21 data points reserved are the last 21 data points in the dataset, and they represent the most recent three subsequent weeks' data in the dataset. The three datasets were labeled as w_0 , w_1 and w_2 according to the week they represents. Therefore, the performance of the forecasting model will be evaluated for three consecutive weeks. Measuring the forecasting accuracy of the model for more than one forecasting week is necessary as the changes in the production environment are dynamic, and we would like to examine if our model is capable of modelling such production environment.

The LSTM network topology chosen contains one input layer, one hidden layer and one output layer. For epochs, numerical values of 100 and 200 are selected. For batch size, numerical values of 4, 8, and 12 are selected, and for hidden neuron, numerical values of 256 and 512 are selected. It is worthwhile to mention that by using seven *IWIP* points per dataset as the number of previous *IWIP* lags to be examined (refer to Table 3.5), each of the numerical values for batch size denotes the number of weeks presented to the LSTM model per epoch. The neural network is initialized with uniformly distributed weights where the ranges of the weights are $(-0.1, 0.1)$ and trained using *MSE* as the loss function (as defined in Equation (2.1)). Training is done by BPTT as explained in Section 3.6. Adam optimizer is used as the optimization function in BPTT to update the weights of LSTM during training with default learning rate, $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 0$ and $\gamma = 0$. Each combination of the selected values is then evaluated three times to obtain the three RMSE results of each combination. Parameter size selection is done by selecting the lowest RMSE among the three experimental runs followed by examining the graphs of the supervised-learning result of the same run that produced the lowest RMSE.

The desired supervised-learning graph should resemble the pattern illustrated in Figure 3.6. If the supervised-learning graph does not resemble Figure 3.6, the second lowest RMSE is examined. If none of the graphs obtained meet the required pattern, the parameter size combination is discarded from further evaluation.

4.3 Results

Table 4.1 tabulates the results of the parameter size combinations selections. For combinations that does not exhibit curve pattern similar to Figure 3.6, it is discarded by marking its RMSE column as “DISCARD”.

Table 4.1: Parameter size selection results

Dataset Size	Parameter			RMSE
	Epoch	Batch Size	Hidden Neuron	
50	100	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD
	200	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD
100	100	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD
	200	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD

Table 4.1: continued

150	100	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD
	200	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD
200	100	4	256	0.0089
		8		0.0091
		12		0.0086
		4	512	DISCARD
		8		DISCARD
		12		0.0095
	200	4	256	DISCARD
		8		DISCARD
		12		DISCARD
		4	512	DISCARD
		8		DISCARD
		12		DISCARD

Figure 4.1 – Figure 4.4 show the graphs of the supervised-learning results for the four selected combinations, respectively. From the figures, it can be seen that both lines are far apart although they move along in descending pattern. In addition, the line graph of the training is descending slowly and remained high at the end of the epoch. This sign signifies that the number of features provided to the neural network is insufficient for it to discover the time-dependent relation in the given incoming WIP data. However, the neural network was still able to forecast the incoming WIP that are quite close to the actual incoming WIP, as shown by the line graph of the testing's RMSE. With the weak discovery of time-dependent relation, we would expect the forecasting model to have lower trend accuracy.

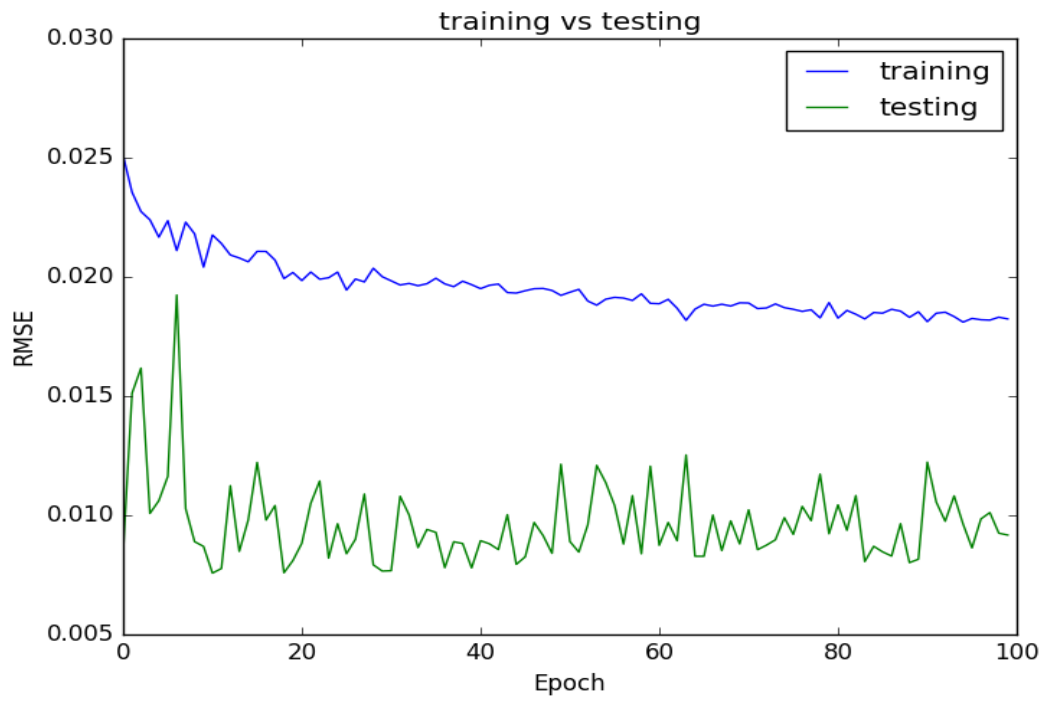


Figure 4.1: Supervised-learning result for 100 epochs, 4 batch size and 256 hidden neurons

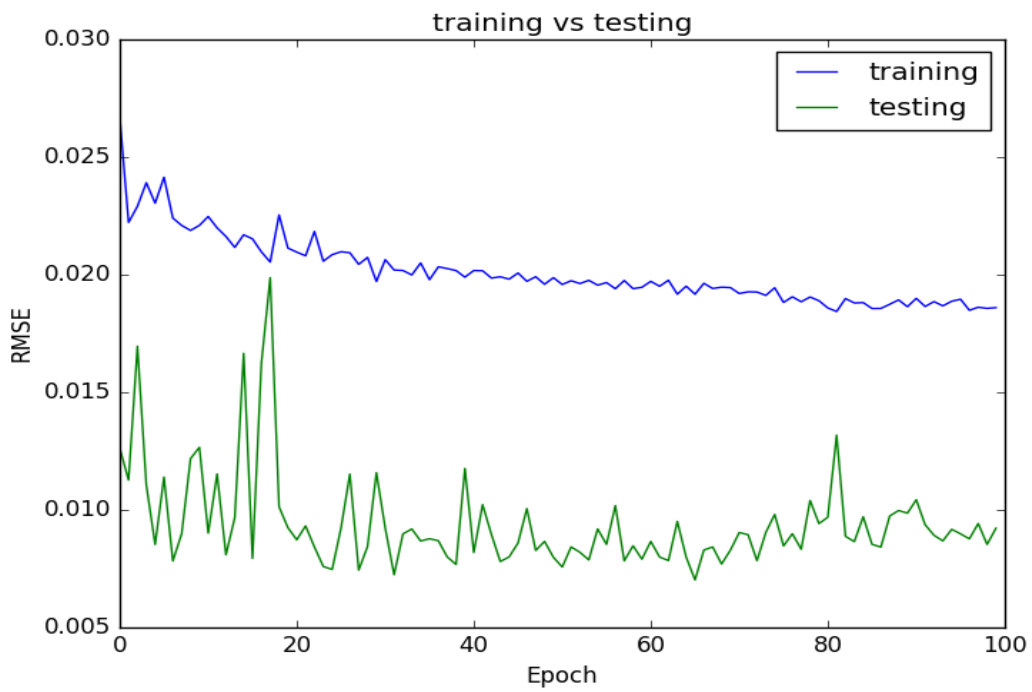


Figure 4.2: Supervised-learning result for 100 epochs, 8 batch size and 256 hidden neurons

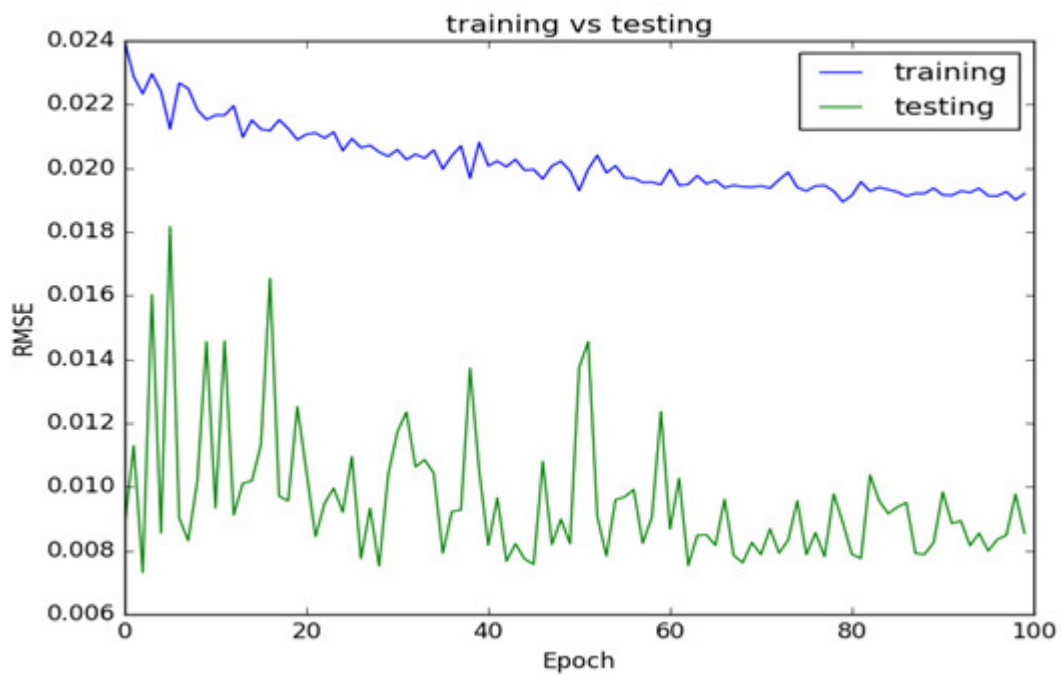


Figure 4.3: Supervised-learning result for 100 epochs, 12 batch size and 256 hidden neurons

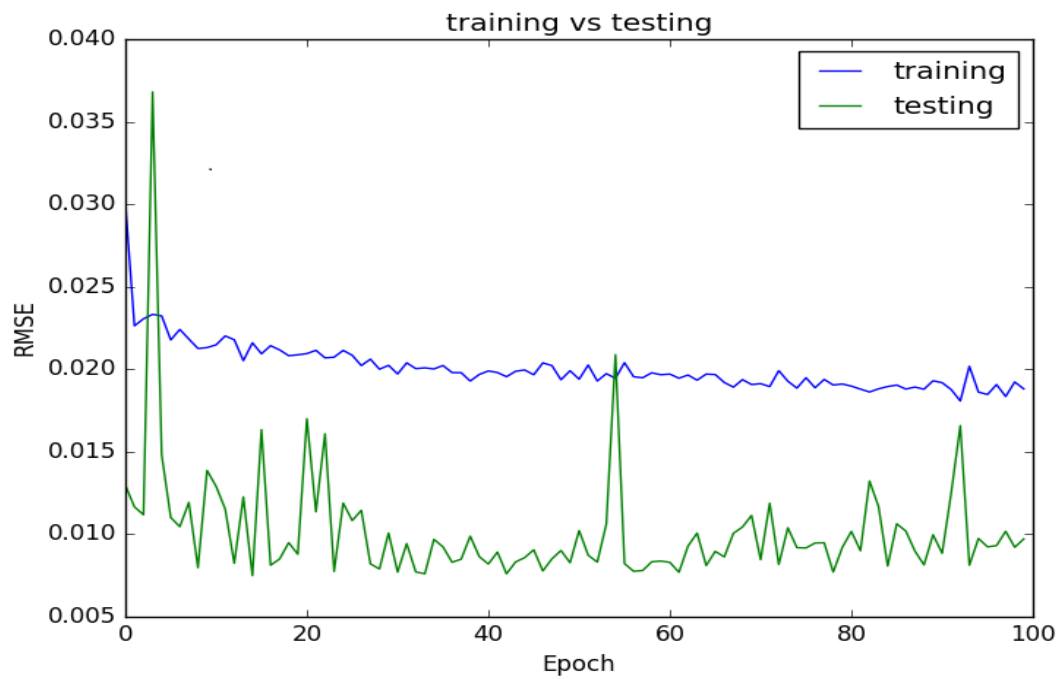


Figure 4.4: Supervised-learning result for 100 epochs, 12 batch size and 512 hidden neurons

By referring to Figure 4.1 – Figure 4.4 alone, we are not able to identify the best parameter size combination because all four graphs exhibit similar pattern. We would need to examine the value accuracies and trend accuracies of the forecasting results of each combination to identify the best parameter size combination.

For each of the selected parameter size combination, the respective parameters in the LSTM model were configured to the same size to perform forecasting for three consecutive weeks. In order to measure the forecasting performance of each model, the residual errors of the forecasted results is calculated. The residual error e , is calculated as

$$e = (y - \tilde{y})^2 \quad (4.1)$$

where $\tilde{y}$ represents the forecasted incoming WIP, y represents the actual incoming WIP. Let f represents the forecast day of each week, e^2 represents the squared residual error, and ACC represents the accuracy scored for each forecasted incoming WIP, Table 4.2 – Table 4.9 tabulates the accuracy results of the forecasting of the selected parameter size combinations.

Table 4.2 and Table 4.3 tabulate the forecasted value accuracy and trend accuracy respectively for parameter size combinations of 100 epochs, 4 batch sizes and 256 hidden neurons. Table 4.4 and Table 4.5 tabulate the forecasted value accuracy and trend accuracy respectively for parameter size combinations of 100 epochs, 8 batch sizes and 256 hidden neurons. Table 4.6 and Table 4.7 tabulate the forecasted value accuracy and trend accuracy respectively for parameter size combinations of 100 epochs, 12 batch sizes and 256 hidden neurons. Table 4.8 and Table 4.9 tabulate the forecasted value accuracy and trend accuracy respectively for parameter size combinations of 100 epochs, 12 batch sizes and 512 hidden neurons. Table 4.10 summarizes the value and trend accuracies for all four parameter size combinations in a single table.

Table 4.2: Value accuracy of forecasted result for 100 epochs, 4 batch size and 256 hidden neurons

Week	f	$\tilde{y}$	y	e	e^2		ACC	
w0	1	0.3846	0.4353	0.0507	0.0026		1	
	2	0.3970	0.4122	0.0153	0.0002		1	
	3	0.4112	0.4608	0.0496	0.0025		1	
	4	0.3960	0.3390	-0.0571	0.0033		1	
	5	0.4039	0.3651	-0.0387	0.0015		1	
	6	0.4203	0.3994	-0.0209	0.0004		1	
	7	0.4186	0.4121	-0.0065	0.0000		1	
					RMSE	0.0387	ACCP	100.0
w1	1	0.4020	0.3586	-0.0433	0.0019		1	
	2	0.3958	0.3422	-0.0537	0.0029		1	
	3	0.4160	0.4424	0.0263	0.0007		1	
	4	0.4214	0.4548	0.0334	0.0011		1	
	5	0.4160	0.4311	0.0150	0.0002		1	
	6	0.4133	0.3568	-0.0565	0.0032		1	
	7	0.4170	0.4323	0.0153	0.0002		1	
					RMSE	0.0382	ACCP	100.0
w2	1	0.4424	0.3828	-0.0596	0.0035		1	
	2	0.4154	0.3849	-0.0305	0.0009		1	
	3	0.4044	0.4713	0.0669	0.0045		1	
	4	0.4113	0.4134	0.0022	0.0000		1	
	5	0.4283	0.4240	-0.0043	0.0000		1	
	6	0.4157	0.3385	-0.0772	0.0060		0	
	7	0.4106	0.3997	-0.0109	0.0001		1	
					RMSE	0.0462	ACCP	85.7

Table 4.3: Trend accuracy of forecasted result for 100 epochs, 4 batch size and 256 hidden neurons

Week	Trend Set	Measurement Points	$\bar{y}$	y	ACC	
w0	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	1	1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	66.7
w1	1	(P_1, P_2)	-1	-1	1	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	1	1	1	
	4	(P_4, P_5)	-1	-1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	100.0
w2	1	(P_1, P_2)	-1	1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	50.0

Table 4.4: Value accuracy of forecasted result for 100 epochs, 8 batch size and 256 hidden neurons

Week	f	$\tilde{y}$	y	e	e^2		ACC	
w0	1	0.3889	0.4353	0.0464	0.0022		1	
	2	0.4023	0.4122	0.0100	0.0001		1	
	3	0.4089	0.4608	0.0520	0.0027		1	
	4	0.4001	0.3390	-0.0611	0.0037		0	
	5	0.4065	0.3651	-0.0413	0.0017		1	
	6	0.4182	0.3994	-0.0188	0.0004		1	
	7	0.4187	0.4121	-0.0067	0.0000		1	
					RMSE	0.0393	ACCP	85.7
w1	1	0.4025	0.3586	-0.0438	0.0019		1	
	2	0.3991	0.3422	-0.0569	0.0032		1	
	3	0.4140	0.4424	0.0283	0.0008		1	
	4	0.4201	0.4548	0.0347	0.0012		1	
	5	0.4172	0.4311	0.0138	0.0002		1	
	6	0.4148	0.3568	-0.0580	0.0034		1	
	7	0.4169	0.4323	0.0154	0.0002		1	
					RMSE	0.0395	ACCP	100.0
w2	1	0.4350	0.3828	-0.0522	0.0027		1	
	2	0.4206	0.3849	-0.0357	0.0013		1	
	3	0.4069	0.4713	0.0644	0.0041		1	
	4	0.4132	0.4134	0.0003	0.0000		1	
	5	0.4236	0.4240	0.0004	0.0000		1	
	6	0.4185	0.3385	-0.0801	0.0064		0	
	7	0.4137	0.3997	-0.0140	0.0002		1	
					RMSE	0.0458	ACCP	85.7

Table 4.5: Trend accuracy of forecasted result for 100 epochs, 8 batch size and 256 hidden neurons

Week	Trend Set	Measurement Points	$\tilde{y}$	y	ACC	
w0	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	1	1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	83.3
w1	1	(P_1, P_2)	-1	-1	1	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	1	1	1	
	4	(P_4, P_5)	-1	-1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	100.0
w2	1	(P_1, P_2)	-1	1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	1	-1	0	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	33.3

Table 4.6: Value accuracy of forecasted result for 100 epochs, 12 batch size and 256 hidden neurons

Week	f	$\bar{y}$	y	e	e^2		ACC	
w0	1	0.4010	0.4353	0.0344	0.0012		1	
	2	0.4029	0.4122	0.0093	0.0001		1	
	3	0.3970	0.4608	0.0639	0.0041		1	
	4	0.4027	0.3390	-0.0638	0.0041		0	
	5	0.4060	0.3651	-0.0408	0.0017		1	
	6	0.4146	0.3994	-0.0152	0.0002		1	
	7	0.4161	0.4121	-0.0040	0.0000		1	
					RMSE	0.0404	ACCP	85.7
w1	1	0.4000	0.3586	-0.0414	0.0017		1	
	2	0.4002	0.3422	-0.0580	0.0034		1	
	3	0.4113	0.4424	0.0311	0.0010		1	
	4	0.4182	0.4548	0.0366	0.0013		1	
	5	0.4158	0.4311	0.0153	0.0002		1	
	6	0.4142	0.3568	-0.0574	0.0033		1	
	7	0.4158	0.4323	0.0165	0.0003		1	
					RMSE	0.0400	ACCP	100.0
w2	1	0.4319	0.3828	-0.0491	0.0024		1	
	2	0.4133	0.3849	-0.0284	0.0008		1	
	3	0.4098	0.4713	0.0615	0.0038		1	
	4	0.4126	0.4134	0.0009	0.0000		1	
	5	0.4188	0.4240	0.0053	0.0000		1	
	6	0.4169	0.3385	-0.0785	0.0062		0	
	7	0.4123	0.3997	-0.0126	0.0002		1	
					RMSE	0.0438	ACCP	85.7

Table 4.7: Trend accuracy of forecasted result for 100 epochs, 12 batch size and 256 hidden neurons

Week	Trend Set	Measurement Points	$\tilde{y}$	y	ACC	
w0	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	1	-1	0	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	1	1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	50.0
w1	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	1	1	1	
	4	(P_4, P_5)	-1	-1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	83.3
w2	1	(P_1, P_2)	-1	1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	1	-1	0	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	33.3

Table 4.8: Value accuracy of forecasted result for 100 epochs, 12 batch size and 512 hidden neurons

Week	f	$\tilde{y}$	y	e	e^2		ACC	
w0	1	0.3859	0.4353	0.0495	0.0024		1	
	2	0.4019	0.4122	0.0104	0.0001		1	
	3	0.4185	0.4608	0.0424	0.0018		1	
	4	0.3957	0.3390	-0.0567	0.0032		1	
	5	0.4102	0.3651	-0.0450	0.0020		1	
	6	0.4297	0.3994	-0.0303	0.0009		1	
	7	0.4245	0.4121	-0.0124	0.0002		1	
					RMSE	0.0389	ACCP	100.0
w1	1	0.4004	0.3586	-0.0418	0.0017		1	
	2	0.3952	0.3422	-0.0531	0.0028		1	
	3	0.4260	0.4424	0.0164	0.0003		1	
	4	0.4269	0.4548	0.0279	0.0008		1	
	5	0.4185	0.4311	0.0125	0.0002		1	
	6	0.4158	0.3568	-0.0590	0.0035		1	
	7	0.4239	0.4323	0.0084	0.0001		1	
					RMSE	0.0367	ACCP	100.0
w2	1	0.4450	0.3828	-0.0622	0.0039		1	
	2	0.4166	0.3849	-0.0318	0.0010		1	
	3	0.4007	0.4713	0.0706	0.0050		1	
	4	0.4164	0.4134	-0.0029	0.0000		1	
	5	0.4366	0.4240	-0.0126	0.0002		1	
	6	0.4178	0.3385	-0.0793	0.0063		0	
	7	0.4107	0.3997	-0.0110	0.0001		1	
					RMSE	0.0486	ACCP	85.7

Table 4.9: Trend accuracy of forecasted result for 100 epochs, 12 batch size and 512 hidden neurons

Week	Trend Set	Measurement Points	$\tilde{y}$	y	ACC	
w0	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	1	1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	66.7
w1	1	(P_1, P_2)	-1	-1	1	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	1	1	1	
	4	(P_4, P_5)	-1	-1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	100.0
w2	1	(P_1, P_2)	-1	1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	1	-1	0	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	33.3

Table 4.10: Summary table of value accuracy and trend accuracy

Parameter			Value Accuracy			Trend Accuracy		
			Week			Week		
Epoch	Batch Size	Hidden Neuron	w0	w1	w2	w0	w1	w2
100	4	256	100.0	100.0	85.7	66.7	100.0	50.0
100	8	256	85.7	100.0	85.7	83.3	100.0	33.3
100	12	256	85.7	100.0	85.7	50.0	83.3	33.3
100	12	512	100.0	100.0	85.7	66.7	100.0	33.3

From the results obtained, the model performed the best using combinations 100 epochs, 4 batch size and 256 hidden neurons. In terms of value accuracy, both combinations 100 epochs, 4 batch size, 256 hidden neurons and 100 epochs, 12 batch size and 512 hidden neurons had the same performance where the model scored 100% accuracy for two weeks, but in terms of trend accuracy, combinations 100 epochs, 4 batch size, 256

hidden neurons performed better where on w2 forecast, the model scored 50% accuracy using 100 epochs, 4 batch size, 256 hidden neurons over 33.3% when 100 epochs, 12 batch size and 512 hidden neurons was used. Figure 4.5 – Figure 4.8 show the graph plots for the incoming WIP forecast for the four parameter size combinations, respectively.

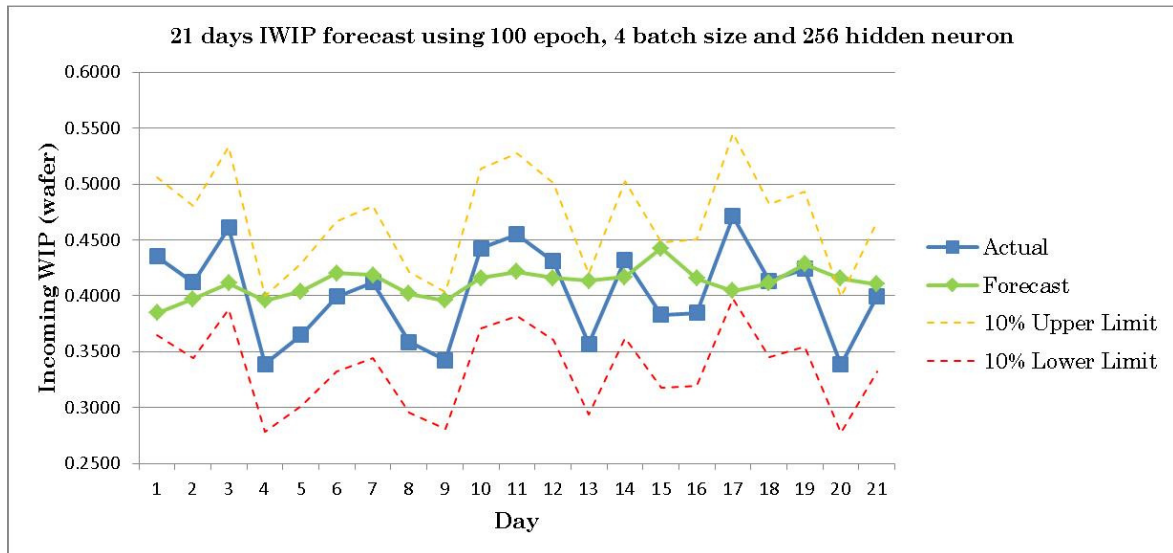


Figure 4.5: Incoming WIP forecast using 100 epoch, 4 batch size and 256 hidden neuron

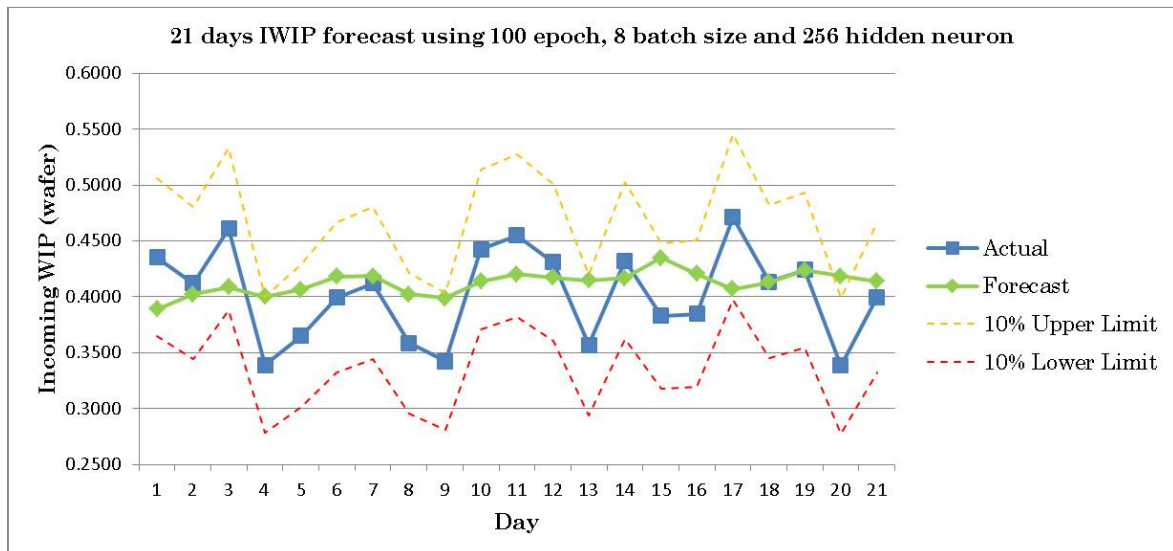


Figure 4.6: Incoming WIP forecast using 100 epoch, 8 batch size and 256 hidden neuron

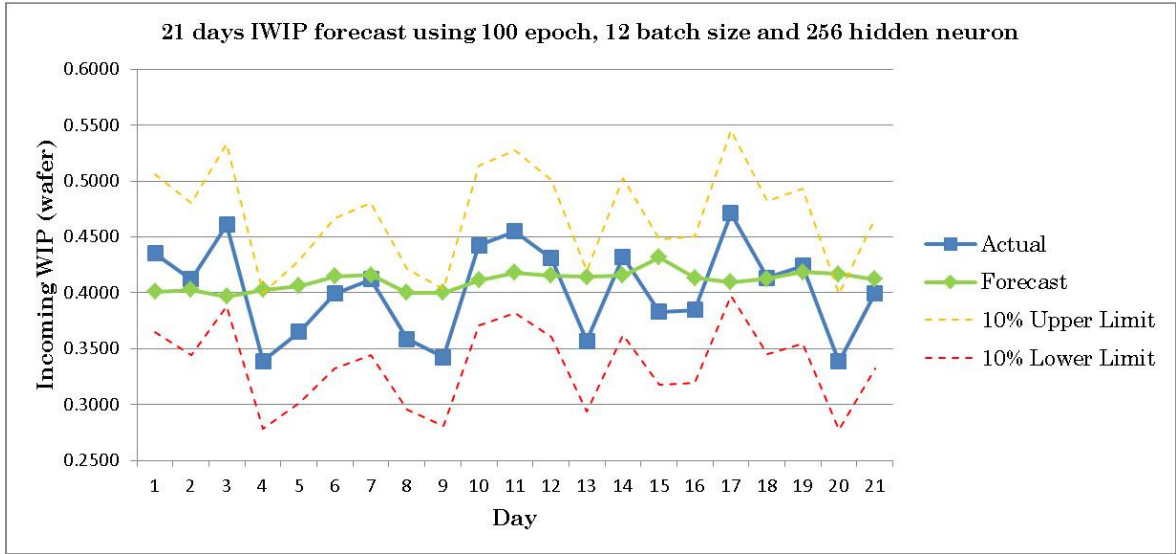


Figure 4.7: Incoming WIP forecast using 100 epoch, 12 batch size and 256 hidden neuron

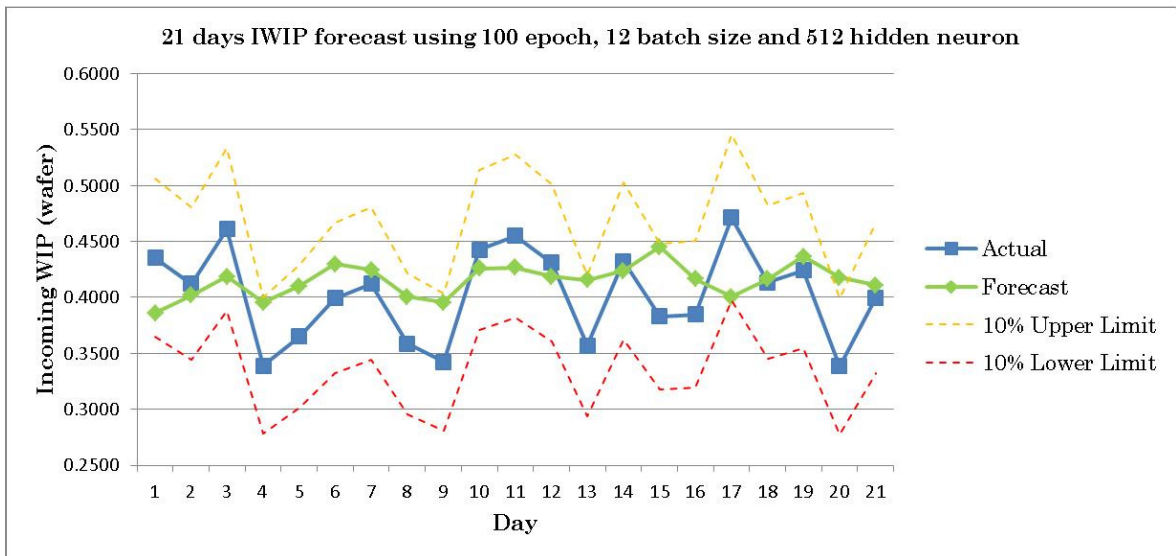


Figure 4.8: Incoming WIP forecast using 100 epoch, 12 batch size and 512 hidden neuron

We then compare the forecast result using 100 epochs, 4 batch size and 256 hidden neurons to compare with the current forecasting method. In order to make the writing clearer, current forecasting method used in the Fab is abbreviated as Fab method. Table 4.11 tabulates the value accuracy of Fab method, Table 4.12 tabulates the trend accuracy of Fab method, and Table 4.13 tabulates the comparison of the forecast results between LSTM and Fab method.

Table 4.11: Value accuracy of forecasted result for Fab method

Week	f	$\bar{y}$	y	e	e^2	ACC	
w0	1	0.8094	0.4353	-0.3741	0.1399	0	
	2	0.8469	0.4122	-0.4346	0.1889	0	
	3	0.8176	0.4608	-0.3568	0.1273	0	
	4	0.7552	0.3390	-0.4162	0.1732	0	
	5	0.7672	0.3651	-0.4021	0.1617	0	
	6	0.7779	0.3994	-0.3785	0.1433	0	
	7	0.7530	0.4121	-0.3409	0.1162	0	
					RMSE	0.3874	ACCP 0.0
w1	1	0.8109	0.3586	-0.4523	0.2046	0	
	2	0.8831	0.3422	-0.5409	0.2926	0	
	3	0.9805	0.4424	-0.5381	0.2896	0	
	4	0.9871	0.4548	-0.5323	0.2834	0	
	5	0.9709	0.4311	-0.5399	0.2915	0	
	6	0.9308	0.3568	-0.5740	0.3295	0	
	7	0.8949	0.4323	-0.4626	0.2140	0	
					RMSE	0.5217	ACCP 0.0
w2	1	0.9504	0.3828	-0.5676	0.3222	0	
	2	1.0490	0.3849	-0.6641	0.4410	0	
	3	1.1112	0.4713	-0.6399	0.4095	0	
	4	1.0871	0.4134	-0.6736	0.4538	0	
	5	1.0193	0.4240	-0.5953	0.3543	0	
	6	0.9428	0.3385	-0.6043	0.3652	0	
	7	0.9499	0.3997	-0.5502	0.3028	0	
					RMSE	0.6151	ACCP 0.0

Table 4.12: Trend accuracy of forecasted result for Fab method

Week	Trend Set	Measurement Points	$\bar{y}$	y	ACC	
w0	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	-1	1	0	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	1	1	1	
	5	(P_5, P_6)	1	1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	33.3
w1	1	(P_1, P_2)	1	-1	0	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	1	1	1	
	4	(P_4, P_5)	-1	-1	1	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	-1	1	0	
					ACCP	66.7
w2	1	(P_1, P_2)	1	1	1	
	2	(P_2, P_3)	1	1	1	
	3	(P_3, P_4)	-1	-1	1	
	4	(P_4, P_5)	-1	1	0	
	5	(P_5, P_6)	-1	-1	1	
	6	(P_6, P_7)	1	1	1	
					ACCP	85.7

Table 4.13: Forecast result comparison between LSTM and Fab

Method	Value Accuracy			Trend Accuracy		
	Week			Week		
	w0	w1	w2	w0	w1	w2
LSTM (100 epochs, 4 batch size, 256 hidden neurons)	100.0	100.0	85.7	66.7	100.0	50.0
FC	0.0	0.0	0.0	33.3	66.7	85.7

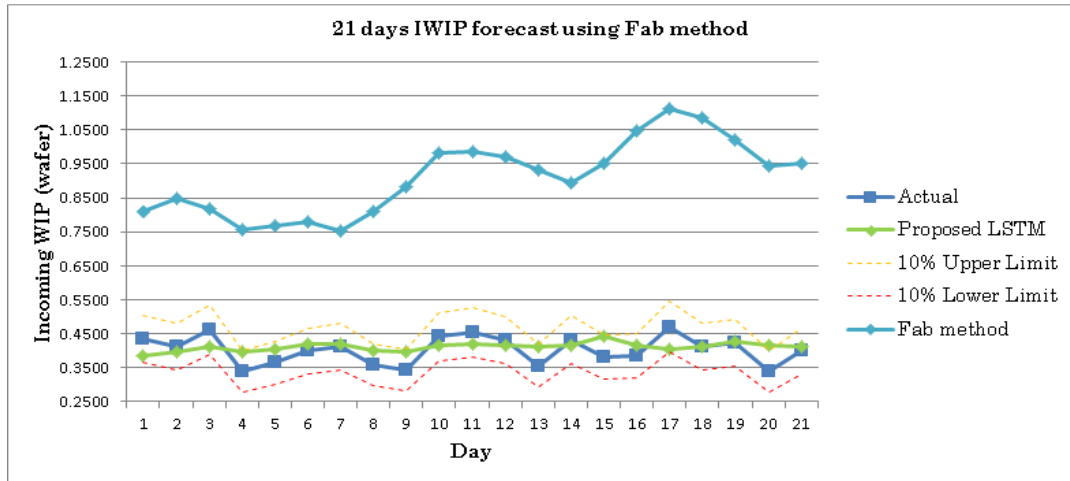


Figure 4.9: Incoming WIP forecast using FC method

4.4 Discussion

The Fab method serves as the baseline to measure the performance of the LSTM forecasting model. From the results obtained, the proposed method with LSTM forecasting model outperformed the Fab method. Table 4.14 tabulates the comparison of approaches both methods and their respective impact to the forecast results.

Table 4.14: Comparison of the modeling approaches of the proposed method and the FC method

Proposed model	FC method	Impact to forecast results
Does not make assumption of non-resource sharing	Makes assumption of dedicated resources	- Proposed model's forecast stays closer to actual - FC method's forecast is much higher than actual
Not product specific – models the WIP	- Requires modeling of the capacity of the equipment to process wafer (OEE) - Requires equipment specific modeling for their capacity	- FC method - did not consider this hence affected its value accuracy - complex to model because tools are unique - Proposed model - uses historical data to model the pattern of the IWIP
Models IWIP as time series – each observation of IWIP is related to its previous	Makes assumption that each IWIP is an independent observation	Trend accuracy of proposed model is better than FC's method

With higher value and trend accuracy, Table 4.15 tabulates the reasons why PM planning is better using the proposed model.

Table 4.15: Advantages of PM planning using the proposed model over FC method

Proposed model	FC method
IWIP forecasts are more reasonable. Hence PM planning is clearer	Consistently higher than actual. Brings about uncertainty in PM planning
• Safeguard by value accuracy – forecasted IWIP within the manageable limit should trend is less accurate • Lower chance of wrong PM time allocation • Impact due to wrong PM planning is less severe	• No safeguard – consistently higher than actual • Higher chance of wrong PM time allocation • Impact due to wrong PM planning can be severe

From Table 4.13, it can be seen that the values forecasted by Fab method consistently failed to meet the 10% accuracy range from the actual value, and for trend accuracy, the Fab method was consistently below 90% accuracy. The main reason for the inaccuracy of the Fab method is that it only considers for products with the number of wafers ordered dominating the total WIP in the production line. However, operators need to process other wafers from other products as well. Hence, the wafers did not arrive on time as predicted. In addition, the Fab method does not consider the number of tools available at each process steps to process the wafers, and the total amount of time that each tool is used to process the wafers. In real environment, a tool can be taken offline for maintenance purposes or it could be used by the respective engineers to process specially crafted wafers for research and development purposes. Without taking into these considerations, the Fab method indirectly assumed that the number of tools available and the time of each tool dedicated to process the production purpose wafers are the same across the entire period of the wafer fabrication process.

From the results obtained of the proposed model in Table 4.13, it can be seen that both the value and trend accuracy of the proposed method did not remain consistent above

90%. This is a limitation observed for the proposed model. One of the factors that could contribute to the lack of trend accuracy could be due to the reason that single hidden layer is not optimum for LSTM to capture the hidden time-dependency relations in the given time series data. Although according to the universal approximation theorem which states that a MLP with a single hidden layer and sufficient amount of neurons can approximate any measurable function regardless of activation function used and the dimension of the inputs (Hornik et al., 1989), the forecasting results obtained may suggest the need of more hidden neurons. .

The next factor that could contribute to the inconsistent result of the proposed model is the size of the historical data to train the LSTM model. Larger historical incoming WIP data could potentially allow the LSTM model of the proposed method to discover more time-dependent relations in the Fab's production environment. With the additional time-dependent relations discovered, the accuracy of the model's forecasting can be increased.

The last factor that could contribute to the inconsistent result of the proposed model is the limited number of features that was used to represent the Fab's environment. Having additional features to represent the Fab's environment could allow LSTM to perform better modeling of WIP arrival pattern in the Fab environment. The examples of additional data that could serve as such features are actual number of equipment that supplies the WIP to the tool group of interest, the amount of time each equipment in the tool group was processing production wafers instead of performing other maintenance activities, and the number of wafers that each equipment in the tool group of interest actually processed.

In neural network based solution, the smallest parameter sizes combination that returns the best result is preferred over higher parameter sizes. This preference is because the higher the parameter sizes used, the more computing resources in terms of CPU power

and RAM allocation are required. However, comparing the incoming WIP graphs of the LSTM model with 100 epochs, 4 batch size, 256 hidden neurons and 100 epochs, 12 batch size, 512 hidden neurons in Figure 4.5 and Figure 4.8, respectively, it can be noticed that the forecasted trend using 4 batch size and 256 hidden neurons resembles a flat line more than 12 batch size and 512 hidden neurons, using the same number of epochs. Forecast result resembling flat line is less preferred in real production environment because there is no distinguishable peak in the result to indicate if the incoming WIP on a particular day ahead will be high or low. Without knowing the actual incoming WIP, the engineers would prefer the forecast result using 12 batch size and 512 hidden neurons as the peak of the forecasted trend are more visible. The flat line condition in the result could be caused by the failure of Adam optimizer to find the local optimum point in the gradient to minimize error of the loss function. Potential improvement to this weakness of the model can be done by increasing the number of hidden layers of the LSTM model and increasing the number of features that represents the Fab's environments.

From the experiments conducted, the selection of sizes for the LSTM model's parameters and the number of experimental runs were largely based on the hardware resources allocated, and the software capability setup for this experiment. From the hardware resource perspective, sufficient CPU should be allocated in the computing machine, while from the software capability perspective, parallelization should be enabled in the TensorFlow through special installation method. With the amount of hardware resources allocated in this work and parallelization enabled in TensorFlow, it took approximately 3 hours to complete one full experiment. For real production deployment, 3 hours is still consider long to obtain one complete set of results. Parallelization with sufficient number of CPUs in the computing machine are therefore critical in the production environment as the results should be obtained as fast as possible in order for the

managements to make the necessary decision for production line stability. Hence, proper hardware planning is required for production deployment of this system.

4.5 Summary

In this chapter, the results and analysis of the proposed method was presented. The software tools used in this research was first presented followed by the description of the experimental setup. We then presented the results of the experiment and discussed on the obtained results. The results obtained from the proposed method were compared with the results from the Fab's method. Although the results of the proposed method were better than the Fab method, its accuracies were not consistently above 90% as required. The lack of accuracy consistency in the forecasting results of the proposed method was potentially due to the lack hidden layers to model the time-dependency relations in the dataset. As such, potentially important arrival pattern of the WIP across the time series were not captured and thus, affecting the accuracy and consistency of the forecast results. The accuracy inconsistency of the forecast results could also be caused by the need for bigger historical dataset to train the forecasting model. Large historical dataset potentially contains more information that could assist the LSTM model in discovering the WIP arrival patterns across the time horizon. Lastly, the accuracy consistency of the forecasting results could be affected by the limited number of features that represents the Fab's environment. With features such as the number of equipment that supplies the WIP to the targeted tool group and the number of equipment in the targeted tool group that processes the WIP, it may be able to increase the capability of the LSTM model to model the WIP arrival pattern to the targeted tool group. Although the results of the proposed method lacks consistent accuracies, the result has proved that forecasting the incoming WIP using the proposed method was a great improvement compare to the Fab's method,

and the proposed method has demonstrated clear potential to deliver a better forecasting result than the Fab's method with further improvement works. In the next chapter, we present the conclusion of this work and the potential future works that can be carried out to improve the accuracies of the proposed method.

CHAPTER 5

CONCLUSION

In this chapter, we present a summary of the research's contributions, and finally conclude this research by presenting the conclusion and the future works for this research.

5.1 Summary of Research Contributions

In this work, three research objectives were first established and presented in Chapter 1. The first research objective was to develop a forecasting model utilizing machine learning approach that can model the time-dependent behavior in the incoming WIP. The second research objective was to develop a generic forecasting model that could cater to the incoming WIP of all products in the production line. The third research objective was to develop a forecasting model capable for forecasting the incoming WIP within the accuracy of the Fab to assist in PM activities planning.

Towards the end of this work, the following research contributions were achieved:

- a) Generic incoming WIP forecasting model for all production line products of the Fab.

The proposed method presents an incoming WIP forecasting model that is generic to all products present in the production line. Hence, no product-specific preparation approach is required.

- b) Generic incoming WIP forecasting model for all tool groups in the Fab.

The proposed method presents a general incoming WIP forecasting model that can be used to forecast the incoming WIP of any tool group of interest in the Fab as long as the tool group of interest has historical incoming WIP data.

- c) Adopted machine learning in incoming WIP forecasting.

The proposed method demonstrated the potential capability of machine learning in incoming WIP forecasting with improved accuracies than statistical forecasting

method. Specifically, machine learning method is capable of modeling the time series behavior of the incoming WIP with the use of LSTM neural network which statistical forecasting method could not.

5.2 Conclusion and Future Works

In this work, an incoming WIP forecasting model using machine learning is proposed. Specifically, LSTM is used as the machine learning algorithm in the proposed forecasting model to forecast the incoming WIP of the Fab. The historical incoming WIP of the Fab was used as the data to train the forecasting model to predict the incoming WIP for three subsequent weeks. The results of the proposed forecasting model are compared with the Fab's statistical forecasting model. The comparison between the forecasting results of the two models showed that the proposed forecasting model performed better than the Fab's method. Although the proposed forecasting model did not consistently meet the 90% accuracy requirement of the Fab, the results of this work showed that the proposed forecasting model is of research significance and contains research opportunities to increase its accuracies.

To further improve the accuracies of the proposed forecasting model, various future works will be carried out. The first future work is the exploration of the extension of the number of hidden layers in the LSTM forecasting model. This approach is also an initial step to experiment the potential use of deep learning model in time series forecasting. The second future work is to increase number of historical dataset. With the use of larger historical dataset that spans across longer historical time horizon to train the LSTM model, it may be potential for the LSTM model to discover significant WIP arrival pattern that could have been missed out in smaller historical dataset. The next future work is to extend the univariate forecasting model in this research to multivariate forecasting model. The

reason for this extension is to allow the inclusion of more features to train the LSTM model so that the LSTM model can better model the actual environment of the Fab. Among the features targeted are the actual number of equipment that supplies the WIP to the tool group of interest, the amount of time each machine in the tool group is processing production wafers instead of performing other maintenance activities, and the number of wafers that each machine in the tool group of interest processed.

REFERENCES

- Abadi, A., Rajabioun, T., & Ioannou, P. A. (2015). Traffic Flow Prediction for Road Transportation Networks with Limited Traffic Data. *IEEE Transactions on Intelligent Transportation Systems*, 16(2), 653-662.
- Abadi, M. et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, 265-285.
- Abhishek, K., Singh, M.P., Ghosh, S., & Anand, A. (2012), Weather Forecasting Model using Artificial Neural Network, *Procedia Technology*, 4, 311-318.
- Ahmed, M. S., & Cook, A. R. (1979). Analysis of freeway traffic time-series data by using Box-Jenkins Techniques. *Transportation Research Record*, 722, 1-9.
- Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-Shishiny, H. (2012). An empirical comparison of machine learning models for time series forecasting. *Expert Systems with Applications*, 39(8), 7067-7083.
- Alpaydin, E. (2010). *Introduction to Machine Learning. Adaptive Computation and Machine Learning*. 2nd Edition. The MIT Press.
- AutoSched AP (ASAP). (n.d.). Retrieved March 28, 2018, from <http://www.appliedmaterials.com/global-services/automation-software/autosched-ap>
- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1), 1-127.
- Bengio, Y., Delalleau, O., & Simard, C. (2010). Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4), 449-467.
- Bengio, Y., Goodfellow, I. J., & Courville, A. (2016). *Deep Learning*. The MIT Press.
- Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. *Large-scale Kernel Machines*, The MIT Press.
- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions*, 5(2), 157-166.

- Box, G. E., Jenkins, G.M., Reinsel, G.C., & Ljung, G.M. (2016). *Time series analysis: Forecasting and control*. Hoboken, NJ: John Wiley & Sons.
- Chollet, F. (2017). *Deep Learning with Python*. Greenwich, CT: Manning Publications Co.
- Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*, 160-167.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297.
- Duan, Y., Lv, Y., & Wang, F. Y. (2016). Travel time prediction with LSTM neural network. *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, 1053-1058.
- Eads, D., & Rosten, E. (2008). Experiences Using SciPy for Computer Vision Research. *Proceedings of the 7th Python in Science Conference*, Pasadena, CA, 22-27.
- Elman, J. L. (1990). *Finding Structure in Time*, John Wiley & Sons. Ltd.
- ElSaid, A., Wild, B., Higgins, J., & Desell, T. (2016). Using LSTM recurrent neural networks to predict excess vibration events in aircraft engines. *2016 IEEE 12th International Conference on e-Science*, Baltimore, MD, 260-269.
- Fischer, T., Krauss, C., & Treichel, A. (2018), Machine learning for time series forecasting - a simulation study, No 02/2018, *FAU Discussion Papers in Economics*, Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics.
- Fu, R., Zhang, Z., & Li, L. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Wuhan, 324-328.
- Gers F.A., Eck D., & Schmidhuber J. (2002) Applying LSTM to Time Series Predictable Through Time-Window Approaches. *Tagliaferri R., Marinaro M. (eds) Neural Nets WIRN Vietri-01. Perspectives in Neural Computing*. Springer, London, 193-200.
- Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2014). Multi-digit number recognition from street view imagery using deep convolutional neural networks. Retrieved

March 28, 2018, from <https://arxiv.org/abs/1312.6082>.

- Gooijer, G. D., & Hyndman, R. J. (2006). 25 years of time series forecasting. *International Journal of Forecasting*, 443-473.
- Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009). A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), 855-868.
- Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, 6645-6649.
- Guo, J.H. (2013). Back Propagation Through Time. Retrieved from <http://ir.hit.edu.cn/~jguo/docs/notes/bppt.pdf>
- Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1), 107-123.
- Hassoun, M. (2013). Prediction of Product Layer Cycle Time Using Data Mining. *Proceedings of the 2013 Winter Simulation Conference (WSC)*, Washington, DC, 3905-3911.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009), *The elements of statistical learning: data mining, inference and prediction*. 2nd edition. Springer-Verlag New York.
- Haykin, S. (2009). *Neural network and learning machines*. 3rd edition. Pearson Prentice Hall.
- Head, L. (1995). Event-Based Short-Term Traffic Flow Prediction Model. *Transportation Research Record*, 1(1510), 45-52.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507.
- Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen*. (Master's thesis), Technische Universität München.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal

- approximators. *Neural networks*, 2(5), 359-366.
- Huang, W., Song, G., Hong, H., & Xie, K. (2014). Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. *Intelligent Transportation Systems, IEEE Transactions*, 15(5), 2191-2201.
- Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. *Journal in Computing Science and Engineering*, 9(3), 90-95.
- Huval, B., Coates, A., & Ng, A. (2013). Deep learning for class-generic object detection. Retrieved March 28, 2018, from <https://arxiv.org/abs/1312.6885>.
- Hyndman, R. J., & Athanasopoulos, G. (2012). Forecasting: principles and practice. Retrieved July 18, 2017, from <https://www.otexts.org/fpp>
- Ji, H., Xu, A., Sui, X., & Li, L. (2010). The applied research of Kalman in the dynamic travel time prediction. *2010 18th International Conference on Geoinformatics*, Beijing, 1-5.
- Karlaftis, M., & Vlahogianni, E. (2011). Statistical methods versus neural networks in transportation research: Differences, similarities and some insights. *Transportation Research Part C: Emerging Technologies*, 19(3), 387-399.
- Kantz, H., & Schreiber, T. (2004). *Nonlinear time series analysis*. New York, NY, USA: Cambridge University Press.
- Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Retrieved March 28, 2018, from <https://arxiv.org/abs/1408.5882>
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. Retrieved March 28, 2018, from <https://arxiv.org/abs/1312.6980>.
- Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, 1, 1097-1105.
- Lai, J., Chen, B., Tan, T., Tong, S., & Yu, K. (2016). Phone-aware LSTM-RNN for voice conversion. *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, Chengdu, China, 177-182.

- Lapedes, A., & Farber, R. (1987). Nonlinear Signal Processing Using Neural Networks: Prediction and System Modelling. *Technical Report, Theoretical Division*, Los Alamos National Laboratory.
- Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F. (2015). Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2), 865-873.
- Ma, X., Tao, Z., Wang, Y., Yu, H., & Wang, Y. (2015). Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54(3), 187-197.
- Mckinney, W. (2011). Pandas: a Foundational Python Library for Data Analysis and Statistics. Retrieved March 28, 2018, from <http://pandas.sourceforge.net>
- Mohamed, A. R., Dahl, G. E., & Hinton, G. (2012). Acoustic modelling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1), 14-22.
- Montufar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems (NIPS) 2014*, 2924-2932.
- Oancea, B. (2014). Time Series Forecasting Using Neural Networks. Retrieved March 28, 2018, from <https://arxiv.org/abs/1401.1333>
- Ojemakinde, B. T. (2006). *Support Vector Regression for Non-Stationary Time Series* (Master's thesis), University of Tennessee.
- Okutani, I. (1987). The Kalman filtering approaches in some transportation and traffic problems. *Transportation Research Record*, 2(1), 397-416.
- Okutani, I., & Stephanedes, Y. J. (1984). Dynamic prediction of traffic volume through Kalman filtering theory. *Transportation Research Part B: Methodological*, 18(1), 1-11.
- Palit, A. K., & Popovic, D. (2005). *Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications*. Springer-Verlag London.
- Parascandolo, G., Huttunen, H., & Virtanen, T. (2016). Recurrent neural networks for polyphonic

- sound event detection in real life recordings. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6440-6444.
- Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(1), 2825-2830.
- Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1-17.
- Poskitt, D. S., & Tremayne, A. R. (1986). The selection and use of linear and bilinear time series models. *International Journal of Forecasting*, 2(1), 101-114.
- Quirk, M., & Serda, J. (2001). *Semiconductor Manufacturing Technology*. 1st edition. Pearson Prentice Hall.
- Ramirez-Hernandez, J. A. et al. (2010, August). Optimal Preventive Maintenance Scheduling in Semiconductor Manufacturing Systems: Software Tool and Simulation Case Studies. *IEEE Transactions on Semiconductor Manufacturing*, 23(3), 477-489.
- Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*, Cambridge, The MIT Press.
- Ren, X., Lin, R., & Zou, H. (2011). A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, Beijing, 220-224.
- Rozen, K., & Byrne, N. M. (2016). Using simulation to improve semiconductor factory cycle time by segregation of Preventive Maintenance activities. *2016 Winter Simulation Conference (WSC)*, Washington, DC, 2676-2684.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. Retrieved March 28, 2018, from <https://arxiv.org/abs/1609.04747>
- Rumelhart, D. E., & McClelland, J. L. (1987). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, 1, MIT Press, 318-362.
- Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory recurrent neural network

- architectures for large scale acoustic modelling. *Proceedings of the Annual Conference of the International Speech Communication Association*, 338-342.
- Scholl, W., Boon, P. G., & Lendermann, P. (2011). Implementation of a simulation-based short-term lot arrival forecast in a mature 200mm semiconductor FAB. *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, AZ, 2011, 1927-1938.
- Schwartz, J. D., & Rivera, D. E. (2009). A System Identification Approach to PDE Modeling of a Semiconductor Manufacturing Process. *IFAC Proceedings*, 42(10), 964-969.
- Shao, H., & Soong, B. H. (2016). Traffic flow prediction with Long Short-Term Memory Networks (LSTMs). *2016 IEEE Region 10 Conference (TENCON)*, Singapore, 2986-2989.
- Shin, H. C., Orton, M. R., Collins, D. J., Doran, S. J., & Leach, M. O. (2013). Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4D patient data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1930-1943.
- Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1), 132-150.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.
- Sutskever, L. (2014). Sequence to Sequence Learning with Neural Networks (2014). Retrieved March 28, 2018, from <https://arxiv.org/abs/1409.3215>.
- Taieb, S. B. (2014). *Machine learning strategies for multi-step-ahead time series forecasting*. (PhD Thesis). Universit Libre de Bruxelles, Belgium.
- Taieb, S. B., Bontempi, G., Atiya, A. F., & Sorjamaa, A. (2012). A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems with Applications*, 39(8), 7067-7083.
- Tang, B., Qiu, W., & Sun, X. (2008). Business Failure Prediction Using Exponential Smoothing Forecasting and Pattern Recognition. *2008 International Conference on Risk Management &*

Engineering Management, Beijing, 576-581.

- Tian, Y., & Pan, L. (2015). Predicting Short Term Traffic Flow by Long Short-Term Memory Recurrent Neural Network. *2015 IEEE International Conference on Smart City, Chengdu*, 153-158.
- Tieleman, T., & Hinton, G. (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *2012 COURSERA: Neural Networks for Machine Learning*, 4.
- Tong, H., & Lim, K. S. (1980). Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(3), 245-292.
- Toqué, F., Côme, E., El Mahrsi, M. K., & Oukhellou, L (2016). Forecasting dynamic public transport Origin-Destination matrices with long-Short term Memory recurrent neural networks. *2016 IEEE 19th International Conference on Intelligent Transportation Systems*, Rio de Janeiro, 1071-1076.
- VanderVoort, M., Dougherty, M., & Watson, S. (1996). Combining Kohonen maps with ARIMA time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5), 307-318.
- Vlahogianni, E.I., Karlaftis, M. G., & Golias, J. C. (2005). Optimized and meta-optimized neural networks for short-term traffic flow prediction: a genetic approach. *Transportation Research Part C: Emerging Technologies*, 13(3), 211-234.
- Walt, S. V. D., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Journal of Computing in Science and Engineering*, 13(2), 22-30.
- Wang, J., Zhang, J., & Wang, X (2018a). A Data Driven Cycle Time Prediction With Feature Selection in a Semiconductor Wafer Fabrication System. *2018 IEEE Transactions on Semiconductor Manufacturing*, 31(1), 173-182.
- Wang, J., Zhang, J., & Wang, X (2018b). Bilateral LSTM: A Two-Dimensional Long Short-Term Memory Model With Multiply Memory Units for Short-Term Cycle Time Forecasting in Re-

- entrant Manufacturing Systems. *2018 IEEE Transactions on Industrial Informatics*, 14(2), 748-758.
- Wang, Y., & Papageorgiou, M. (2005). Real-time freeway traffic state estimation based on extended Kalman filter: A general approach. *Transportation Research Part B: Methodological*, 39(2), 141-167.
- Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. (PhD thesis). Harvard University, Cambridge, MA.
- Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10), 1550-1560.
- Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4), 339-356.
- Williams, B. M., & Hoel, L.A. (2003). Modelling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results. *Journal of Transportation Engineering*, 129(6), 664-672.
- Xie, Y., Zhang, Y., & Ye, Z. (2007). Short-term traffic volume forecasting using Kalman filter with discrete wavelet decomposition. *Computer-Aided Civil and Infrastructure Engineering*, 22(5), 326-334.
- Zeiler M.D., & Fergus R. (2014) Visualizing and Understanding Convolutional Networks. *Computer Vision – ECCV 2014. Lecture Notes in Computer Science*, Springer, Cham. 8689, 818-833.
- Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14(1), 35-62.
- Zhang, K., Xu, J., Min, M. R., Jiang, G., Pelechris, K., & Zhang, H. (2016). Automated IT system failure prediction: A deep learning approach. *2016 IEEE International Conference on Big Data (Big Data)*, Washington, DC, 1291-1300.
- Zhao, R., Wang, J., Yan, R., & Mao, K. (2016). Machine health monitoring with LSTM networks. *2016 10th International Conference on Sensing Technology (ICST)*, Nanjing, 1-6.

Zhu; G., Zhang; L., Shen; P., & Song, J. (2017). Multimodal Gesture Recognition Using 3D Convolution and Convolutional LSTM. In *IEEE Access*, 5, 4517-4524.