

NANYANG
TECHNOLOGICAL
UNIVERSITY

Recurrent Neuro-Fuzzy Parallel System (RNFPS)

implemented with an automated trading strategy

by

Koh Liang Jing

SCSE20-0787

School of Computer Science and Engineering

2020 - 2021

NANYANG TECHNOLOGICAL UNIVERSITY

Recurrent Neuro-Fuzzy Parallel System (RNFPS)

implemented with an automated trading strategy

Submitted in Partial Fulfillment of Requirements

for the Degree of Bachelor of Computer Engineering

of the Nanyang Technological University

Submitted by: Koh Liang Jing

Supervisor: Prof Quek Hiok Chai

SCSE20-0787

School of Computer Science and Engineering

2020 - 2021

Abstract

Neural network usage has been on a rise, as various companies and countries are beginning to incorporate Artificial Intelligence into daily life. Neural networks have been used in eCommerce to personalize the user experience, Finance to detect fraud, Healthcare to propose the best treatment pattern and so much more.

Even though neural networks could be used to predict situations with such high importance as healthcare, neural networks have a black box behavior which makes it seem unreliable and uncertain. Without a proper explanation of the predicted results, humans tend to feel unsure and will not be able to trust the system. Where fuzzy logic is able to match human linguistic terms to systems, even for neural networks.

In this paper, the usage of fuzzy neural networks like Yager Pattern Classifying Fuzzy Neural Network (Yager-PC FNN) and Modified Yager Fuzzy Neural Network (Modified Yager FNN) will be explored. Including the various clustering techniques such as Modified Learning Vector Quantization (MLVQ), Fuzzy Kohonen Partitioning (PFKP), and Unsupervised Discrete Clustering Technique (UDCT). The combination of fuzzy logic with Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM) will be explored as well.

This paper will propose a novel architecture called Recurrent Neuro-Fuzzy Parallel System (RNFPS) which integrates a fuzzy system and recurrent neural network together, which aims to solve the neural network's issue of a black box behavior. The system will be able to apprehend sequential data with superior accuracy and interpretability, in various applications.

RNFPS provides the crisp inputs into the fuzzy system and the recurrent neural network at the same time. The recurrent neural network used is Long Short-Term Memory (LSTM) which tries to determine the sequential pattern, predicts the output, and eventually delivers the information needed to the fuzzy system by tagging the respective nodes in the fuzzy system.

Lastly, RNFPS will experiment with traffic data and the stock market. With the main focus on the stock market, which an automated trading strategy will be introduced as well. A genetic algorithm will be utilized to optimize RNFPS and the trading strategy. The trading system is able to generate

buy and sell signals days in advance owing to the ability to predict the future close prices of a chosen stock.

Acknowledgment

I would like to express my sincere gratitude to my Final Year Project supervisor, Dr. Quek Hiok Chai for his guidance, motivation, and encouragement throughout this research project. His willingness to provide valuable and constructive advice during the planning and development of this project has helped inspire me and contributed greatly to the success of this project.

I would like to offer my special thanks to Tavis for his love and understanding, more importantly, for sharing his knowledge, and for his endless encouragement that rallied me on through this project.

Finally, my deepest gratitude goes to my family and friends for their constant support and encouragement.

Table of Contents

Abstract	3
Acknowledgment	5
List of Figures	8
1 Introduction	10
1.1 Background	10
1.2 Research Gap	11
1.3 Objective	11
1.4 Scope	11
1.5 Organization of Report	11
2 Literature Review	12
2.1 Types of problems	12
2.2 Artificial Neural Networks (ANN)	13
2.3 Fuzzy System	17
2.4 Fuzzy Clustering	24
2.5 Fuzzy Neural Network	32
2.6 Optimization	39
2.7 Cross-validation (CV)	43
2.8 Stock market	44
3 Yager Pattern Classifying Fuzzy Neural Network (Yager-PC FNN)	50
3.1 Clustering on iris dataset	50
3.2 Yager-PC FNN with MLVQ, PFKP, and UDCT clustering technique	53
4 Yager-PC FNN tagged with Multi-Layer Perceptron	55
4.1 Hebbian Weight Process	56
4.2 Tagging Mechanism	57
4.3 Ability of the different optimizers	58
4.4 Ability of different activation functions	58
4.5 Yager-PC FNN versus Yager-PC FNN tagged with Multi-Layer Perceptron	59
5 Modified Yager FNN	62

5.1 Clustering on traffic flow dataset	62
5.2 Modified Yager FNN with MLVQ, PFKP, and UDCT clustering technique	66
6 Modified Yager FNN tagged with Multi-Layer Perceptron	68
6.1 Parameters used for Multi-Layer Perceptron (MLP)	68
6.2 Modified Yager FNN tagged with Multi-Layer Perceptron vs Modified Yager FNN	69
6.3 Modified Yager FNN tagged with Multi-Layer Perceptron vs Pure Multi-Layer Perceptron	72
7 Modified Yager-FNN tagged with Long Short-Term Memory	74
7.1 Recurrent Neuro-Fuzzy Parallel System (RNFPS)	74
7.2 Look ahead	75
7.3 Parameters used for Long Short-Term Memory	75
7.4 Recurrent Neuro-Fuzzy Parallel System (RNFPS) vs Modified Yager Network	76
7.5 Recurrent Neuro-Fuzzy Parallel System (RNFPS) vs Pure Long Short-Term Memory	78
8 Stock market	83
8.2 Implementation of Recurrent Neuro-Fuzzy Parallel System (RNFPS)	86
8.3 Look forward	87
8.5 Accuracy of Recurrent Neuro-Fuzzy Parallel System used to predict Stock Market prices	87
9 Trading Strategies and Analysis	89
9.1 Trading Strategies	89
9.2 Trading Analysis	91
9.3 Automated Trading Algorithms	91
10 Implementation of Automated Trading Algorithm with Recurrent Neuro-Fuzzy Parallel System	94
10.1 Automated Trading Algorithm	94
10.2 Result of Automated Trading Algorithm with Recurrent Neuro-Fuzzy Parallel System	95
11 Conclusion	98
11.1 Recommendations for future works	98
12 References	100
13 Appendix	106

List of Figures

Figure 2.1 Multi-Layer Perceptron with 2 hidden layers	13
Figure 2.2 Recurrent Neural Network with 2 hidden layers	15
Figure 2.3 Long Short Term Memory with 2 hidden layers	15
Figure 2.4 An example of Crisp Logic	16
Figure 2.5 An example of Fuzzy Logic	17
Figure 2.6 Graph of membership function for Crisp Set and Fuzzy Set	18
Figure 2.7 Graph of Triangular, Trapezoidal, Gaussian, and Bell-Shaped membership function	19
Figure 2.8 Flowchart of a Fuzzy Inference System	22
Figure 2.9 Architecture of Neural Network for Discrete Clustering Technique	27
Figure 2.10 Parallel fuzzy neural network system	31
Figure 2.11 Cascading fuzzy neural network systems	32
Figure 2.12 Structure of Fuzzy Logic Inference Neural Network	34
Figure 2.13 Structure of Yager-PC FNN	35
Figure 2.14 Structure of Modified Yager FNN	37
Figure 2.15 Flowchart of Genetic Algorithm	39
Figure 2.16 Graph of input vs output for a ReLu activation function	40
Figure 2.17 Graph of input vs output for a Sigmoid and a Tanh activation function	41
Figure 2.18 Technical indicators separated by categories	45
Figure 3.1 Membership Functions of Iris dataset identified by MLVQ	50
Figure 3.2 Membership Functions of Iris dataset identified by PFKP	51
Figure 3.3 Membership Functions of Iris dataset identified by UDCT	52
Figure 4.1 Architecture of Yager-PC FNN tagged with Multi-Layer Perceptron	54
Figure 4.2 Tagging Mechanism	56
Figure 5.1 Membership Functions of Traffic flow dataset identified by MLVQ	62
Figure 5.2 Membership Functions of Traffic flow dataset identified by PFKP	63
Figure 5.3 Membership Functions of Traffic flow dataset identified by UDCT	65

Figure 6.1 Architecture of Modified Yager FNN tagged with Multi-Layer Perceptron	67
Figure 6.2 flowchart of geneticalgorithm from python library	68
Figure 6.3 Result of Traffic data (fold 1) tested with Yager Network (UDCT)	69
Figure 6.4 Result of Traffic data (fold 1) tested with Fuzzy-MLP (UDCT & optimized node)	69
Figure 6.5 Result of Traffic data (fold 1) tested with Pure MLP & optimized node	71
Figure 6.6 Result of Traffic data (fold 1) tested with Fuzzy-MLP (UDCT & optimized node)	71
Figure 7.1 Architecture of Recurrent Neuro-Fuzzy Parallel System (RNFPS)	73
Figure 7.2 Architecture of hindsight LSTM	74
Figure 7.3 Architecture of look forward LSTM	74
Figure 7.4 Result of Traffic data (fold 1) tested with Yager Network (UDCT)	75
Figure 7.5 Result of Traffic data (fold 1) tested with Fuzzy-LSTM (UDCT & optimized node)	75
Figure 8.1 Prices of S&P between March 2016 to March 2019	83
Figure 8.2 Prices of Dow Jones Industrial between January 2020 to December 2020	84
Figure 8.3 Prices of GameStop between September 2020 to September 2021	85
Figure 8.4 The architecture of Recurrent Neuro-Fuzzy Parallel System (RNFPS)	85
Figure 8.5 Architecture of hindsight LSTM	86
Figure 8.6 Architecture of look forward LSTM	86
Figure 8.7 Result of Stock Market prediction (NDAQ) tested with RNFPS (look forward = 1)	87
Figure 8.8 Result of Stock Market prediction (NDAQ) tested with RNFPS (look forward = 7)	87
Figure 9.1 Pseudocode of Data Trader trading strategy	91
Figure 9.2 Pseudocode of Dourra trading strategy	91
Figure 10.1 Pseudocode of Genetic Algorithm	93
Figure 10.2 Diagram of Function used within Genetic Algorithm	93
Figure 10.3 Pseudocode of Function used within Genetic Algorithm	94
Figure 10.4 Pseudocode of Trading Algorithm	94
Figure 13 Result of traffic flow dataset optimized for all 6 folds	105-107

1 Introduction

1.1 Background

With the increasing amount and diversity of data being generated at a swift speed, it is crucial to turn the endless amount of data generated into useful information that can be analyzed and eventually utilized. [39] To utilize the raw data, artificial neural networks have been used for solving problems like clustering, regression, text classification, speech recognition, time series prediction, and many more. [44] At the same time, the accuracy of neural networks typically thrives along with the amount of data. Hence, with the immense amount of data along with the high accuracy of artificial neural networks, it has been broadly utilized in many areas, such as financial prediction, robot control, medical image recognition, and data mining. [44]

Even with such great qualities of artificial neural networks, there is a concerning issue, where it does not give any clue on why or how it obtains the result, hence reducing trust in the network. To tackle this issue, various have explored the possibility of a neuro-fuzzy system that combines fuzzy systems along with neural networks. Where fuzzy systems mimic human reasoning and the decision-making process by introducing the concept of linguistic variables, which is able to counterbalance the disadvantage of neural networks. There are various neuro-fuzzy systems that were developed, for example; ANFIS [40], GenSoFnn [41], HyFIS [42], POPFNN [43], Yager-PC FNN [35], and Modified Yager FNN [35].

Most of the systems mentioned above explore the usage of fuzzy neural networks to predict stock market prices along with an automatic trading system. Investors have been keen on predicting the stock market movement due to the potentially high returns. However, the nature of the stock market is highly volatile and has a dynamic nature; it is complicated to predict movement without intensive analysis. Hence, traders turn to time series forecasting and algorithmic trading to trade automatically using systems with high confidence which they usually turn to neural networks.

1.2 Research Gap

Hence, a neural network should be used along with a fuzzy system for users to understand the reasoning and implications of the prediction generated by the neural network.

1.3 Objective

The goal of the project is to implement a parallel fuzzy neural network to predict the stock market's movement accurately and eventually understand the decision-making process behind the prediction. The model consists of an artificial neural network and a fuzzy system used with a genetic algorithm to optimize the trading strategy.

1.4 Scope

Firstly, Python programming language will be used to implement the artificial neural network and fuzzy system. Secondly, both Yager-PC FNN and Modified Yager FNN will be experimented to determine their abilities. Additionally, Yager-PC FNN will be tagged with Multi-Layer Perceptron (MLP), while Modified Yager FNN will be tagged with both Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM) to evaluate if there is an increase in performance. The dataset used to determine the abilities of Yager-PC FNN, and Modified Yager FNN will be iris dataset and traffic dataset respectively. Thirdly, to predict the stock market movement, the architecture used will be Modified Yager FNN tagged with Long Short-Term Memory (LSTM) in parallel. Afterward, various trading strategies will be experimented with to determine which gives the highest profit. Lastly, to implement the best trading strategy from the experiment with the parallel architecture of Modified Yager FNN tagged with Long Short-Term Memory (LSTM). Thereafter, to evaluate the architecture implemented, the accuracy of the architecture will be tested using the trading action generated to calculate the profit earned.

1.5 Organization of Report

The report is organized into - chapters as follows:

- Chapter 1: Introduction which outlines the basic understanding of the research project
- Chapter 2: Literature review covering the topic associated with the project including artificial neural network, fuzzy system, clustering techniques, fuzzy neural network, optimization, cross-validation, stock market
- Chapter 3: Implementation of Yager-PC FNN tested with iris data
- Chapter 4: Implementation of Yager-PC FNN tagged with MLP tested with iris data
- Chapter 5: Implementation of Modified Yager FNN tested with traffic data
- Chapter 6: Implementation of Modified Yager FNN tagged with MLP tested with traffic data
- Chapter 7: Implementation of Recurrent Neuro-Fuzzy Parallel System tested with traffic data
- Chapter 8: Stock market
- Chapter 9: Trading strategies
- Chapter 10: Implementation of Automated Trading Algorithm with Recurrent Neuro-Fuzzy Parallel System
- Chapter 11: Conclusion

2 Literature Review

2.1 Types of problems

Before deciding on which neural network to utilize, one has to have a clear understanding of the type of problem that is being handled. In machine learning, there are two fundamental types of problems, which are classification and regression problems. The output variable is the main difference between the two problems, classification predicts a label while regression predicts a continuous value instead.

[1]

2.1.1 Classification

The main goal of predicting a classification problem is to approximate a mapping function f from input variables x to discrete output variables y . The output variables can also be called labels or

categories, where the mapping function predicts a class or category. To calculate the accuracy of a classification predictive model, it is as follow:

$$\text{accuracy} = (\text{correct prediction}/\text{total prediction}) * 100 \quad (2-1)$$

2.1.2 Regression

The main goal of predicting a regression problem is to approximate a mapping function f from input variables x to continuous output variables y . A continuous output variable refers to a real value like an integer or a floating point value. Regression model predicts a quantity, hence to calculate the accuracy, root mean squared error is normally used. The formula for root mean squared error is as follow:

$$RMSE = \sqrt{\text{average}(\text{error}^2)} \quad (2-2)$$

2.2 Artificial Neural Networks (ANN)

In machine learning terms, neural networks are a family of models that imitate the structure of a biological neural system that learns patterns by observation. [2] A neural network that contains more than one hidden layer is considered ‘deep’. The default type of neural network is a feed-forward network, which flows in only one direction, from input to output. However, there are ways to train models to use backpropagation which moves in the opposite direction, from output to the input. When using backpropagation, it computes and labels the error associated with each neuron, where models are able to fix its algorithm accordingly.

2.2.1 Deep Neural Network

A deep neural network refers to a neural network with multiple layers and is able to handle a huge amount of data.[3] It includes various classifiers working together, which are based on linear regression and activation functions. [4] Deep neural networks are able to reuse computed features in a given layer when iterating in a higher hidden layer, whereas, a dense neural network has to piece together the functions it estimated, mimicking a lookup table. [5] There are several types of deep

neural network which includes Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). [6]

Deep neural networks have proven time and again that it is able to provide results with strong accuracy, even with complicated and volatile problems such as stock market and usages in medical fields. However, there is a huge issue with deep neural network algorithms, where it is not able to provide any reasoning or explanation on how the results were being computed. This issue will be mentioned in a later chapter, mainly under fuzzy neural networks.

2.2.2 Multi-layer Perceptron (MLP)

Multi-layer Perceptron is a feedforward ANN that generates a set of outputs from a set of inputs. MLPs are the classical type of neural network, and it is created by stacking several fully-connected layers on top of one another. Each layer feeds into the above layer until the output is generated. It uses backpropagation to train the network. [7]

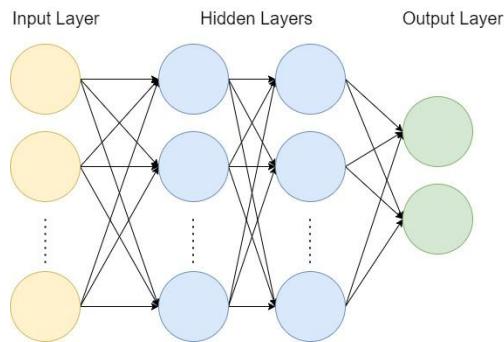


Figure 2.1 Multi-Layer Perceptron with 2 hidden layers

Every input affects every neuron in the hidden layer and in turn, each of these affects every neuron in the output layer. MLPs are generally very flexible which can be used to learn a general mapping from inputs to outputs. With such flexibility, MLPs can be applied to several types of problems. It is commonly used for tabular datasets such as classification prediction problems and regression prediction problems. However, it can also be used for image data, time-series data, and even text data.

There is an issue that MLPs face, where it is prone to overfit the data due to the full connectivity of the networks. It is possible to penalize the parameters during training or trim the connectivity, to prevent overfitting. [7]

2.2.3 Convolutional Neural Network (CNN)

Convolutional Neural Network is one of the popular deep neural networks where the name comes from a mathematical linear operation between matrices. It is specifically developed to map image data to an output variable.

CNN includes multiple layers, which consist of a combination of convolution, non-linearity, pooling, and fully connected layers. The convolution layer comprises a series of convolutional operations, each performing on a different slice of the input matrix.[3] The output shape of the convolution layer is affected by various variables like the shape of its input and the choice of the kernel shape, paddings, and strides. [8]

CNN is commonly used for image data as mentioned before, but it can be used for several different types of data such as classification prediction problems, regression prediction problems, time-series data, text data, and even sequence input data.

2.2.4 Recurrent Neural Network (RNN)

RNNs were specifically designed to handle sequence prediction problems. It is used when the context is important, meaning the decisions from the previous iteration or samples can influence the next. Hence the network is made to intentionally run multiple times, where values learned in the hidden layers from the first run become one of the inputs to the same hidden layers in the second run.

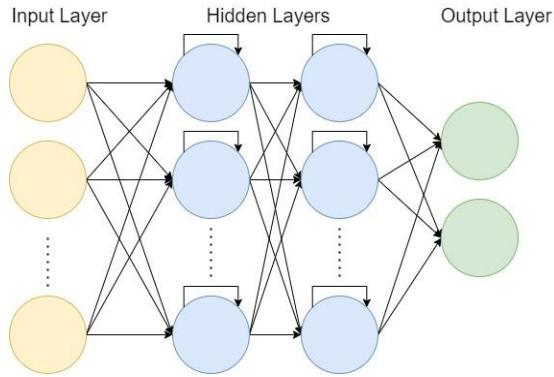


Figure 2.2 Recurrent Neural Network with 2 hidden layers - adapted from [9]

RNNs are commonly used for sequence data such as classification prediction problems, regression prediction problems, speech data, text data, generative models, and time-series data.

2.2.5 Long Short Term Memory (LSTM)

Long Short Term Memory is a variant of RNN that is used to handle the data sequences. LSTM is a special type of RNN that addresses the vanishing gradient issues that happen when training RNNs caused by long data sequences. It addresses the issues by retaining history in an internal memory state which is based on new input and context passed from the previous cells.

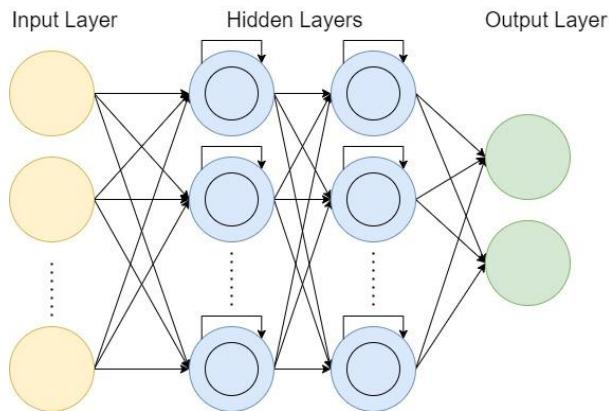


Figure 2.3 Long Short Term Memory with 2 hidden layers - adapted from [9]

Memory cells used by RNNs comprise several elements called gates, which control how information is being retained and cleared. Input gate determines how much information from the previous sample is being kept in the memory. While output gate controls the amount of data that is being passed to the next layer. And the forget gate regulates the forget rate of the memory stored. LSTM is commonly used for time series forecasting problems, handwriting recognition, machine translation, and image captioning.

2.2.6 Gated Recurrent Units (GRU)

2.2.7 Hybrid Neural Network

Hybrid neural network refers to a neural network that is constructed by merging neural networks with at least one other intelligent technology like reinforcement learning, fuzzy logic, etc. By combining several techniques on a computational model, it expands the range of capabilities of the system. These capabilities include learning and reasoning in an imprecise and uncertain environment. [10] Typically, hybrid networks outperform single-type networks in performance in accuracy and generalization. [11] There are hybrid deep neural networks that target mixed input as well, which includes data of various types or formats. For example, the diagram below shows a hybrid deep neural network that takes in four types of inputs; image, sequential logs, numerical and categorical tags. And the hybrid deep neural network includes CNN and MLP. [12]

2.3 Fuzzy System

2.3.1 Crisp Logic

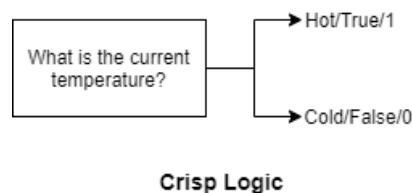


Figure 2.4 An example of Crisp Logic

Crisp logic refers to boolean logic where a crisp set contains only two values where it is either 0 (false) or 1 (true). For a crisp set, each element within the universe can only take up two kinds of values, which state that either it is a member of the crisp set or not.

2.3.2 Fuzzy Logic

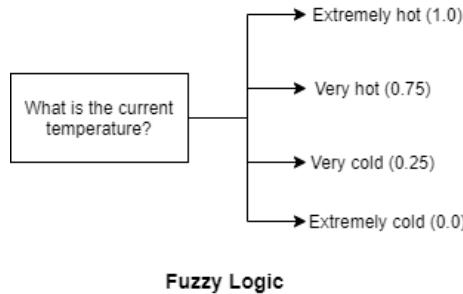


Figure 2.5 An example of Fuzzy Logic

However, fuzzy logic aims to model human logic and thoughts that tend to be abstract which could not be able to be explained using boolean logic. Fuzzy logic refers to the fact that the logic that takes on a continuous value is based on degrees of truth, which cannot be expressed using boolean logic, which modern computers are based on. Alternative approaches like neural networks and genetic algorithms can be used as well. Fuzzy logic has the edge, where the solution to the problem can be explained in linguistic terms, where humans are able to understand the reasoning behind the results.

[13]

2.3.3 Set theory

Crisp set comprises elements that satisfy the exact properties of membership while fuzzy set comprises elements that satisfy partial properties of membership. In other words, a fuzzy set allows partial membership, which means it contains elements with varying degrees of membership within the set.

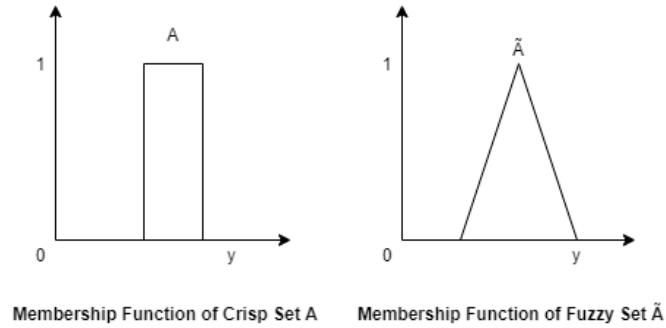


Figure 2.6 Graph of membership function for Crisp Set and Fuzzy Set

A fuzzy set is a pair (U, m) where U is a set (often required to be non-empty) and $m: U \rightarrow [0, 1]$ a membership function. The reference set U (sometimes denoted by Ω or X) is called universe of discourse, and for each $x \in U$, the value $m(x)$ is called the grade of membership of x in (U, m) . The function $m = \mu_A$ is called the membership function of the fuzzy set $A = (U, m)$.

There are three kinds of fuzzy set operations that are commonly used, namely fuzzy union, fuzzy intersection, and fuzzy complement.

	Fuzzy Set Operation	Membership Function	
Fuzzy Union (Fuzzy Or)	$C = A \cap B$	$\mu_C(x) = \min \{ \mu_A(x), \mu_B(x) \}$	(2-3)
Fuzzy Intersection (Fuzzy AND)	$C = A \cup B$	$\mu_C(x) = \max \{ \mu_A(x), \mu_B(x) \}$	(2-4)
Fuzzy Complement (Fuzzy NOT)	$C = \tilde{A}$	$\mu_C(x) = 1 - \mu_A(x)$	(2-5)

Table 2.1 Definition of the different Fuzzy Set Operations

2.3.4 Membership functions (MF)

A fuzzy set is a mapping of a set of real numbers onto membership values that generally lies in the range [0, 1], which means that a fuzzy set is characterized by its Membership Function (MF). In other words, the membership function represents the degree of truth.

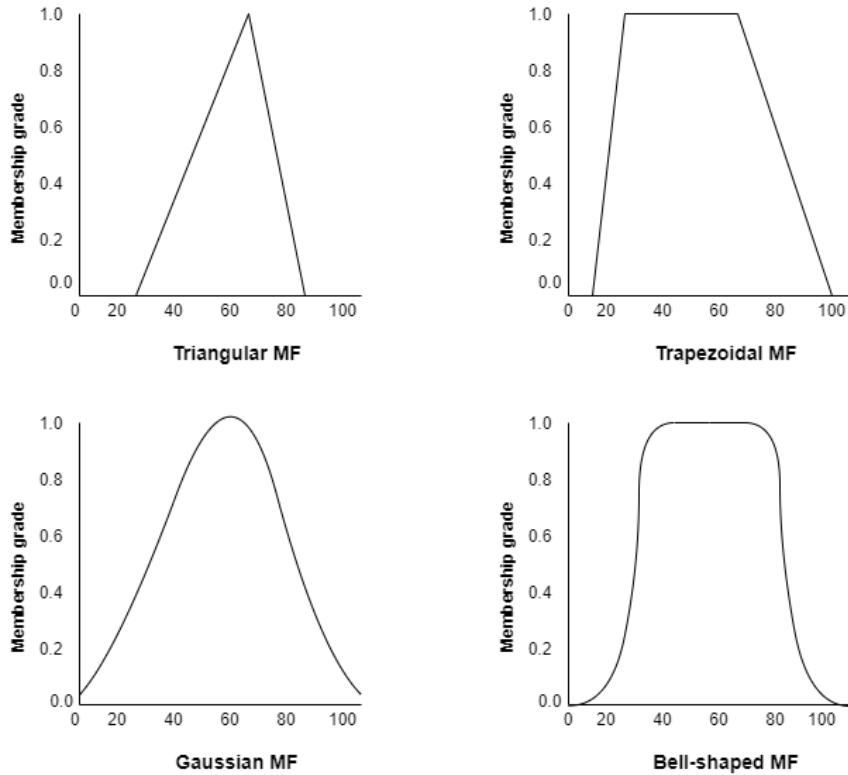


Figure 2.7 Graph of Triangular, Trapezoidal, Gaussian, and Bell-Shaped membership function

There are mainly four kinds of membership functions, namely; triangular, trapezoidal, Gaussian, and bell-shaped functions. Fuzzy set has an infinite amount of membership functions that may represent it. Hence a concise way to define membership functions is by expressing it as a mathematical formula.

Membership Functions	Expression
Triangular	<p>A triangular MF is specified by three parameters $\{a, b, c\}$ as follows:</p> $triangle(x; a, b, c) = \begin{cases} 0, & x \leq a. \\ \frac{x-a}{b-a}, & a \leq x \leq b. \\ 0, & x \geq b. \end{cases}$

	$\begin{aligned} \frac{c-x}{c-b}, & \quad b \leq x \leq c. \\ 0, & \quad c \leq x. \end{aligned} \tag{2-6}$
	An alternative expression is $triangle(x; a, b, c) = \max(\min(\frac{x-a}{b-a}, \frac{c-x}{c-b}), 0) \tag{2-7}$
Trapezoidal	A trapezoidal MF is specified by four parameters {a, b, c, d} as follows: $\begin{aligned} 0, & \quad x \leq a. \\ \frac{x-a}{b-a}, & \quad a \leq x \leq b. \\ trapezoid(x; a, b, c, d) = & \quad 1, \quad b \leq x \leq c. \\ & \quad \frac{d-x}{d-c}, \quad c \leq x \leq d. \\ & \quad 0, \quad d \leq x. \end{aligned} \tag{2-8}$
	An alternative expression is $trapezoid(x; a, b, c, d) = \max(\min(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}), 0) \tag{2-9}$
Gaussian	A gaussian MF is specified by two parameters {c, σ} as follows: $gaussian(x; c, \sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2} \tag{2-10}$
Bell-shaped	A bell-shaped Function is specified by three parameters {a,b,c} as follows: $bell(x; a, b, c) = \frac{1}{1 + \left \frac{x-c}{a}\right ^{2b}} \tag{2-11}$

Table 2.2 Definition of the Membership Functions

2.3.5 Fuzzification/Defuzzification

Fuzzification refers to the process of transforming crisp values into fuzzy values. Fuzzification utilizes if-then rules to fuzzify crisp values. Various fuzzification methods include inference, rank-ordering, neural network, and angular fuzzy set. While the defuzzification process is the opposite where it reduces a fuzzy member into a crisp member. Defuzzification utilizes the center of gravity methods to

obtain centroid of sets. Various defuzzification methods include centroid method, center of sums, weighted average method, and maximum membership principle.

2.3.6 Fuzzy If-Then Rules

Fuzzy if-then rule is also known as fuzzy rule, fuzzy conditional statement, or fuzzy implication. It can be generalized as

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B, (A \rightarrow B) \quad (2-12)$$

where A and B are linguistic values established by fuzzy sets on the universe of discourse X and Y respectively. '*x is A*' is the if part of the rule which is called the antecedent while '*y is B*' is the then-part of the rule which is called the consequent. Which can be expressed using human natural language as

$$\text{if antecedent then consequent.} \quad (2-13)$$

There are four terms that characterize the linguistic variable; name of the variable represented by *x*, term set of the variable represented by $t(x)$, syntactic rules, and semantic rules. Where syntactic rules generate the values of the variable while semantic rules link every value of variables and their significance.

2.3.7 Fuzzy Inference System (FIS)

Fuzzy Inference System refers to a system that uses fuzzy set theory to map inputs to outputs, where inputs refer to features and outputs refer to classes. Fuzzy inference process involves membership functions, fuzzy logic operators, and if-then rules. FIS includes five key units; a rule base, database, decision-making unit, fuzzification interface unit, and defuzzification unit. Where the fuzzification unit applies the fuzzification methods to the crisp inputs, turning them into fuzzy inputs. Thereafter, the decision-making unit converts fuzzy input into fuzzy output. Which defuzzification unit applies the defuzzification methods to fuzzy outputs, turning them into crisp output. Lastly, the knowledge

base contains a collection of rules and a database is created upon conversion of crisp inputs into fuzzy inputs.

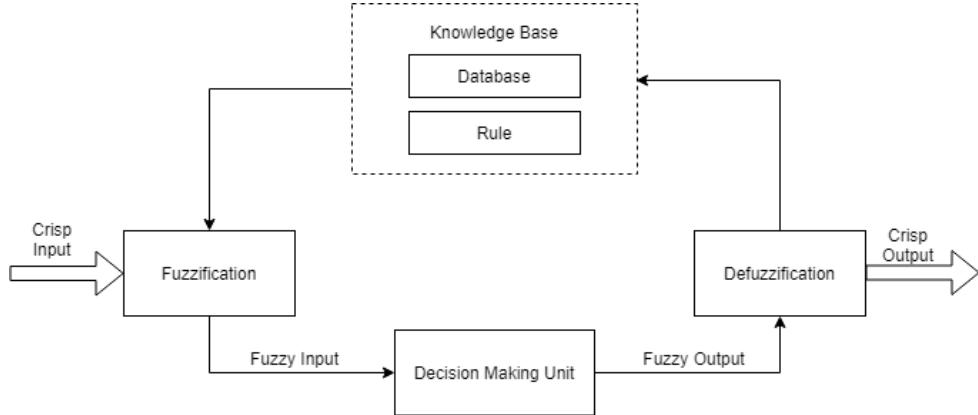


Figure 2.8 Flowchart of a Fuzzy Inference System

There are mainly two kinds of Fuzzy Inference Systems, Mamdani, and Sugeno. The two types of inference systems vary in the way outputs are determined.

Mamdani fuzzy inference systems obtain a set of fuzzy rules from human input where the output of each rule is a fuzzy logic set. The output of each rule is derived from the system's implication method and output membership function. Afterward, the output fuzzy set is then combined into a fuzzy set using the system's aggregation method. Finally, using the system's defuzzification method, the combined output fuzzy set is defuzzified, producing the final crisp output value.

While Sugeno fuzzy inference systems use a fuzzy singleton, a fuzzy set with membership functions that are either a linear or constant function of the input values. The system result produces a constant output membership function which corresponds to the centroid of Mamdani output membership function. Sugeno fuzzy rule has the form of if-then rule

$$if \ x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = f(x, y) \quad (2-14)$$

where A and B are fuzzy sets in the antecedent and $z = f(x, y)$ is a crisp function in the consequent.

2.4 Fuzzy Clustering

To determine fuzzy sets for each input variable, clustering techniques can be used in the fuzzification process. Clustering techniques are used to automatically generate fuzzy membership functions for the fuzzy sets using fuzzy clustering techniques that organize the training data into the appropriate fuzzy sets. Fuzzy clustering is different from hard clustering technique, where fuzzy clustering can have data points belonging to multiple clusters while hard clustering can only have data points belonging to exactly one cluster.

2.4.1 Modified Learning Vector Quantization (MLVQ)

Modified Learning Vector Quantization (MLVQ) was introduced to address the issue associated with the original Learning Vector Quantization (LVQ). [21] The issues include the selection of proper training parameters, initial weights, and forced termination. MLVQ has been proven that it is able to identify centroids of overlapping clusters and is able to ignore outliers without associating them as separate clusters regardless of the sequence of training data. MLVQ algorithm is able to produce the centroids of the fuzzy clusters by using an initial partitioning of weight vectors, as well as gradient descent termination. The algorithm requires three important parameters, number of clusters (c), learning constant (λ), and stopping criterion (ε). The MLVQ algorithm to generate Gaussian-shaped fuzzy membership function is as follows:

1. Initialize training iteration and weight

$$v_i^{(0)} = \min_k(x_k) + \frac{i+0.5}{c} (\max_k(x_k) - \min_k(x_k)) \text{ for } i = 1..c, k = 1..n \quad (2-15)$$

where

n refers to the number of data vectors

2. $for k = 1..n$

2.1. Find the winner i

$$|x_k - v_i^{(T+1)}| = \min_j (|x_k - v_j^{(T+1)}|) \text{ for } j = 1..c \quad (2-16)$$

2.2. Update the weight of the winner i

$$v_i^{(T+1)} = v_i^{(T)} + \lambda(x_k - v_i^{(T)}) \quad (2-17)$$

3. Compute $e^{(T+1)}$

$$e^{(T+1)} = \sum_{k=1}^n \min_j (|x_k - v_i^{(T+1)}|) \quad (2-18)$$

4. Compare $e^{(T+1)}$ and $e^{(T)}$

$$de^{(T+1)} = e^{(T+1)} - e^{(T)} \quad (2-19)$$

where

$$e^{(0)} = 0$$

5. If $|de^{(T+1)}| \leq \epsilon$ stop, else repeat step 3-5 for $T = T + 1$

6. Calculate the width of the Gaussian membership function for the given centroid

$$w_i = \frac{v_i - v_{closest}}{\sigma} \quad (2-20)$$

where

σ refers to the width constant

w_i refers to the width of the Gaussian membership function for centroid v_i

$v_{closest}$ refers to the centroid that is closest to v_i

7. Calculate the width of the Gaussian membership function for the given centroid

$$\mu_i(x) = e^{-0.5\left(\frac{x-v_i}{w_i}\right)^2} \quad (2-21)$$

where

$\mu_i(x)$ refers to the membership grade of x in cluster i

v_i refers to the centroid of cluster i

w_i refers to the width of the Gaussian membership function calculated in Equation

2-20

2.4.2 Pseudo Fuzzy Kohonen Partition (PFPK)

Pseudo Fuzzy Kohonen Partition (PFPK) [21] was created based on MLVQ algorithm, where it is able to directly create fitting membership functions from the training data. PFPK generates a triangular and trapezoidal-shaped membership function, which is highly utilized due to its simplicity in computation and storage. PFPK is an unsupervised algorithm that uses competitive learning to calculate the centroid and the pseudo weight. The algorithm requires four important parameters, number of clusters (c), learning constant (λ), learning width (n), and stopping criterion (ε). The PFPK algorithm is as follows:

1. Initialize training iteration and weight

$$v_i^{(0)} = \min_k(x_k) + \frac{i+0.5}{c} (\max_k(x_k) - \min_k(x_k)) \text{ for } i = 1..c, k = 1..n$$

where

n refers to the number of data vectors

2. Initialize $v_i^{(T+1)} = v_i^{(T)}$ for $i = 1..c$

3. for $k = 1..n$

3.1. Find the winner i

$$|x_k - v_i^{(T+1)}| = \min_j (|x_k - v_j^{(T+1)}|) \text{ for } j = 1..c$$

3.2. Update the weight of the winner

$$v_i^{(T+1)} = v_i^{(T)} + \lambda(x_k - v_i^{(T)})$$

4. Compute $e^{(T+1)}$

$$e^{(T+1)} = \sum_{k=1}^n \min_j (|x_k - v_i^{(T+1)}|)$$

5. Compare $e^{(T+1)}$ and $e^{(T)}$

$$de^{(T+1)} = e^{(T+1)} - e^{(T)}$$

where

$$e^{(0)} = 0$$

6. If $de^{(T+1)} \leq \epsilon$ stop, else repeat step 3-7 for $T = T + 1$

7. for all $i = 1..c$ initialize $\alpha_i = \beta_i = \delta_i = \gamma_i = \varphi_i = v_i^{(T+1)}$

8. for $k = 1..n$

8.1. Find winner i

$$|x_k - \varphi_i| = \min_j (|x_k - \varphi_j|) \text{ for } j = 1..c \quad (2-22)$$

8.2. Update pseudo weight of winner i

$$\varphi_i = \varphi_i + \eta(x_k + \varphi_i) \quad (2-23)$$

8.3. Update the four points of Trapezoidal Fuzzy Number

$$\alpha_i = \min(\alpha_i, x_k) \quad (2-24)$$

$$\beta_i = \min(\beta_i, \varphi_i)$$

$$\delta_i = \max(\delta_i, \varphi_i)$$

$$\gamma_i = \max(\gamma_i, x_k)$$

2.4.3 Unsupervised Discrete Clustering Technique (UDCT)

Unsupervised Discrete Clustering Technique (UDCT) creates fuzzy sets by determining their membership grade equate to a number of evenly distributed discrete points in the sample space.

Neural network architecture is used to execute data clustering for each variable. The neural network consists of three layers namely an input layer, centroid layer, and membership layer.

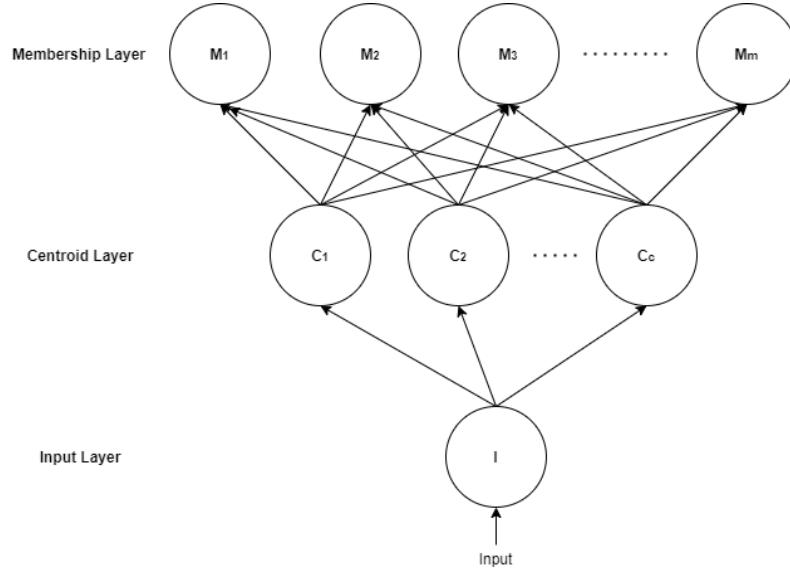


Figure 2.9 Architecture of Neural Network for Discrete Clustering Technique

From the diagram above, we can see that the input layer comprises a single neuron, a centroid layer with c neurons where it refers to the number of clusters, and the membership layer with m neurons which refers to the number of discrete points evenly spaced in the sample space. The name UDCT comes from the centroid layer and membership layer, which both are unsupervised. The algorithm requires four important parameters, number of clusters (c), learning constant (λ), number of fixed discrete points (m), and stopping criterion (ε). The UDCT algorithm is as follows:

1. Initialize training iteration and weight

$$v_i^{(0)} = \min_k(x_k) + \frac{i+0.5}{c} (\max_k(x_k) - \min_k(x_k)) \text{ for } i = 1..c, k = 1..n$$

where

n refers to the number of data vectors

2. Initialize $v_i^{(T+1)} = v_i^{(T)}$ for $i = 1..c$
3. for $k = 1..n$

- 3.1. Find the winner i

$$|x_k - v_i^{(T+1)}| = \min_j (|x_k - v_j^{(T+1)}|) \text{ for } j = 1..c$$

3.2. Update the weight of the winner

$$v_i^{(T+1)} = v_i^{(T)} + \lambda(x_k - v_i^{(T)})$$

4. Compute $e^{(T+1)}$

$$e^{(T+1)} = \sum_{k=1}^n \min_j (|x_k - v_i^{(T+1)}|)$$

where

x_k belongs to the i^{th} cluster

5. Compare $e^{(T+1)}$ and $e^{(T)}$

$$de^{(T+1)} = e^{(T+1)} - e^{(T)}$$

where

$$e^{(0)} = 0$$

6. If $|de^{(T+1)}| \leq \varepsilon$ stop, else repeat step 3-7 for $T = T + 1$

7. Compute the value of discrete points X_j

$$X_j = \min_k (x_k) + \frac{j+0.5}{m} (\max_k (x_k) - \min_k (x_k)) \quad (2-25)$$

8. Initialize the weights $\mu_i(X_j)$ to 0 for all $i = 1..c$ and all $j = 1..m$

9. Compute the spatial neighborhood for the Membership Layer

$$nb = \frac{n}{c \times 2} \quad (2-26)$$

10. Compute the learning rate for each individual cluster

$$\lambda_i = \frac{1}{\Omega_i} \text{ for all } i = 1..c \quad (2-27)$$

where

Ω_i refers to the number of data vectors that belong to cluster i

11. for $k = 1..n$

11.1. Find winner i

$$|x_k - v_i^{(T+1)}| = \min_j (|x_k - v_j^{(T+1)}|) \text{ for } j = 1..c$$

11.2. Find the winning neuron j in the Membership Layer

$$j = \arg(\min_l (|x_k - X_l|)) \text{ for all } l = 1..m \quad (2-28)$$

11.3. Compute the gravitational ratio g_{jk}

$$\sigma = \frac{\max_k(x_k) - \min_k(x_k)}{m} \text{ for all } k = 1..n$$

$$g_{jk} = \frac{|x_k - X_j|}{\sigma} \quad (2-29)$$

where

σ refers to the interval between any two adjacent discrete points

11.4. Update the weight $\mu_i(X_j)$

$$\mu_i(X_j) = \mu_i(X_j) + \lambda_i(1 - g_{jk}) \quad (2-30)$$

11.5. for $p = j - nb .. j + nb, p \neq j$

11.5.1. Compute the gravitational ratio g_{pk}

$$d = |j - p| \quad (2-31)$$

$$g_{pk} = \frac{|x_k - X_p|}{d \times \sigma}$$

where

d refers to the distance between neighbor p and the winning neuron j

11.5.2. Update the weight $\mu_i(X_p)$

$$\mu_i(X_p) = \mu_i(X_p) + \frac{\lambda_i}{d}(1 - g_{pk}) \quad (2-32)$$

12. Normalize all weights

$$\mu_i(X_j) = \frac{\mu_i(X_j)}{\max_p(\mu_i(X_p))} \quad (2-33)$$

for all $i = 1..c$ and $j = 1..m$

2.5 Fuzzy Neural Network

As mentioned before, neural networks have proven that it is indeed accurate even when tasked to predict highly volatile outputs, however, it does not provide any sort of reasoning behind the output.

Hence, fuzzy neural networks have been introduced as it provides interpretability and will be able to imitate human reasoning processes. [29], [30], [31], [32], [33], [34] There are mainly two kinds of fuzzy neural networks, cascading neural networks and parallel fuzzy neural networks. For cascading neural networks, a fuzzy layer can technically be implemented anywhere in the deep learning architecture, as long as it is in between the input and output layer. The fuzzy layer takes its input from the previous layer and passes its output to the next layer. [30], [31], [32], [33], [34] While for parallel networks, the fuzzy system can be activated simultaneously, along with the neural network model.

[29]

2.5.1 Parallel

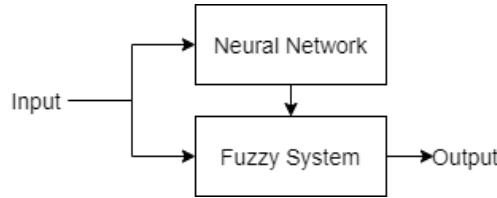


Figure 2.10 Parallel fuzzy neural network system - adapted from [14]

For parallel systems, the rules do not interact with each other. Rules applied are independent of the result from the other rules. Parallelization is introduced to handle large datasets where parallel classifiers are normally used so as to minimize the execution time and raise the learning rate. [15] For parallel classifiers, prior knowledge of the number of clusters is not required, however, correlations will not be uniformly distributed among data partitions. [16] Since the placement of the fuzzy layer can be anywhere as long as it is in between the input and output layer, the input of the parallel fuzzy system can be directly from the neural network or it can be from one of the hidden layers and the output of the parallel fuzzy system can be to the output layer of neural network or to one of the hidden layer.

Possible inputs	Possible outputs
-----------------	------------------

Input layer	Output layer
Hidden layer	Hidden layer
Fuzzy layer	Fuzzy layer

Table 2.3 Possible inputs and outputs of parallel fuzzy neural network

2.5.2 Cascading

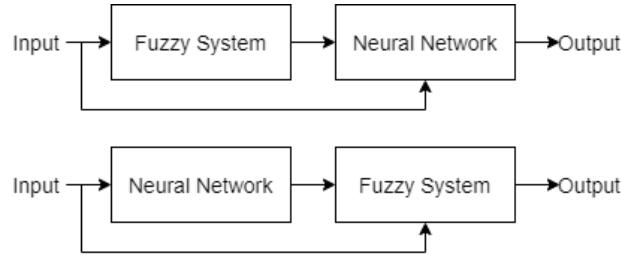


Figure 2.11 Cascading fuzzy neural network systems - adapted from [14]

For cascading systems, the input of the subsequent rule is determined by the previous rule, hence the rule applied is dependent on the results of the previous rule. This also shows that the sequence of the rule matters. Cascade fuzzy includes a sequence of various fuzzy rules in several stages, where the initial fuzzy rule provides input to the next fuzzy rule and until the last fuzzy rule to obtain the final result. [18], [19] Since the placement of the fuzzy layer can be anywhere as long as it is in between the input and output layer, the input of the series fuzzy system can be directly from the neural network or one of the hidden layers or the previous rule and the output of the parallel fuzzy system can be to the output layer of neural network or to one of the hidden layer or to the subsequent rule.

Possible inputs	Possible outputs
Input layer	Output layer

Hidden layer	Hidden layer
Fuzzy layer	Fuzzy layer
Previous rule	Subsequent rule

Table 2.4 Possible inputs and outputs of series fuzzy neural network

2.5.3 Yager Rule Network

Yager Rule Network is a four layer artificial neural network that computes fuzzy logic inferences, where each network structure executes a rule in the rule base in the form of:

If X_1 is A_1 and X_2 is A_2 and ... and X_n is A_n then Y is B

The network takes in fuzzy input and output fuzzy values as well. The Yager Inference Rule works by merging the functions performed at different layers to acquire the fuzzy output value. [36]

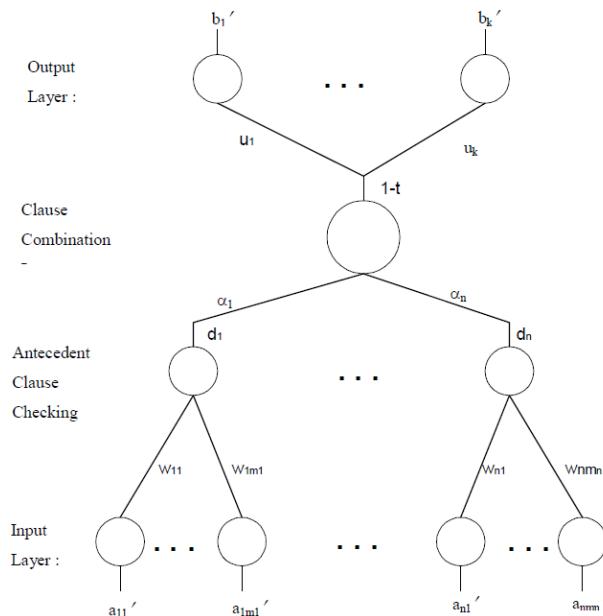


Figure 2.12 Structure of Fuzzy Logic Inference Neural Network - adapted from [36]

Layer 1: Input Layer

The input layer is a pseudo layer that analyses the input fuzzy sets and characterizes the possibility distribution of the facts.

Layer 2: Antecedent Clause Checking Layer

The antecedent clause checking layer assesses the disagreement between the fuzzy input possibility distribution and the antecedent clause distribution. Hence the neurons in this layer represent an antecedent clause of the rule.

Layer 3: Clause Combination Layer

This layer restricts the firing of the rules by computing the disagreement between the antecedent clauses and input fuzzy sets.

Layer 4: Output Layer

The weight on the output neurons conveys the information from the consequent of the rule. If the total disagreement equals to 0 then the rule is fired but if the total disagreement equals to 1 then the conclusion is unknown.

2.5.4 Yager Pattern Classifying Fuzzy Neural Network (Yager-PC FNN)

Yager Pattern Classifying Fuzzy Neural Network (Yager-PC FNN) proposed by Singh and Quek is a six-layer neural network that takes in multiple inputs and returns a single output. [35] Hence it is usually used for classification predictive modeling. Both input and output of the system are crisp values, as the fuzzification will be done at the input layer while the defuzzification is done at the output layer. The yager fuzzy inference will be performed simultaneously by the Input, Antecedent, Clause Combination, Output Possibility, and Deviation layer.

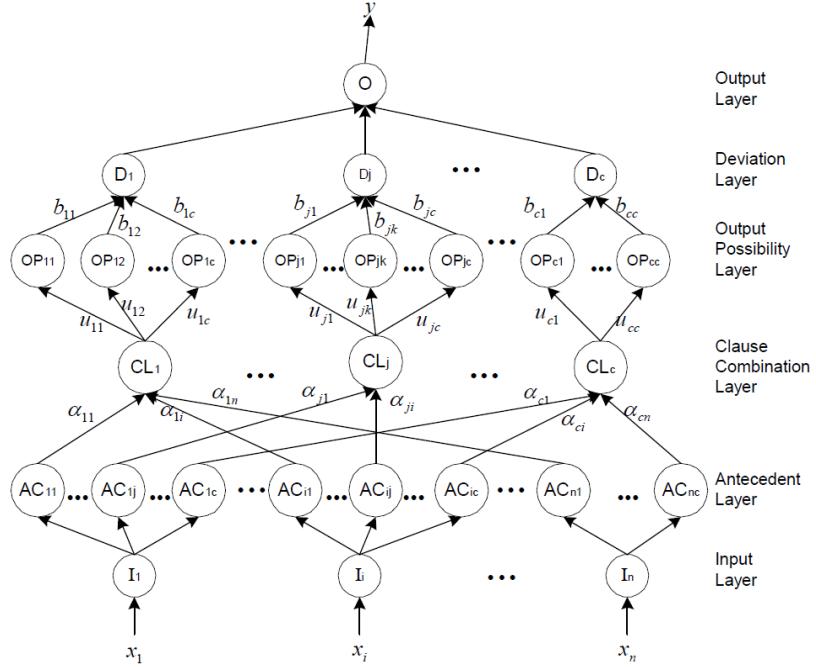


Figure 2.13 Structure of Yager-PC FNN - adapted from [35]

Layer 1: Input Layer

Input layer comprises of n neurons, where n refers to the number of features. Each feature is mapped to a neuron in the layer, where the neuron passes the crisp input value to the next layer.

Layer 2: Antecedent Layer

The antecedent layer comprises of $n \times c$ neurons, where c refers to the number of patterns. The antecedent layer accumulates the disagreement between the input and several antecedent clauses.

Layer 3: Clause Combination Layer

Clause combination layer comprises of c neurons, where each neuron denotes each fuzzy rule. This layer accumulates the disagreement between features and patterns where the output of each neuron is the complement of the net dissimilarity.

Layer 4: Output Possibility Layer

Output possibility layer comprises of $c \times c$ neurons, where each output class is linked with c neurons.

This layer computes the possibility distribution for each conclusion.

Layer 5: Deviation Layer

The deviation layer comprises of c neurons, where each denotes a pattern. This layer computes the deviation of the possibility distribution.

Layer 6: Output Layer

Output layer comprises a single neuron, where it analyses the deviations for the various conclusions and eventually chooses the pattern that deviated least from the rule consequent.

2.5.5 Modified Yager Fuzzy Neural Network (Modified Yager FNN)

Modified Yager Fuzzy Neural Network (Modified Yager FNN) proposed by Singh and Quek is a seven-layer neural network that takes in multiple inputs and returns multiple outputs. [35] Hence it is usually used for regression predictive modeling.

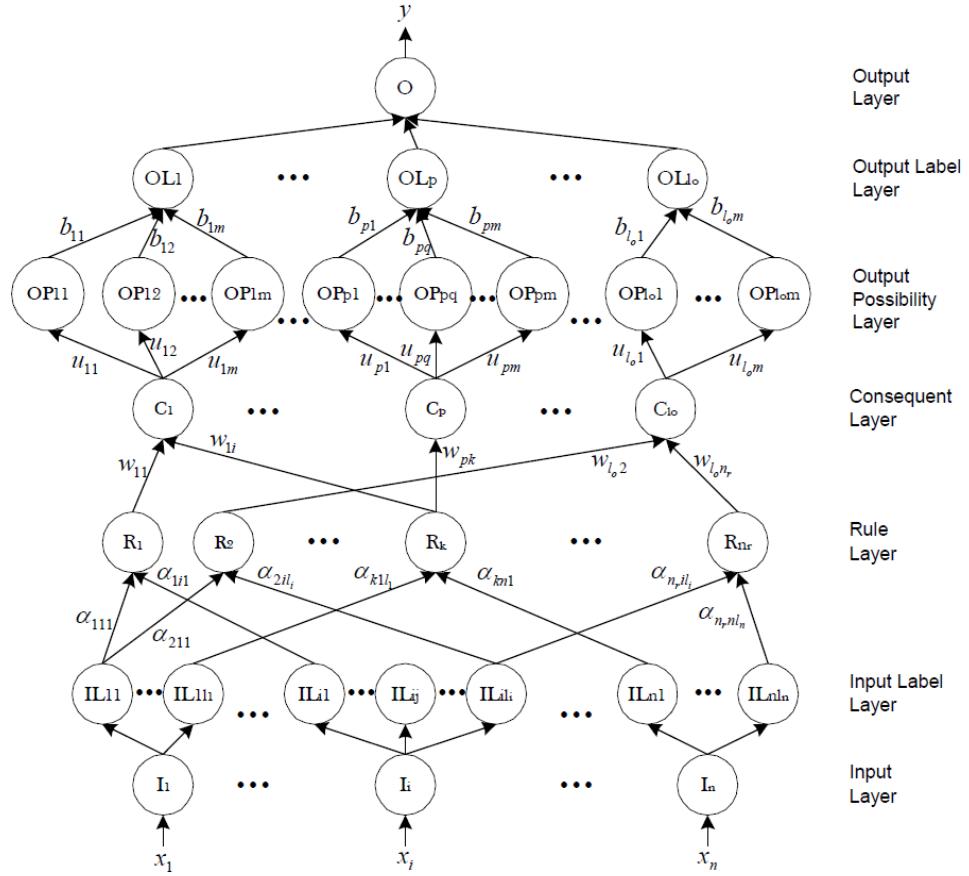


Figure 2.14 Structure of Modified Yager FNN - adapted from [35]

Layer 1: Input Layer

Input layer comprises of n neurons, where n refers to the number of features. Each feature is mapped to a neuron in the layer, where the neuron passes the crisp input value to the next layer.

Layer 2: Input Label Layer

The antecedent layer comprises of $n \times l_i$ neurons, where l_i refers to the number of output labels. The input label layer accumulates the disagreement between the input and several antecedent clauses.

Layer 3: Rule Layer

Rule layer comprises of n_r neurons, where n_r is the number of rules established by the network. Each neuron represents a fuzzy if-then rule, with various antecedent clauses and a consequent clause.

Layer 4: Consequent Layer

Each output variable is associated with consequent neurons equal to the number of output labels for the corresponding output variable, where each neuron calculates the weighted sum of all the rules that led to the consequent.

Layer 5: Output Possibility Layer

Output possibility layer comprises of $l_o \times m$ neurons, where m refers to the number of discrete points used and l_o refers to the number of output labels. This layer computes the possibility distribution for each conclusion.

Layer 6: Output Label Layer

Output label layer comprises of l_o neurons, where each denotes an output label. This layer computes two outputs where the first output computes the deviation of the possibility distribution and the second input stores the defuzzified fuzzy set.

Layer 7: Output Layer

The number of neurons in the output label depends on the number of output variables. The output neurons analyze the deviations for the various conclusions and eventually choose the output label that deviated least from the rule consequent.

2.6 Optimization

There are several factors that determine the effectiveness of the neural network, these include the number of hidden layers, the number of neurons within each hidden layer, activation function used, and optimizer used. For neural networks to perform well, the factors mentioned should be optimized using their respective methods.

2.6.1 Genetic Algorithm (GA)

Genetic algorithm is a search-based algorithm that offers a global and general optimization procedure. Even when it is analyzing a highly complex problem with a mandatory huge amount of hidden layers and neurons, it is quite fast. The algorithm mainly uses a global optimization algorithm to vary the number of neurons and hidden layers. Each variation of the network structure needs a full training of the current network variation. So if the optimal solution is included in the pool, genetic algorithms will be able to decide on the number of nodes and hidden layers required. [22]

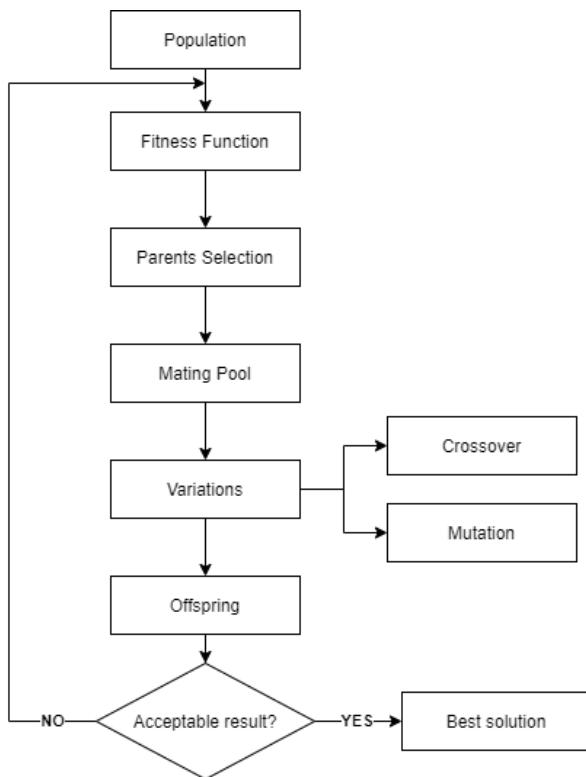


Figure 2.15 Flowchart of Genetic Algorithm

GA comprises three main stages; selection, crossover, and mutation. At the start, a pool of possible solutions is created, and where the solutions then go through selection, crossover, and mutation to create new children. This process is repeated until the best solution has been found. During the selection stage, it mainly selects the parents to create new children. The selection gives higher priority to fitter individuals as it is more likely to create superior offspring. While for the crossover stage, it

combines information from two different individuals to create new offspring. And during the mutation stage, it randomly alters the heredity to increase genetic diversity.

The drawback of GA includes it being computationally expensive, however, it is much more efficient than manual search, hence, GA is commonly used to optimize neural network architecture.

2.6.2 Activation functions

Activation functions are required for each layer of the neural network to determine the output, where it plots the resulting values between -1 to 1 or 0 to 1, depending on the activation functions. The activation function chosen has a huge impact on the performance and capability of the neural network. Hence, with the optimization of neural networks in mind, the activation function should be taken into consideration as well. Usually, all the hidden layers use the same activation function, while the output layer uses a different activation function which is determined by the type of prediction expected. The different activation functions include; ReLu, Sigmoid, Tanh, Softmax, Softplus, Softsign, SELU, Elu, and Exponential.

Hidden layers normally use a differentiable nonlinear activation function as it enables the model to learn more complex functions compared to a network that uses a linear activation function. The common activation functions used in hidden layers are ReLu, Sigmoid, and Tanh. Where ReLu is normally used in multilayer perceptron and convolutional neural networks while Sigmoid and Tanh are used in recurrent neural networks.

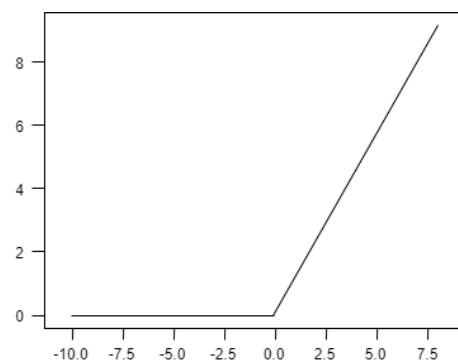


Figure 2.16 Graph of input vs output for a ReLu activation function

Relu is commonly used as it is easy to implement and less susceptible to vanishing gradients compared to Sigmoid and Tanh activation functions. However, a drawback of ReLu is that it can suffer from saturated units. The ReLu activation function is calculated as follows:

$$\max(0, x) \quad (2-34)$$

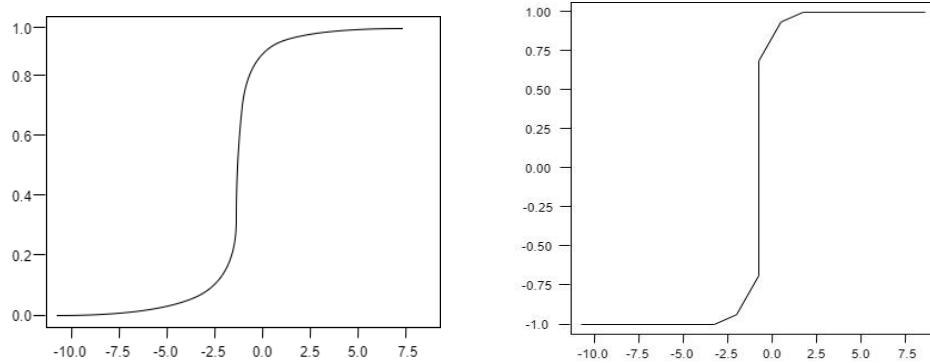


Figure 2.17 Graph of input vs output for a Sigmoid and a Tanh activation function

As mentioned before, Sigmoid and Tanh have a drawback called vanishing gradient. As shown above, the Sigmoid and Tanh activation function has a similar S shape. The Sigmoid activation function is calculated as follows:

$$1.0/(1.0 + e^{-x}) \quad (2-35)$$

The Tanh activation function is calculated as follows:

$$(e^x - e^{-x})/(e^x + e^{-x}) \quad (2-36)$$

2.6.3 Optimizers

The optimizers are used to adjust the parameters of a neural network to minimize the cost function.

The different activation functions include; Adam, Adadelta, Adagrad, Adamax, SGD, RMSprop, Nadam, Ftrl. Optimizers are accountable for reducing losses to provide the most accurate result possible.

Adam has been proved to be the best adaptive optimizer in most cases, it can be seen as a combination of Adagrad and RMSprop algorithms. [37] It uses the estimation of the first and second moments of the gradient to derive the learning rate for each weight of the neural network. It is formulated with the finest properties of AdaGrad and RMSProp, hence it will be able to handle noisy problems with sparse gradients.

2.7 Cross-validation (CV)

Cross-validation (CV) is a resampling procedure that is used to assess machine learning models with a limited data sample. It is used to assess the skill of a machine learning model on unobserved data, to predict how the model performs when it is used to make predictions on data that was not observed during the training of the model. Mainly, cross-validation is used to determine the generalization ability of a model.

K-fold cross-validation includes four steps;

1. Randomize the training data set
2. Split the dataset into k groups
3. For each group:
 - 3.1. It will be the test data set
 - 3.2. The remaining groups will be training data set
 - 3.3. Fit a model on the training set and assess it on the test set
 - 3.4. Record the evaluation score
4. Calculate the skill of the model using the sample of the evaluation scores

There are two types of error in cross-validation, ‘in-sample error’ which refers to training error, and ‘out-of-sample error’ which refers to validation partition error. In-sample error refers to the number of errors or mean squared error while out-of-sample error refers to the error rate from a new data set.

To choose the value of k, there are three strategies;

1. Representative: The strategy is to choose a certain value of k that make sure that each test group of data samples is sizeable enough to be statistically representative of the wider dataset
2. k=10/10-fold cv: The value of k is set to 10, which is a value that has been able to derive in a model skill estimate of low bias and a modest variance.
3. k=n/leave-one-out: The value of k is set to n, which n is the size of the dataset to ensure each test sample has an opportunity to be used in the test data set.

Leave-one-out is usually too expensive and a 10-fold CV is used instead, it is recommended to use averaging to get a better estimation of generalization error. (eg. 5 x 2-fold cv) The assumption is that test and training data should be representative samples of the underlying data distribution, which is often violated. Cross-validation chooses the validation segregation by randomizing the data, which gives a better evaluation of generalization error. It is used to find parameters that give good generalization and to compare the quality of models by assessing expected statistical accuracy. Hence, the final test should be performed on data that has not been used, for parameter adaptation and model selection.

2.8 Stock market

Stocks, in other words, shares of a company refers to ownership equity in the firm, which then gives shareholders voting rights and surplus claims on corporate earnings. The stock market allows individuals and institutional investors to buy and sell shares on a public platform. Share prices are determined by supply and demand which is affected when buyers and sellers place orders. It is proven that stock investment over a long period of time generates returns that surpass other asset classes. [prove] Investment returns refer to both dividends and capital gains. Capital gain is gained by selling a

stock at a higher price compared to the purchased price. While dividend is a slice of the company's earnings distributed to shareholders as an incentive to investors.

2.8.1 Trading Strategies

As mentioned above, one of the methods to earn returns is to sell stock at a higher price compared to the purchased price. Trading refers to the strategy of active participation that uses a buy-and-hold strategy. Traders use technical indicators to gain insight into the supply and demand of the market, which helps traders make decisions. Technical indicators use variables like trading volume and past prices to determine buy and sell signals.

There are mainly two types of indicators namely, leading and lagging indicators. Leading indicators give a trading signal before the new trend and reversal occur while lagging indicators give a trading signal after the trend has started. Past research confirms that using them separately was not successful in stock prediction. [23], [24], [25], [26], [27], [28] Leading indicators can be further broken down into two categories, momentum and volume. Momentum or oscillator indicators measure the relative strength of recent price moves while volume indicators measure the strength of a price move by using trading volume information. Lagging indicators can be further broken down into two categories as well, volatility and trend. Volatility indicators measure the rate of price changes regardless of their direction while trend indicators measure the strength and direction of a trend.

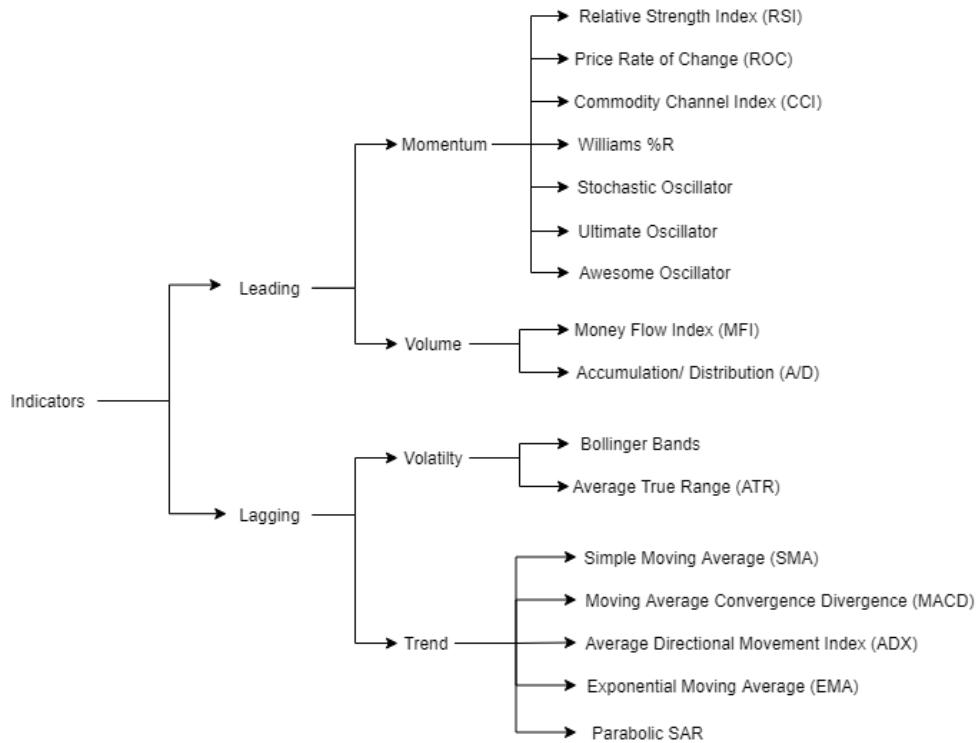


Figure 2.18 Technical indicators separated by categories

Systems that use a combination of lagging and leading technical indicators have proven that it is more effective rather than using the indicators individually. [23], [24], [25], [26], [27], [28]

2.8.2 Technical Indicators

Price Rate of Change (ROC) measures the strength of price momentum by the rate of change. It is used to spot divergences and confirm trends, ROC rising above zero confirms an uptrend while ROC falling below zero confirms a downtrend. The formula for ROC is as follows:

$$ROC = \left(\frac{closingprice_p - closingprice_{p-n}}{closingprice_{p-n}} \right) \times 100 \quad (2-37)$$

where

$closingprice_p$ refers to the closing price of the most recent period

closing price p_{-n} refers to closing price n periods before recent period

n refers to how many periods ago the current closing price is being compared to, which is typically 12, 25, or 200

Stochastic Oscillator is a popular momentum indicator that generates overbought and oversold signals. It relies on the price history of an asset, hence it tends to vary around the mean price level.

The formula for Stochastic Oscillator is as follows:

$$\%K = \left(\frac{C - L14}{H14 - L14} \right) \times 100 \quad (2-38)$$

where

C refers to the closing price of the most recent period

$L14$ refers to the lowest price traded of the 14 previous trading session

$H14$ refers to the highest price traded during the same 14 day period

$\%K$ refers to the current value of the stochastic indicator

Bollinger Bands generates overbought and oversold signals. It comprises three lines, a simple moving average, an upper and lower band. The upper and lower bands refer to the positive and negative standard deviation plotted as trendlines. The formula for Bollinger bands is as follows:

$$TP = (High + Low + Close) / 3 \quad (2-39)$$

$$Upper\ Bollinger\ Band = MA(TP, n) + m * \sigma[TP, n] \quad (2-40)$$

$$Lower\ Bollinger\ Band = MA(TP, n) - m * \sigma[TP, n] \quad (2-41)$$

where

MA refers to moving average

TP refers to typical price

n refers to the number of days in the smoothing period

m refers to the number of standard deviations

$\sigma[TP, n]$ refers to standard deviation over last n periods of TP

Relative Strength Index (RSI) is a momentum indicator that measures the magnitude of recent price changes, it evaluates overbought or oversold conditions in the price of a stock or other asset. RSI is displayed as an oscillator and can have a value between 0 and 100. The formula for RSI is as follows:

$$RSI = 100 - \left(\frac{100}{1 + \frac{\text{average gain}}{\text{average loss}}} \right) \quad (2-42)$$

where

average gain refers to average percentage gain during a look-back period

average loss refers to average loss during a look-back period

Commodity Channel Index (CCI) tracks market movement by comparing the current price with an average price over a specific period. The formula for CCI is as follows:

$$CCI = (\text{typical price} - \text{simple moving average}) / (0.015 \times \text{mean deviation}) \quad (2-43)$$

Simple Moving Average (SMA) calculates the dividing the prices over the selected period, traders usually use closing prices. The formula for SMA is as follows:

$$SMA = \frac{A_1 + A_2 + \dots + A_n}{n} \quad (2-44)$$

where

A_n refers to the price of the selected asset at period n

n refers to the number of days in a given period

Exponential Moving Average (EMA) tracks the price trends over selected time frames, it can be across 50 or even 200 days. Compared to SMA, EMA gives recent data more weight than older data. The formula to calculate multiplier for weighting EMA and current EMA is as follows:

$$k = \frac{2}{N + 1} \quad (2-45)$$

$$EMA = price(t) \times k + EMA(y) \times (1 - k) \quad (2-46)$$

where

k refers to the weighted multiplier

t refers to today

y refers to yesterday

N refers to the number of days in EMA

Moving Average Convergence Divergence (MACD) is calculated by subtracting the 26-period EMA from the 12-period EMA. MACD activates buy signals when it crosses above and sell signals when it falls below the signal line. The speed of crossovers is also used to determine if a market is overbought or oversold. MACD helps investors recognize if the bullish or bearish movement is strengthening or weakening. The formula for MACD is as follows:

$$MACD = 12 \text{ period EMA} - 26 \text{ period EMA} \quad (2-47)$$

where

N refers to the number of days in a given period

period sum refers to the sum of stock closing price during that period

3 Yager Pattern Classifying Fuzzy Neural Network (Yager-PC FNN)

In this chapter, we will explore the abilities of the different clustering techniques and eventually determine the best clustering technique used with Yager-PC FNN by calculating the recall ability of each. Yager-PC FNN was replicated to conduct this experiment, hence a comparison of the result will be conducted. [35] Since it is a comparison, the same dataset would be used, the original paper used Anderson's Iris Data. [38] Anderson's Iris dataset contains data from 3 different classes of iris subspecies (Sentosa, Versicolor, Virginica) with 150 vectors in total, 50 vectors from each class. The dataset contains 4 variables; sepal length, sepal width, petal length, and petal width.

3.1 Clustering on iris dataset

The clustering algorithms that will be applied are MLVQ, PFKP, UDCT respectively.

3.1.1 Results using MLVQ

During this experiment, the MVLQ algorithm was used on the whole set of 150 vectors. Since there is a vector of 50 for every cluster, the following variable was assigned the respective values:

- Learning constant $\lambda = 1/50 = 0.02$
- Number of cluster $c = 3$
- Width constant $\sigma = 1.5$
- Stopping criterion $\varepsilon = 0.0005$

The algorithm took an iteration of 12, 7, 8, and 12 iterations to cluster the four input variables respectively.

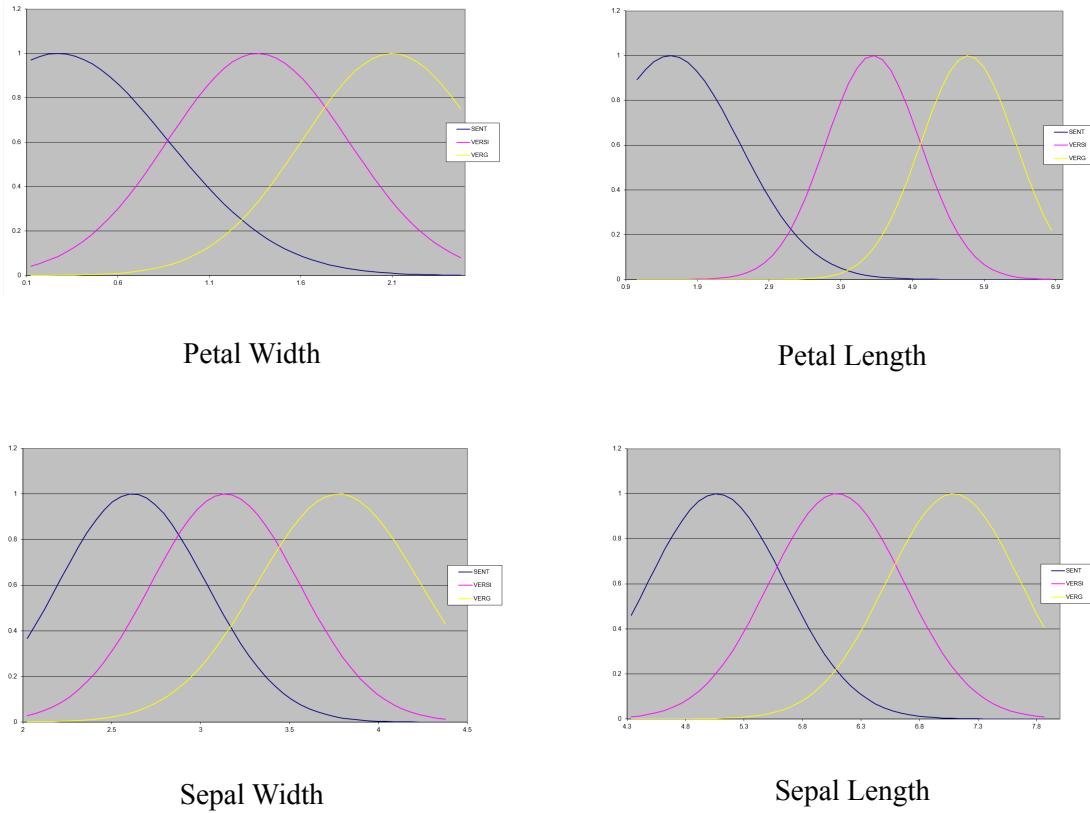


Figure 3.1 Membership Functions of Iris dataset identified by MLVQ

3.1.2 Results using PFKP

During this experiment, the PFKP algorithm was used on the whole set of 150 vectors. Since there is a vector of 50 for every cluster, the following variable was assigned the respective values:

- Learning constant $\lambda = 1/50 = 0.02$
- Number of cluster $c = 3$
- Learning width $\eta = 0$
- Stopping criterion $\varepsilon = 0.0005$

The algorithm took an iteration of 12, 7, 8, and 12 iterations to cluster the four input variables respectively.

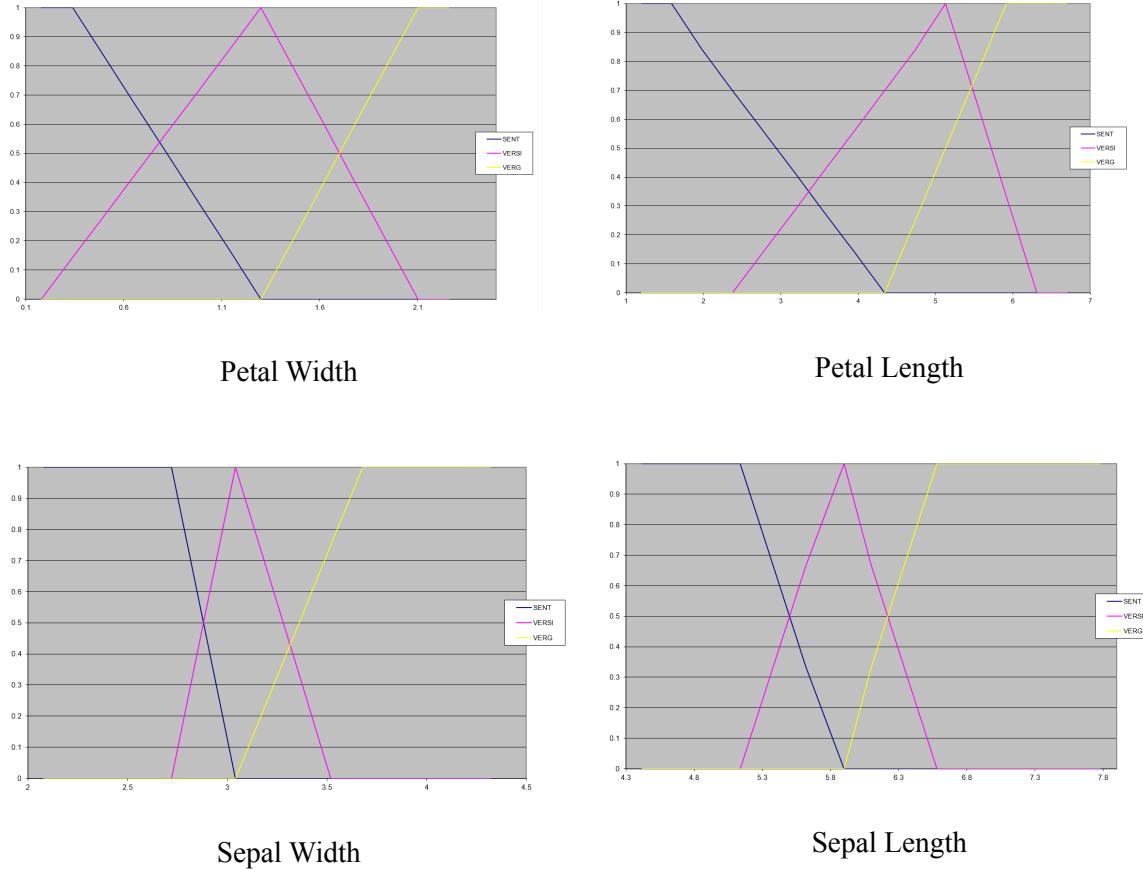


Figure 3.2 Membership Functions of Iris dataset identified by PFKP

3.1.3 Results using UDCT

During this experiment, the UDCT algorithm was used on the whole set of 150 vectors. Since there is a vector of 50 for every cluster, the following variable was assigned the respective values:

- Learning constant $\lambda = 1/150 = 0.0067$
- Number of cluster $c = 3$
- Discrete point sample point $m = 15$
- Stopping criterion $\varepsilon = 0.0005$

The algorithm took an iteration of 12, 7, 8, and 12 iterations to cluster the four input variables respectively.

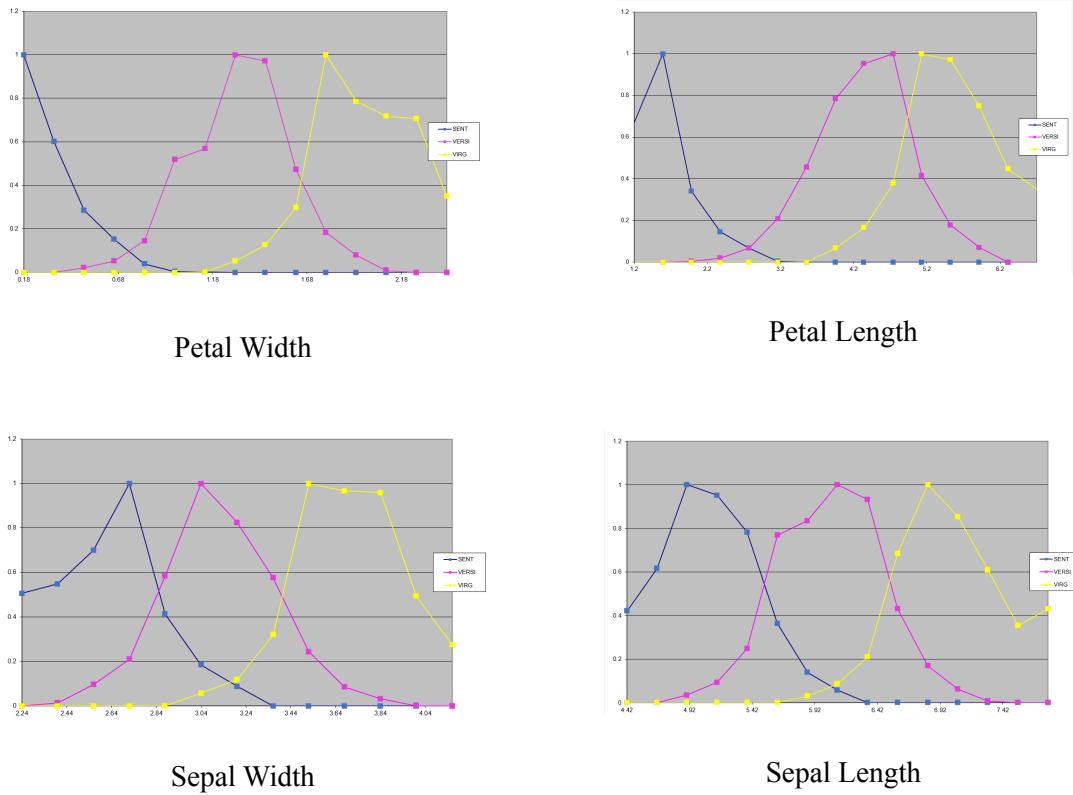


Figure 3.3 Membership Functions of Iris dataset identified by UDCT

3.1.4 Conclusion

It can be seen that all three different clustering techniques plot a very different shape which also means that the accuracy of each clustering technique might differ. In the next section, we would experiment with the different techniques using Yager-PC FNN to assess the accuracy of the respective techniques.

3.2 Yager-PC FNN with MLVQ, PFKP, and UDCT clustering technique

In this section, we would experiment with MLVQ, PFKP, and UDCT clustering techniques using Yager-PC FNN to assess the recall ability of the respective techniques. Since we will be assessing the recall ability of clustering techniques along with the FNN, we will be using 100% training data along with 100% testing data, which means that the same data would be used for both training and testing. To calculate the accuracy of the clustering technique, the following equation is used:

$$accuracy = \text{classified} / 150$$

Training Data	Testing Data	Clustering technique	Classified	Misclassified	Unclassified	Accuracy
100%	100%	PFKP	112	38	0	74%
100%	100%	MLVQ	132	18	0	88%
100%	100%	UDCT	144	6	0	96%

Table 3.1 Result of different clustering techniques used on Yager-PC FNN

With table 3.1, we can determine that UDCT has the best recall ability, while MLVQ and PFKP were not able to match up to UDCT.

4 Yager-PC FNN tagged with Multi-Layer Perceptron

There are various types of artificial neural networks that have been discovered, for a problem such as predicting Iris dataset, some have used Multi-Layer Perceptron to learn and accurately predict the different Iris species from their dimensions, one of the models even achieved 100% accuracy. [48], [49] With such high accuracy from their models, should we use the Multi-Layer Perceptron instead? As mentioned, the prediction results generated from artificial neural networks cannot be interpreted since it has a black box nature. To achieve interpretability, the process includes a data-driven entailment where tagging comes in place.

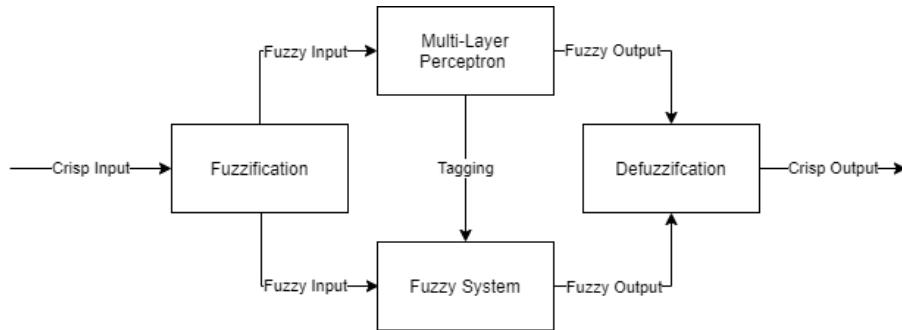


Figure 4.1 Architecture of Yager-PC FNN tagged with Multi-Layer Perceptron

Figure 4.1 shows a fuzzy system tagged with Multi-Layer Perceptron, that solves the two issues mentioned, to reach higher accuracy and to be able to interpret the decision making by the model. The architecture first takes in crisp values and converts them into fuzzy inputs through fuzzification, which is then passed to both the Multi-Layer Perceptron and Fuzzy System. Multi-Layer Perceptron extracts knowledge and learns to model the correlation. After training the Multi-Layer Perceptron, the nodes in Multi-Layer Perceptron will be associated with the nodes in the fuzzy system, which is achieved through tagging. Lastly, Multi-Layer Perceptron generates the fuzzy outputs that are converted to crisp output using defuzzification.

To give meaning to the results generated from the artificial neural network, it passes each prediction result to the fuzzy system to create a fuzzy inference rule, which is formulated by:

$$if x_1 \text{ is } A \wedge x_2 \text{ is } B \wedge \dots \wedge x_n \text{ is } N \text{ then } y \text{ is } Z \quad (4.1)$$

when Antecedents = $\{x_1, x_2, \dots, x_n\}$

Consequents = $\{y\}$

Humans require an explanation for each decision made, hence interpretability holds great significance.

Fuzzy systems help humans to understand the prediction made as the fuzzy sets are matched with human linguistic terms that humans can interpret meaning from, for example; hot, average, and cold. As mentioned before, the black-box nature of artificial neural networks leads people to have little to no trust with the prediction results despite the fact of its high accuracy that has been proven again and again. With artificial neural networks gaining popularity in various real-world applications, it is extremely important to understand the system especially for applications like autonomous vehicles, if there is a mistake or error, it could be life-threatening. However, with explanations for the prediction, developers will be able to adjust the learning environment to solve the error created.

In the first layer of Yager-PC FNN, called the input layer, fuzzify crisp input into a fuzzy singleton and output the fuzzy input to the next layer which is also passed to the Multi-Layer Perceptron simultaneously. To keep track of the firing strength of each rule node over time, Hebbian weight is updated when it is activated by the input.

4.1 Hebbian Weight Process

The Hebbian weight helps to keep track of the firing strength of each fuzzy rule node, where each node is linked to a Hebbian weight that will be updated during the training process. The formula for Hebbian Weight Process is as follows:

$$w_0 = 0 \quad (4-1)$$

$$w(t + 1) = w(t) + \Delta w \quad (4-2)$$

where $\Delta w = \{\mu(x_1) * \mu(x_2) * \dots * \mu(x_n)\} * \{\mu(y_1) * \mu(y_2) * \dots * \mu(y_p)\}$

The initial weight equals 0 and the weight increases if the data traverse through the fuzzy node.

Hence, the Hebbian weight with the highest value indicates the rule that has been fired the most.

4.2 Tagging Mechanism

When the fuzzified input data is passed to both Multi-Layer Perceptron and the fuzzy system, the Hebbian weight in the fuzzy rule node will be activated. The activated nodes in Multi-Layer Perceptron will be hence tagged with a fuzzy rule node. A Multi-Layer Perceptron node is not limited to one tag and it is possible to have multiple tags.

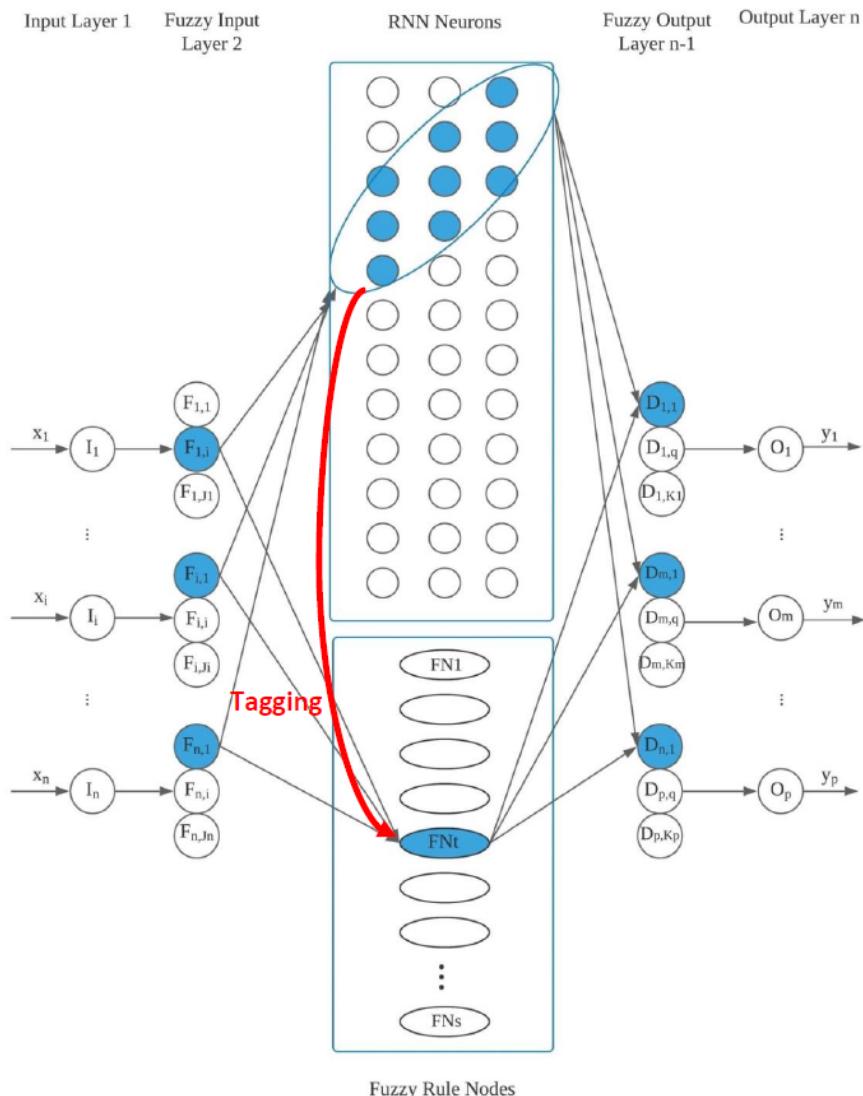


Figure 4.2 Tagging Mechanism - adapted from [47]

4.3 Ability of the different optimizers

To fairly assess the performance of the MLP, optimizers will be tested so that the leading optimizer will be used in the architecture. The optimizer used in any neural network affects the performance and accuracy of any prediction model. The experiment is done on both 100% training and testing data with elu activation function within the hidden layer, while the neural network will consist of a single hidden layer with 500 nodes.

Optimizer	SGD	RMSprop	Adam	Adadelta	Adagrad	Adamax	Nadam	Ftrl
Accuracy	96	98	98	60.6	96	96	98	33.3

Table 4.1 Result of the different optimizers

From table 4.1, we are able to see that RMSprop, Adam, and Nadam tied with an accuracy of 98%. For subsequent MLP architecture mentioned, Adam optimizer will be used.

4.4 Ability of different activation functions

To fairly assess the performance of the MLP, the activation function for the hidden layer(s) will be tested so that the leading activation function will be used in the architecture. The activation used in any neural network affects the performance and accuracy of any prediction model as well. The experiment will be done on both 100% training and testing data with the Adam optimizer function. The only activation function in the hidden layer will be tested, while the input layer uses relu and the output layer uses softmax activation function. Different layers and nodes will be used for this experiment as well, however, the total amount of hidden nodes used will be 500 since we are just experimenting with different kinds of structures possible instead of the amount of total hidden nodes.

Layers	Nodes	relu	sigmoid	softmax	softplus	softsign	tanh	selu	elu	exponential
3 layers	100-300-100	94.6	33.3	33.3	33.3	95.3	96.0	95.3	95.3	64.0

	100-150-250	90.6	33.3	33.3	33.3	95.3	95.3	95.3	93.3	33.3
	200-100-200	94.6	41.3	33.3	36.6	95.3	95.3	96.0	95.3	33.3
2 layers	200-300	95.3	33.3	33.3	33.3	95.3	95.3	95.3	95.3	80.0
	300-200	95.3	33.3	33.3	33.3	95.3	96.0	95.3	95.3	33.3
	250-250	94.0	33.3	33.3	33.3	94.0	95.3	95.3	96.0	70.0
1 layer	500	96.0	33.3	33.3	33.3	94.6	95.3	96.0	98.0	36.0

Table 4.2 Result of the different activation functions along with different layers and nodes

From table 4.2, we are able to see that MLP with 1 hidden layer consisting of 500 nodes along with an elu activation function gave the best accuracy. Hence, for subsequent MLP architecture mentioned, it will use 1 hidden layer consisting of 500 nodes along with an elu activation function.

4.5 Yager-PC FNN versus Yager-PC FNN tagged with Multi-Layer Perceptron

To fully understand the ability of Yager-PC FNN tagged with Multi-Layer Perceptron, it will be benchmarked against Yager-PC FNN. Both will be using UDCT as the clustering techniques, with the following variables:

- Learning constant = $1/150 = 0.0067$
- Number of cluster c = 3
- Discrete point sample point m = 15
- Stopping criterion = 0.0005

Multi-Layer Perceptron used in Yager-PC FNN tagged with Multi-Layer Perceptron will have the following variables:

- Hidden layer = 1 with 500 nodes
- Activation function for input layer = relu
- Activation function output layer = softmax
- Activation function for hidden layer = elu
- Optimizer = Adam

To fully test the ability of Yager-PC FNN tagged with Multi-Layer Perceptron against Yager-PC FNN, a recall ability test, and a generalization ability test will be conducted. A recall ability test uses the exact data for training and testing, 100% and 50% training and testing data would be used. While for generalization ability tests, different slices of the data would be used for training and testing data. A cross-fold of 2 and 5 would be used in this experiment.

	Training data	Testing data	Accuracy	
			Yager-PC FNN	Yager-PC FNN tagged with Multi-Layer Perceptron
Recall ability	100%	100%	0.960	0.980
	Fold 1	Fold 1	0.960	0.986
	Fold 2	Fold 2	0.960	0.986
Generalization ability	Fold 1	Fold 2	0.946	0.960
	Fold 2	Fold 1	0.946	0.960
	Cross-Validation (2 folds)		0.946	0.960
	20% (Fold 1)	80%	0.966	0.950
	20% (Fold 2)	80%	0.833	0.941
	20% (Fold 3)	80%	0.833	0.866
	20% (Fold 4)	80%	0.833	0.833
	20% (Fold 5)	80%	0.933	0.933
	Cross-Validation (5 folds)		0.879	0.904

Table 4.3 Result of Yager-PC FNN versus Yager-PC FNN tagged with Multi-Layer Perceptron

In Table 4.3, we are able to see that overall Yager-PC FNN tagged with Multi-Layer Perceptron has slightly better accuracy compared to Yager-PC. For recall ability, Yager-PC FNN tagged with Multi-Layer Perceptron has a better result for all three tests. While for generalization ability, Yager-PC FNN did have better results during some folds, however on average, Yager-PC FNN tagged with Multi-Layer Perceptron did better.

5 Modified Yager FNN

In this chapter, we will explore the abilities of the different clustering techniques and eventually determine the best clustering technique used with Modified Yager FNN by calculating the recall ability of each. Modified Yager FNN was replicated to conduct this experiment, hence a comparison of the result will be conducted. [35] Since it is a comparison, the same dataset would be used, the original paper used traffic flow data. [45] The data used were collected by using an inductive loop installed by the Land Transport Authority of Singapore (LTA) at exit 15 which is located along the east-bound Pan Island Expressway (PIE). The traffic flow data consist of the traffic density of the five lanes; two exit lanes, and three straight lanes towards the main traffic. For this experiment, only the traffic flow data for the three straight lanes were extracted, and the data were preprocessed to show traffic flow density of 5-minute intervals. There are a total of five variables in this dataset, namely; time, lane 1, lane 2, lane 3, and an output. Where the output, refers to the actual value of lane 1 in $t + 5$.

5.1 Clustering on traffic flow dataset

The clustering algorithms that will be applied are MLVQ, PFKP, UDCT respectively.

5.1.1 Results using MLVQ

During this experiment, the MLVQ algorithm was used on the whole set of data. The following variable was assigned the respective values:

- Learning constant $\lambda = 0.0018$
- Number of cluster $c = 5$
- Width constant $\sigma = 1.5$
- Stopping criterion $\varepsilon = 0.0005$

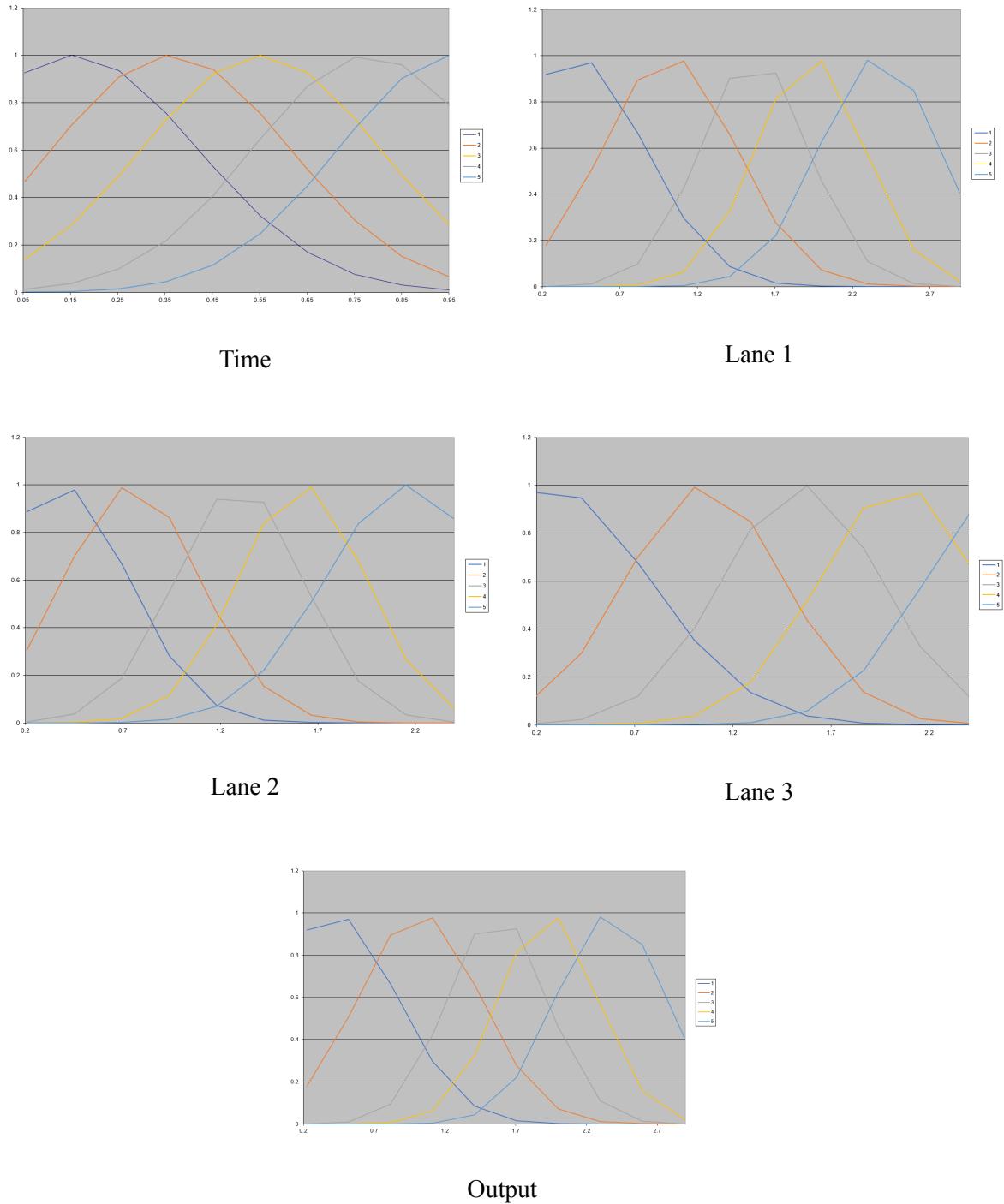


Figure 5.1 Membership Functions of Traffic flow dataset identified by MLVQ

5.1.2 Results using PFKP

During this experiment, the PFKP algorithm was used on the whole set of data. The following variable was assigned the respective values:

- Learning constant $\lambda = 0.0018$

- Number of cluster 5
- Learning width $\eta = 0.2$
- Stopping criterion $\epsilon = 0.0005$

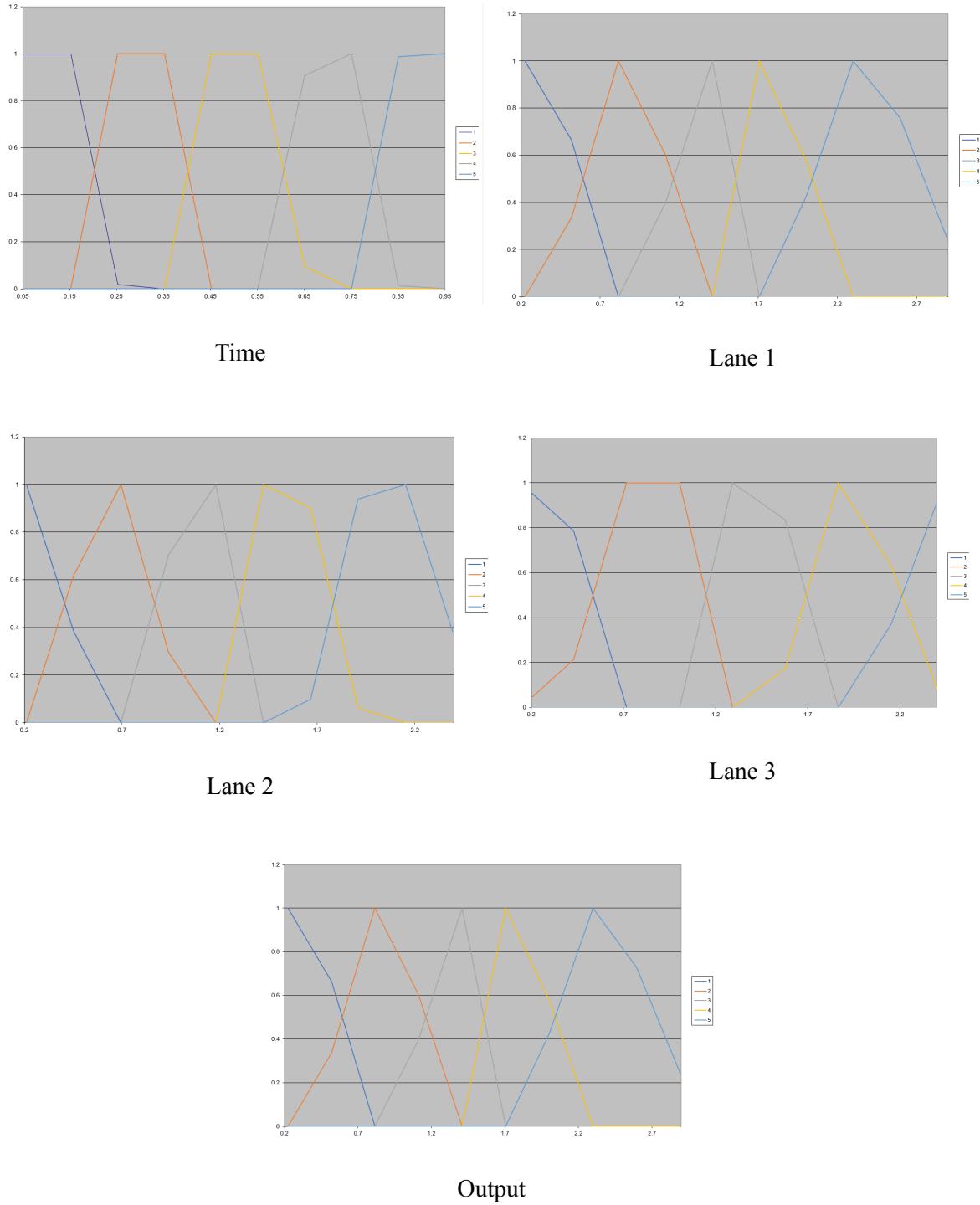
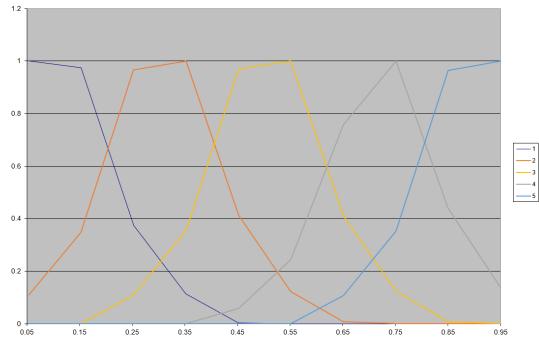


Figure 5.2 Membership Functions of Traffic flow dataset identified by PFKP

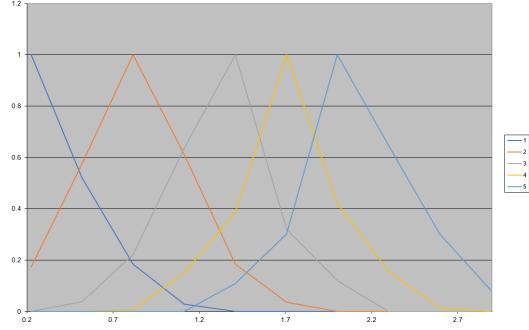
5.1.3 Results using UDCT

During this experiment, the UDCT algorithm was used on the whole set of data. The following variable was assigned the respective values:

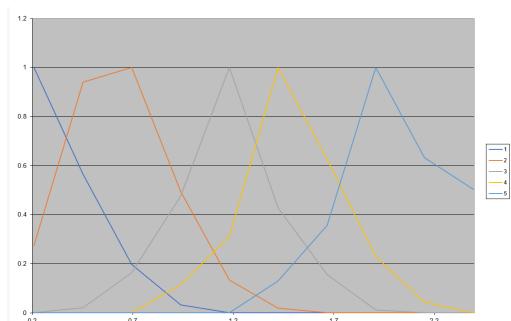
- Learning constant $\lambda = 0.0018$
- Number of cluster 5
- Discrete point sample point $m = 15$
- Stopping criterion $\varepsilon = 0.0005$



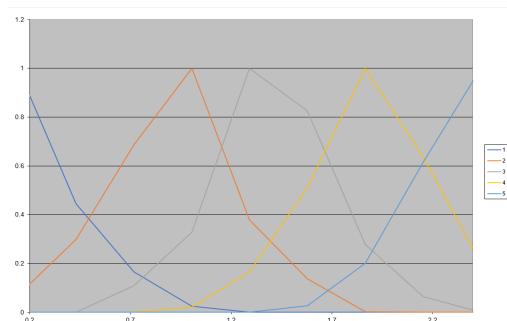
Time



Lane 1



Lane 2



Lane 3

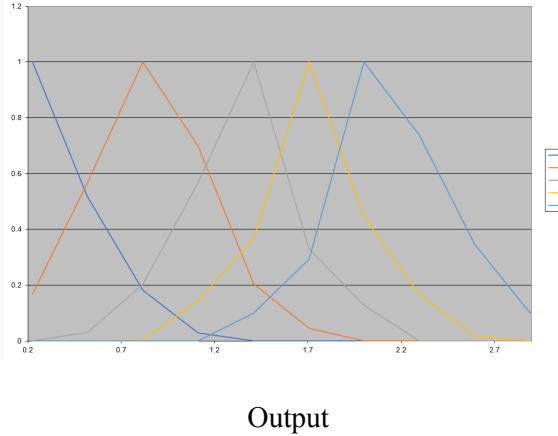


Figure 5.3 Membership Functions of Traffic flow dataset identified by UDCT

5.1.4 Conclusion

It can be seen that all three different clustering techniques plot a very different shape which also means that the accuracy of each clustering technique might differ. In the next section, we would experiment with the different techniques using Modified Yager FNN to assess the accuracy of the respective techniques.

5.2 Modified Yager FNN with MLVQ, PFKP, and UDCT clustering technique

In this section, we would experiment with MLVQ, PFKP, and UDCT clustering techniques using Modified Yager FNN to assess the recall ability of the respective techniques. Since we will be assessing the recall ability of clustering techniques along with the FNN, we will be using 100% training data along with 100% testing data, which means that the same data would be used for both training and testing. To calculate the accuracy of the clustering technique, the following equation is used:

$$\text{accuracy} = \text{classified} / 150$$

Training Data	Testing Data	Clustering technique	RMSE	R2
100%	100%	PFKP	0.054	0.844
100%	100%	MLVQ	0.055	0.866
100%	100%	UDCT	0.052	0.887

Table 5.1 Results of different clustering techniques on Modified Yager FNN

With table 5.1, we can determine that UDCT has the best recall ability, while MLVQ and PFKP were lagging slightly behind UDCT.

6 Modified Yager FNN tagged with Multi-Layer Perceptron

In this chapter, similar architecture from chapter 4 will be used, however, the fuzzy system will be replaced with Modified Yager FNN. The data used will be the same traffic data as the previous chapter.

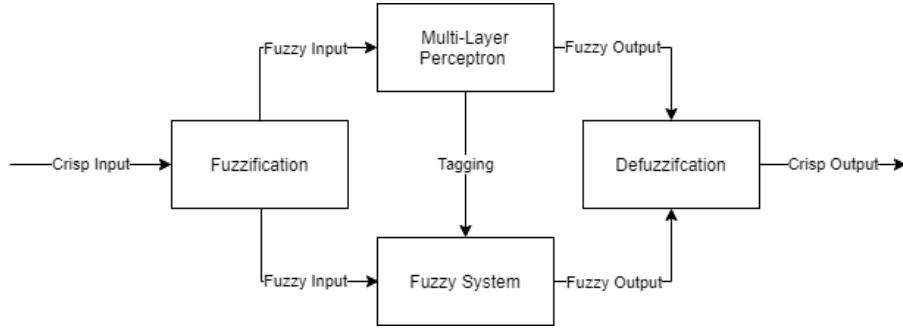


Figure 6.1 Architecture of Modified Yager FNN tagged with Multi-Layer Perceptron

6.1 Parameters used for Multi-Layer Perceptron (MLP)

The clustering technique used will be UDCT as well, as seen from the previous experiment, it managed to give the best recall accuracy. The activation function for the input layer, hidden layers, and output layer will be set to relu, elu, and softmax respectively. The optimizer used will be Adam as experimented from the iris dataset. The number of hidden layers used will be set to three while the number of neurons will be optimized using a Genetic Algorithm (GA). The minimum and maximum number of nodes for the hidden layer is one to two hundred respectively. The goal of the GA is to get the number of nodes that have the highest R^2 value.

geneticalgorithm is a python library that can be used to implement GA, it is able to solve optimization problems with continuous, discrete, and even mixed variables. Diagram - shows the flowchart of the ga function provided in the *geneticalgorithm* library.

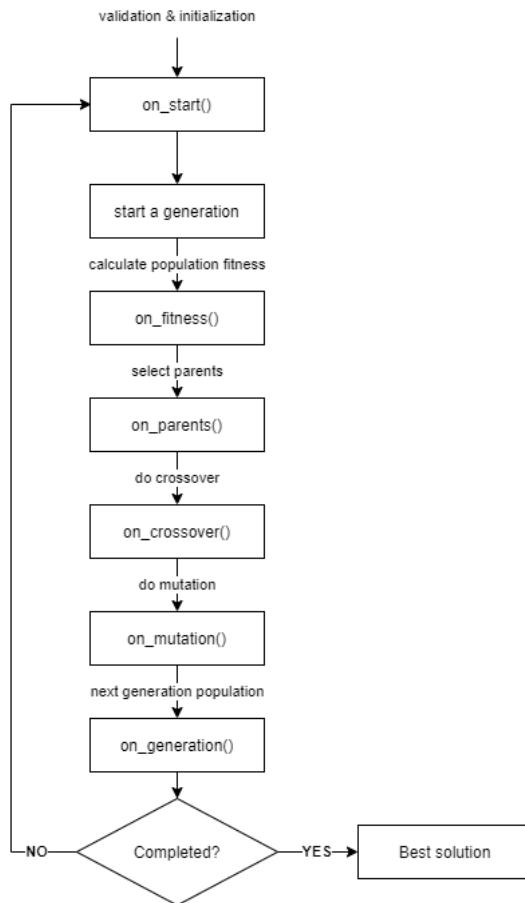


Figure 6.2 flowchart of geneticalgorithm from python library - adapted from [46]

6.2 Modified Yager FNN tagged with Multi-Layer Perceptron vs Modified Yager FNN

In this section, the result of Modified Yager FNN tagged with Multi-Layer Perceptron will be compared with the result of Modified Yager FNN. In addition, various clustering techniques will be covered as well. Lastly, the number of nodes for each hidden layer has been optimized using a genetic algorithm.

Hence the comparison will be between

- Modified Yager Network with UDCT clustering
- Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) with UDCT clustering, nodes not optimized

- Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) with UDCT clustering, nodes optimized
- Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) with MLVQ clustering, nodes optimized
- Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) with PFKP clustering, nodes optimized

There are a total of 6 folds, but a 5-fold CV is being used, while the 6th fold is used as verification.

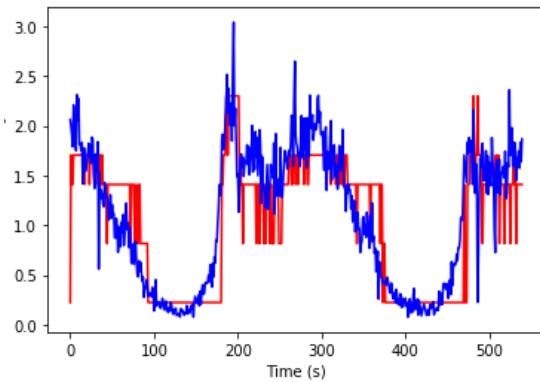


Figure 6.3 Result of Traffic data (fold 1) tested with Yager Network (UDCT)

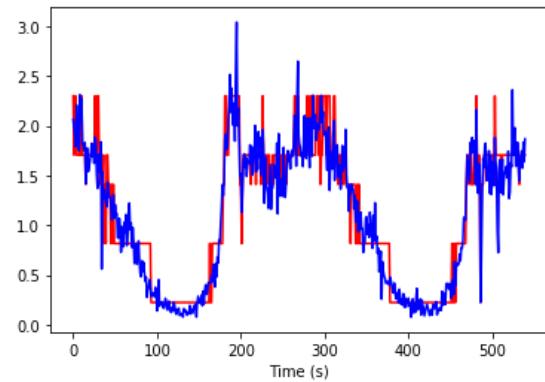


Figure 6.4 Result of Traffic data (fold 1) tested with Fuzzy-MLP (UDCT & optimized node)

Training data	Testing data	Yager Network (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (MLVQ)		Fuzzy-MLP (PFKP)	
				100-100-100		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
fold 1	fold 1	0.1596	0.6945	0.0707	0.8485	0.0531	0.8861	0.0606	0.8682	0.0550	0.8821
fold 2	fold 2	0.2673	0.5872	0.1002	0.7701	0.1257	0.7117	0.0998	0.7710	0.1169	0.7320
fold 3	fold 3	0.0959	0.6640	0.0469	0.8354	0.0356	0.8751	0.0703	0.7536	0.0471	0.8350
fold 4	fold 4	0.1910	0.6480	0.1073	0.8022	0.1017	0.8125	0.0764	0.8592	0.1026	0.8108
fold 5	fold 5	0.2537	0.5133	0.1202	0.7694	0.1275	0.7553	0.1105	0.7879	0.1202	0.7695
fold 6	fold 6	0.0525	0.8870	0.0602	0.8524	0.0606	0.8515	0.0444	0.8913	0.0741	0.8185

Table 6.1 Result of the recall ability

Training data	Testing data	Yager Network (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (MLVQ)		Fuzzy-MLP (PFKP)	
				100-100-100		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
fold 1	fold 1	0.1596	0.6945	0.0707	0.8485	0.0531	0.8861	0.0606	0.8682	0.0550	0.8821
fold 2	fold 2	0.2673	0.5872	0.1002	0.7701	0.1257	0.7117	0.0998	0.7710	0.1169	0.7320
fold 3	fold 3	0.0959	0.6640	0.0469	0.8354	0.0356	0.8751	0.0703	0.7536	0.0471	0.8350
fold 4	fold 4	0.1910	0.6480	0.1073	0.8022	0.1017	0.8125	0.0764	0.8592	0.1026	0.8108
fold 5	fold 5	0.2537	0.5133	0.1202	0.7694	0.1275	0.7553	0.1105	0.7879	0.1202	0.7695
fold 6	fold 6	0.0525	0.8870	0.0602	0.8524	0.0606	0.8515	0.0444	0.8913	0.0741	0.8185

20% (fold 1)	80%	0.2363	0.4879	0.1719	0.6275	0.1513	0.6720	0.1676	0.6367	0.1825	0.6046
20% (fold 2)	80%	0.1848	0.6078	0.1187	0.7481	0.1209	0.7435	0.1327	0.7185	0.1577	0.6654
20% (fold 3)	80%	0.2765	0.5388	0.1350	0.7270	0.1404	0.7161	0.1378	0.7213	0.1675	0.6612
20% (fold 4)	80%	0.2442	0.4357	0.1514	0.6500	0.1696	0.6079	0.1121	0.7392	0.1263	0.7080
20% (fold 5)	80%	0.2071	0.5425	0.1781	0.6065	0.1773	0.6082	0.1248	0.7242	0.2015	0.5547
16.6% (fold 6)	83.4%	0.2454	0.4705	0.1336	0.7117	0.1141	0.7479	0.1021	0.7795	0.1416	0.6944

Table 6.2 Result of the generalization ability (20% training data and 80% testing data)

Training data	Testing data	Yager Network (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)		Fuzzy-MLP (MLVQ)		Fuzzy-MLP (PFKP)	
				100-100-100		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
80%	20% (fold 1)	0.2191	0.5307	0.0790	0.8307	0.0757	0.8377	0.1513	0.7900	0.1243	0.7337
80%	20% (fold 2)	0.1793	0.5888	0.1109	0.7456	0.1053	0.7585	0.1521	0.7803	0.1186	0.7279
80%	20% (fold 3)	0.2092	0.5675	0.0810	0.7164	0.0738	0.7414	0.1484	0.7304	0.0851	0.7019
80%	20% (fold 4)	0.2480	0.5587	0.1149	0.7883	0.1102	0.7969	0.1804	0.7518	0.1282	0.7637
80%	20% (fold 5)	0.2800	0.4630	0.1253	0.7596	0.1206	0.7687	0.1587	0.7873	0.1441	0.7236
83.4%	16.6% (fold 6)	0.1697	0.5845	0.1065	0.7393	0.0729	0.8214	0.1400	0.8019	0.0532	0.8696

Table 6.3 Result of the generalization ability (80% training data and 20% testing data)

From tables 6.1, 6.2, and 6.3, it can be seen those with optimized nodes are performing a little better than those that are not, for both recall ability and generalization ability test. On average, the UDCT clustering technique is performing better. Overall, it can be seen that Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) is able to predict better than Modified Yager FNN, with an average of 20% increase of R^2 . For all architecture, it can be seen that accuracy is much better when there is more training data than testing data, which has been mentioned in the literature review that neural networks do better when trained with more amount of data.

6.3 Modified Yager FNN tagged with Multi-Layer Perceptron vs Pure Multi-Layer Perceptron

Now, it has been proven that Modified Yager FNN tagged with Multi-Layer Perceptron (Fuzzy-MLP) is able to predict well with interpretation, but how much accuracy is lost when using fuzzy values instead of crisp values? Hence this section presents the results between pure MLP versus Fuzzy-MLP.

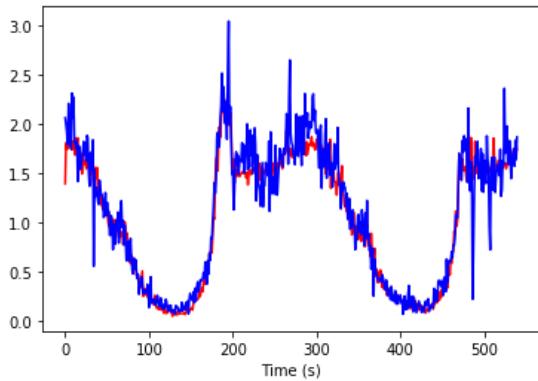


Figure 6.5 Result of Traffic data (fold 1) tested with Pure MLP & optimized node

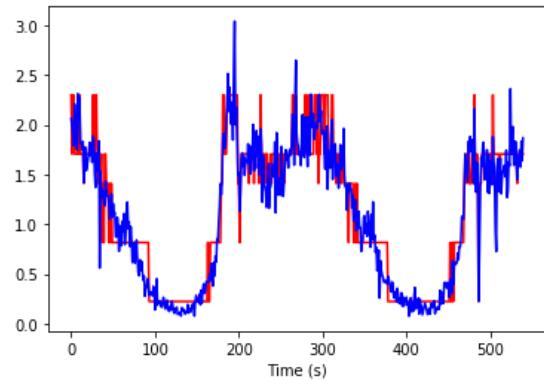


Figure 6.6 Result of Traffic data (fold 1) tested with Fuzzy-MLP (UDCT & optimized node)

Training data	Testing data	Pure MLP		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)	
		Optimized for fold 1 173-187-18		100-100-100		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2
fold 1	fold 1	0.0402	0.9138	0.0707	0.8485	0.0757	0.8377
fold 2	fold 2	0.0802	0.8160	0.1002	0.7701	0.1053	0.7585
fold 3	fold 3	0.0343	0.8799	0.0469	0.8354	0.0738	0.7414
fold 4	fold 4	0.0863	0.8409	0.1073	0.8022	0.1102	0.7969
fold 5	fold 5	0.093	0.8215	0.1202	0.7694	0.1206	0.7687
fold 6	fold 6	0.0426	0.8956	0.0602	0.8524	0.0729	0.8214

Table 6.4 Result of recall ability

Training data	Testing data	Pure MLP		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)	
		Optimized for fold 1 173-187-18		100-100-100		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2
20% (fold 1)	80%	0.1226	0.7343	0.1719	0.6275	0.1513	0.6720

20% (fold 2)	80%	0.0808	0.8285	0.1187	0.7481	0.1209	0.7435
20% (fold 3)	80%	0.1272	0.7427	0.1350	0.7270	0.1404	0.7161
20% (fold 4)	80%	0.0994	0.7701	0.1514	0.6500	0.1696	0.6079
20% (fold 5)	80%	0.0769	0.8301	0.1781	0.6065	0.1773	0.6082
16.6% (fold 6)	83.4%	0.0751	0.8341	0.1336	0.7117	0.1141	0.7479

Table 6.5 Result of the generalization ability (20% training data and 80% testing data)

Training data	Testing data	Pure MLP		Fuzzy-MLP (UDCT)		Fuzzy-MLP (UDCT)	
		Optimized for fold 1 173-187-18		100-100-100		Optimized for fold 1 173-187-18	
		RMSE	R2	RMSE	R2	RMSE	R2
80%	20% (fold 1)	0.0545	0.8832	0.0790	0.8307	0.0757	0.8377
80%	20% (fold 2)	0.0740	0.8302	0.1109	0.7456	0.1053	0.7585
80%	20% (fold 3)	0.0396	0.8613	0.0810	0.7164	0.0738	0.7414
80%	20% (fold 4)	0.0873	0.8390	0.1149	0.7883	0.1102	0.7969
80%	20% (fold 5)	0.0938	0.8200	0.1253	0.7596	0.1206	0.7687
83.4%	16.6% (fold 6)	0.0424	0.8960	0.1065	0.7393	0.0729	0.8214

Table 6.6 Result of the generalization ability (80% training data and 20% testing data)

From Table 6.4, 6.5, and 6.6, we are able to see that about an average of 8% of accuracy is being lost by using fuzzy values instead of crisp values. Even though there is accuracy lost over all the tests conducted, it is considered an insignificant tradeoff for the interpretability gained.

7 Modified Yager-FNN tagged with Long Short-Term Memory

The data used would be the same traffic data as the previous chapter, as we will compare the performance of the two architectures used.

7.1 Recurrent Neuro-Fuzzy Parallel System (RNFPS)

In this chapter, similar architecture from chapter 4 will be used, however, the fuzzy system will be replaced with Modified Yager FNN and the Multi-Layer Perceptron (MLP) will be replaced with Long Short-Term Memory (LSTM). Modified Yager-FNN tagged with Long Short-Term Memory will be named as Recurrent Neuro-Fuzzy Parallel System (RNFPS) as it will be used in later chapters as well.

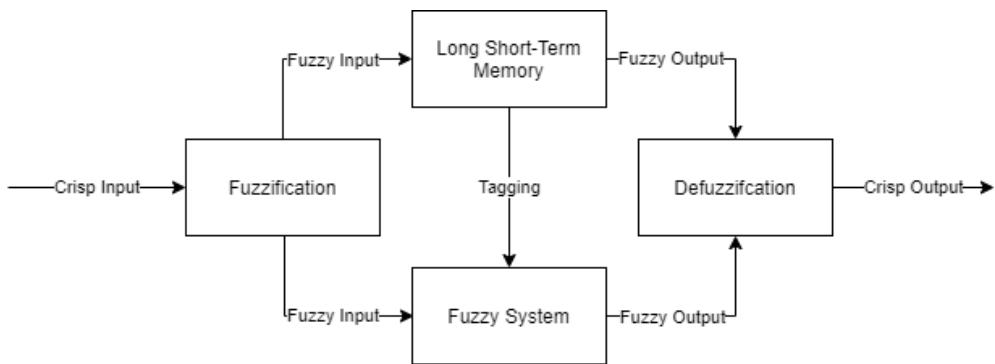


Figure 7.1 Architecture of Recurrent Neuro-Fuzzy Parallel System (RNFPS)

A recurrent neural network is able to learn the order dependence between items in a sequence, hence Long Short-Term Memory (LSTM) is being used instead of MLP. LSTM is a recurrent neural network that does well with time series data as it addresses the vanishing gradient issues that happen when training RNNs caused by long data sequences. It solves the issues by retaining history in an internal memory state which is based on new input and context passed from the previous cells. In addition, LSTM is able to predict multiple steps ahead. With this, LSTM is more effective when used to predict time-series data compared to the other artificial neural network.

7.2 Look ahead

The LSTM within RNFPS is able to look forward, where it predicts the output of the next day, given the input from the prior few days. With more historical data used to train the model, the prediction accuracy will increase.

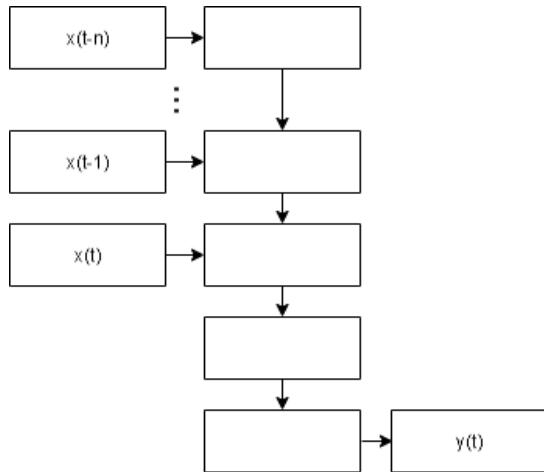


Figure 7.2 Architecture of hindsight LSTM

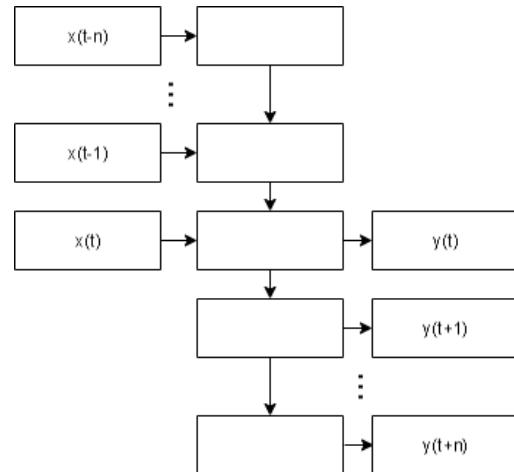


Figure 7.3 Architecture of look forward LSTM

	Input (n=5)	Output
Hindsight	x(t-5), x(t-4), x(t-3), x(t-2), x(t-1)	y(t)
	x(t-4), x(t-3), x(t-2), x(t-1), x(t)	y(t+1)
Look forward (eg. look forward=5)	x(t-5), x(t-4), x(t-3), x(t-2), x(t-1)	y(t+5)
	x(t-4), x(t-3), x(t-2), x(t-1), x(t)	y(t+6)

Table 7.1 Input and output structure of hindsight and look forward LSTM

7.3 Parameters used for Long Short-Term Memory

The clustering technique used will be UDCT as well, as seen from the previous experiment, it managed to give the best recall accuracy. The activation function for the input layer, hidden layers, and output layer will be all set to relu, and Adam optimizer will be used. The number of hidden layers

used will be set to three while the number of neurons will be optimized using a Genetic Algorithm (GA). The minimum and maximum number of nodes for the hidden layer is one to two hundred respectively. The number of nodes that have the highest R^2 value will be used for subsequent experiments. The number of steps ahead used here is 0 as we are comparing with the Modified Yager Network that does not have the ability to look ahead.

7.4 Recurrent Neuro-Fuzzy Parallel System (RNFPS) vs Modified Yager Network

In this section, the result of the Recurrent Neuro-Fuzzy Parallel System will be compared with the result of Modified Yager FNN. In addition, various clustering techniques will be covered as well. Lastly, the number of nodes for each hidden layer has been optimized using a genetic algorithm. Hence the comparison will be between

- Modified Yager Network with DCT clustering
- Recurrent Neuro-Fuzzy Parallel System (RNFPS) with UDCT clustering, nodes not optimized
- Recurrent Neuro-Fuzzy Parallel System (RNFPS) with UDCT clustering, nodes optimized
- Recurrent Neuro-Fuzzy Parallel System (RNFPS) with MLVQ clustering, nodes optimized
- Recurrent Neuro-Fuzzy Parallel System (RNFPS) with PFKP clustering, nodes optimized

There are a total of 6 folds, but a 5-fold CV is being used, while the 6th fold is used as verification.

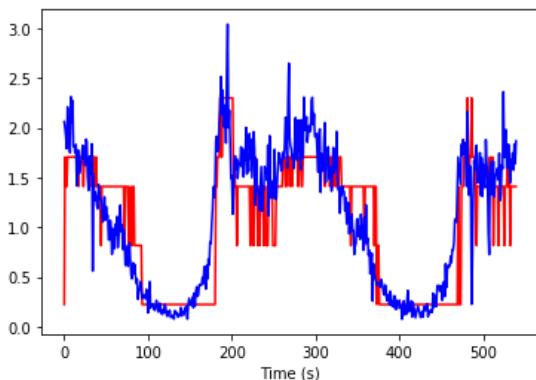


Figure 7.4 Result of Traffic data (fold 1) tested with Yager Network (UDCT)

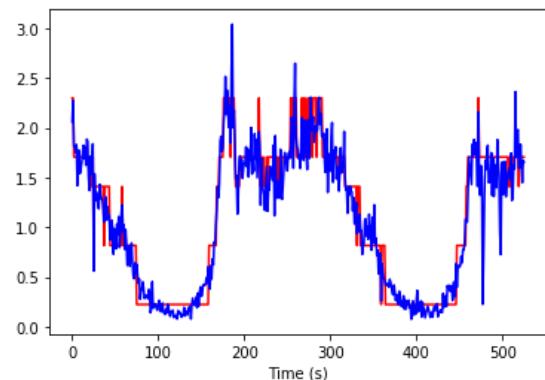


Figure 7.5 Result of Traffic data (fold 1) tested with Fuzzy-LSTM (UDCT & optimized node)

Training data	Testing data	Yager Network (UDCT)		RNFPS (UDCT)		RNFPS (UDCT)		RNFPS (MLVQ)		RNFPS (PFPK)	
				100-100-100		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
fold 1	fold 1	0.1596	0.6945	0.0483	0.8950	0.0391	0.9150	0.0529	0.8865	0.0406	0.9117
fold 2	fold 2	0.2673	0.5872	0.0816	0.8129	0.0655	0.8499	0.0881	0.7981	0.0869	0.8008
fold 3	fold 3	0.0959	0.6640	0.0631	0.7752	0.0589	0.7899	0.0681	0.7574	0.0937	0.6662
fold 4	fold 4	0.1910	0.6480	0.0794	0.8527	0.0667	0.8762	0.0764	0.8583	0.0667	0.8762
fold 5	fold 5	0.2537	0.5133	0.0872	0.8339	0.0701	0.8665	0.0686	0.8694	0.0828	0.8422
fold 6	fold 6	0.0525	0.8870	0.0637	0.8422	0.0732	0.8188	0.0479	0.8815	0.0526	0.8698

Table 7.2 Result of recall ability

Training data	Testing data	Yager Network (UDCT)		RNFPS (UDCT)		RNFPS (UDCT)		RNFPS (MLVQ)		RNFPS (PFPK)	
				100-100-100		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
20% (fold 1)	80%	0.2363	0.4879	0.1528	0.6692	0.1368	0.7038	0.1304	0.6176	0.1506	0.6739
20% (fold 2)	80%	0.1848	0.6078	0.1050	0.7762	0.1056	0.7748	0.1136	0.7579	0.1660	0.6462
20% (fold 3)	80%	0.2765	0.5388	0.1927	0.6089	0.1888	0.6168	0.163	0.6691	0.1763	0.6422
20% (fold 4)	80%	0.2442	0.4357	0.1430	0.6673	0.1156	0.7309	0.1195	0.7238	0.1179	0.7257
20% (fold 5)	80%	0.2071	0.5425	0.1662	0.6310	0.1591	0.6467	0.2206	0.5103	0.1588	0.6474
16.6% (fold 6)	83.4%	0.2454	0.4705	0.1210	0.7378	0.1306	0.7099	0.2137	0.6537	0.1193	0.7414

Table 7.3 Result of the generalization ability (20% training data and 80% testing data)

Training data	Testing data	Yager Network (UDCT)		RNFPS (UDCT)		RNFPS (UDCT)		RNFPS (MLVQ)		RNFPS (PFPK)	
				100-100-100		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170		Optimized for fold 1 162-189-170	
		RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
80%	20% (fold 1)	0.2191	0.5307	0.0395	0.9139	0.0375	0.9183	0.1370	0.8194	0.1358	0.8221
80%	20% (fold 2)	0.1793	0.5888	0.0470	0.8921	0.0453	0.8961	0.1390	0.8105	0.1438	0.7994
80%	20% (fold 3)	0.2092	0.5675	0.0375	0.8661	0.04010	0.8571	0.1319	0.7862	0.1393	0.7599
80%	20% (fold 4)	0.2480	0.5587	0.0749	0.8610	0.0496	0.9079	0.0618	0.8540	0.1417	0.7227
80%	20% (fold 5)	0.2800	0.4630	0.0472	0.9100	0.0465	0.9113	0.0501	0.9045	0.0474	0.8097

83.4%	16.6% (fold 6)	0.1697	0.5845	0.0470	0.8837	0.0374	0.9072	0.1309	0.8235	0.1344	0.8148
-------	-------------------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Table 7.4 Result of the generalization ability (80% training data and 20% testing data)

From tables 7.2, 7.3, and 7.4, it can be seen those with optimized nodes are performing a little better than those that are not, for both recall ability and generalization ability tests. On average, the UDCT clustering technique is performing better. Overall, it can be seen that Recurrent Neuro-Fuzzy Parallel System (RNFPS) is able to predict better than Modified Yager FNN, with an average of 28% increase of R^2 . For all architecture, it can be seen that accuracy is much better when there is more training data than testing data, which has been mentioned in the literature review that neural networks do better when trained with more amount of data.

7.5 Recurrent Neuro-Fuzzy Parallel System (RNFPS) vs Pure Long Short-Term Memory

Similar to section 6.3, this section presents the accuracy of RNFPS against the accuracy of pure LSTM. The accuracy of pure LSTM is expected to be higher than the accuracy of RNFPS due to the information lost during fuzzification, in exchange, users do gain interpretability.

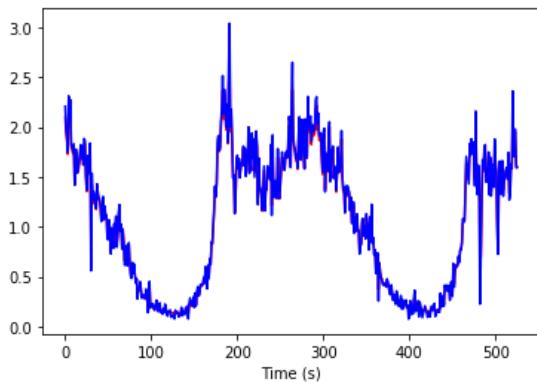


Figure 7.4 Result of Traffic data (fold 1) tested with pure LSTM (UDCT & optimized node)

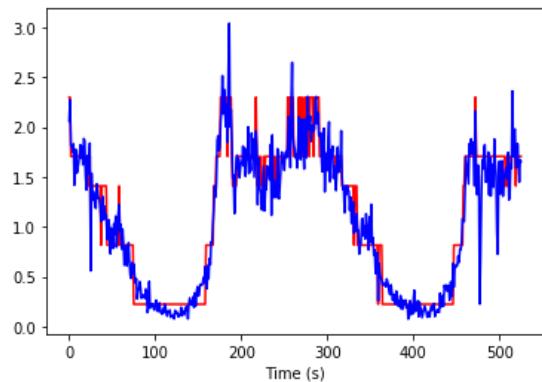


Figure 7.5 Result of Traffic data (fold 1) tested with Fuzzy-LSTM (UDCT & optimized node)

Training data	Testing data	Pure LSTM	RNFPS (UDCT)	RNFPS (UDCT)
		Optimized for fold 1 162-189-170	100-100-100	Optimized for fold 1 162-189-170

		RMSE	R2	RMSE	R2	RMSE	R2
fold 1	fold 1	0.0050	0.9892	0.0483	0.8950	0.0391	0.9150
fold 2	fold 2	0.0036	0.9916	0.0816	0.8129	0.0655	0.8499
fold 3	fold 3	0.0044	0.9840	0.0631	0.7752	0.0589	0.7899
fold 4	fold 4	0.0041	0.9922	0.0794	0.8527	0.0667	0.8762
fold 5	fold 5	0.0051	0.9901	0.0872	0.8339	0.0701	0.8665
fold 6	fold 6	0.0008	0.9978	0.0637	0.8422	0.0732	0.8188

Table 7.5 Result of recall ability

Training data	Testing data	Pure LSTM		RNFPS (UDCT)		RNFPS (UDCT)	
		Optimized for fold 1 162-189-170		100-100-100		Optimized for fold 1 162-189-170	
		RMSE	R2	RMSE	R2	RMSE	R2
20% (fold 1)	80%	0.0054	0.9881	0.1528	0.6692	0.1368	0.7038
20% (fold 2)	80%	0.0039	0.9916	0.1050	0.7762	0.1056	0.7748
20% (fold 3)	80%	0.0085	0.9827	0.1927	0.6089	0.1888	0.6168
20% (fold 4)	80%	0.0017	0.9959	0.1430	0.6673	0.1156	0.7309
20% (fold 5)	80%	0.0033	0.9924	0.1662	0.6310	0.1591	0.6467
16.6% (fold 6)	83.4%	0.0090	0.9800	0.1210	0.7378	0.1306	0.7099

Table 7.6 Result of the generalization ability (20% training data and 80% testing data)

Training data	Testing data	Pure LSTM		RNFPS (UDCT)		RNFPS (UDCT)	
		Optimized for fold 1 162-189-170		100-100-100		Optimized for fold 1 162-189-170	
		RMSE	R2	RMSE	R2	RMSE	R2
80%	20% (fold 1)	0.0002	0.9993	0.0395	0.9139	0.0375	0.9183
80%	20% (fold 2)	0.0009	0.9978	0.0470	0.8921	0.0453	0.8961
80%	20% (fold 3)	0.0003	0.9987	0.0375	0.8661	0.04010	0.8571
80%	20% (fold 4)	0.0004	0.9991	0.0749	0.8610	0.0496	0.9079
80%	20% (fold 5)	0.0005	0.9989	0.0472	0.9100	0.0465	0.9113
83.4%	16.6% (fold 6)	0.0002	0.9994	0.0470	0.8837	0.0374	0.9072

Table 7.7 Result of the generalization ability (80% training data and 20% testing data)

From Tables 7.5, 7.6, and 7.7, we are able to see that about an average of 12% of accuracy is being lost by using fuzzy values instead of crisp values. It can be seen from Table 7.5 and 7.7 that PNFPS compared to pure LSTM have an average of just 8% decrease of accuracy but in Table 7.6, it has an average of 25% decrease. It indicates that PNFPS does not work well with small amounts of training data.

7.6 Rules generated from Recurrent Neuro-Fuzzy Parallel System (RNFPS)

There are a total of five clusters for each variable, namely; Very Low, Low, Medium, High, and Very High.

Clusters: [VL, L, M, H, VH]

Time	Lane 1	Lane 2	Lane 3	Output
VL	VL	VL/L	VL	VL
L	VL	VL	VL	
L	M	H	VL	
VL	L	L	VL/L	L
L	VL	L	VL	
L	L	VL/L	VL	
L	L	L	L\ M	
L	L	M	M	
M	VL	H	M	
M	L	VH	H	
VH	VL \ L	L	VL	
VH	L	L	L \ M	
VH	L	M	L \ M	
VH	L	H	M	
VH	M	L \ M	M	
L	M	M	H	M

L	M	H	M / H	
L	H \ VH	H	H	
M	L	M \ H	L	
M	M	M	M	
M	M	L	L \ M	
M	H \ VH	M	VL	
M	VH	M	L \ M \ H	
M	VH	H	M \ H	
VH	L	H	L \ H	
VH	M	M \ H	H	
VH	M \ H	H	M	
VH	VH	H	H	
L	M	L	M	H
L	M \ H	M	M	
L	H \ VH	H	M	
L	H	H	VH	
M	M	M	VL \ L \ H	
M	M	H	VL \ L	
M	M	H	M	
M	L	H	M	
M	H	L	L \ M	
M	H	M	L \ M	
M	H	H	L \ M \ H	
M	VH	L	L	
H	M	M	H	
H	M	H	M \ H	
H	H	L \ M	M	
H	H	M	H	
H	H	H	VL \ M \ H	

H	H	VH	H	
H	VH	H	M	
VH	H	M	M	
VH	H	H	H \ VH	
L	H	VH	H \ VH	VH
L	VH	M	M	
L	VH	H	VH	
L	VH	VH	H \ VH	
M	H	VL	H	
H	L	L	VL	
H	M	VH	H	
H	VH	M	M	
H	VH	H	H \ VH	
H	VH	VH	H \ VH	

Table 7.8 Rules generated by RNFPS for traffic data

8 Stock market

The stock market has mainly two kinds of movements, specifically bull and a bear market. Where for the bull market, the stock prices are rising, while for the bear market, stock prices fall. Investors tend to buy when the stock prices are predicted to rise in the future and sell when the stock prices are predicted to fall in the future. In short, investors buy when the market is bullish and sell when the market is bearish. [50]

Investors have been investing the majority of their time and assets to understand the stock market, anticipating the next movement to gain high returns. However, due to the highly dynamic and volatile nature of the stock market, it is impractical to predict the movement without proper analysis. With the Recurrent Neuro-Fuzzy Parallel System, we are able to look ahead several steps, hence with the system, we will be able to predict the closing price even without technical indicators.

To state some significant events that have made a huge impact on the stock market movements; Cryptocurrency crash (September 2018) [51], Coronavirus stock market crash and rebound (February - December 2020) [52], Meme stocks (January 2021) [53].

Cryptocurrency Crash (September 2018)

S&P index refers to exchange-traded funds (ETFs) that represent around 500 of the largest companies in the US in terms of market capitalization, hence the S&P index is viewed as the market leader and is used as an indicator for market movements. And Bitcoin is the world's first decentralized cryptocurrency, which is also viewed as the market leader of cryptocurrency.

On 17th December 2017, Bitcoin reached its peak at the price of \$19,738 while on 22nd December 2017, it dropped by 45%. Eventually, on 15 November 2018, Bitcoin's price dropped to \$5,500.



Figure 8.1 Prices of S&P between March 2016 to March 2019

During the fall of Bitcoin or the market of cryptocurrency in general, S&P has peaked at the price of \$2,930 on 20th September 2018 but on 24th December 2018, it has dropped by 19.73% which is a value that is a four-year low.

Coronavirus Stock Market Crash (February - March 2020)

Dow Jones is a price-weighted measurement stock market index of 30 leading companies on the US stock exchange, hence the Dow Jones index can be viewed as the market leader and is usually used as an indication of prominent market movements.

Coronavirus stock market crash started on 20th February 2020 and ended on 7th April 2020, the market instability was caused by recession fears and market liquidity crisis due to the pandemic. From figure 8.2 below, we can see the movement of Dow Jones between January 2020 to December 2020 which shows the pre-crash high at the start of February and the following crash during the pandemic.

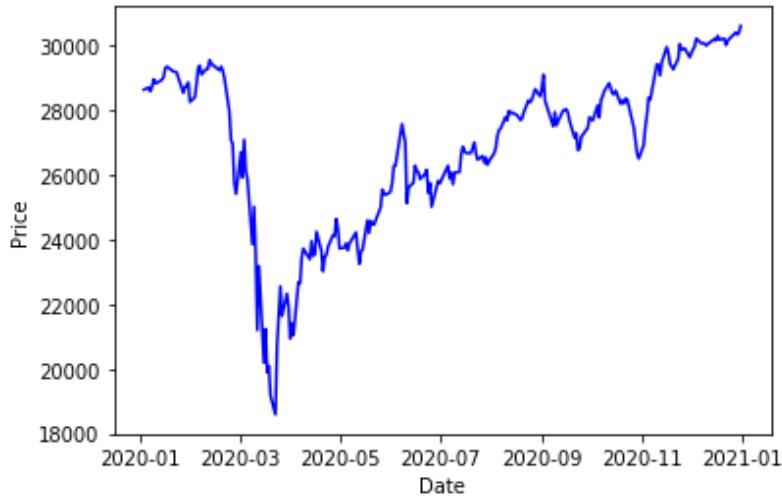


Figure 8.2 Prices of Dow Jones Industrial between January 2020 to December 2020

Coronavirus Stock Market Rebound (April 2020 - December 2020)

The global stock market reemerged into a bull market and even reached new highs during this period.

From figure 8.2 above, we can see that new peaks were made nearing the end of 2020, where the prices of Dow Jones surpassed the highest price right before the market crash.

Meme stocks (January 2021)

Meme stocks generally refer to a stock that has gone viral on online platforms like social media or online forums, which then catch the attention of retail investors. Eventually, the volumes and hence price skyrocket, not due to the company's performance but due to the hype on social media and online forums.

An example would be GameStop, where the stock prices soared in January 2021 which were fueled by a Reddit forum called WallStreetBets. Users on WallStreetBets started buying GameStop stocks after they found out a hedge fund had shorted the stock. [53] Before the Reddit post about GameStop, it was priced at \$13.66 and it surged more than 50% after the post, with prices in the range of \$31.4 to \$96.73. Eventually, Elon Musk - CEO of Tesla, tweeted about this topic and hence the public started to catch wind of it. It rose to new heights after the tweets, and at this point, it soared by 140% with a price of \$354.83.

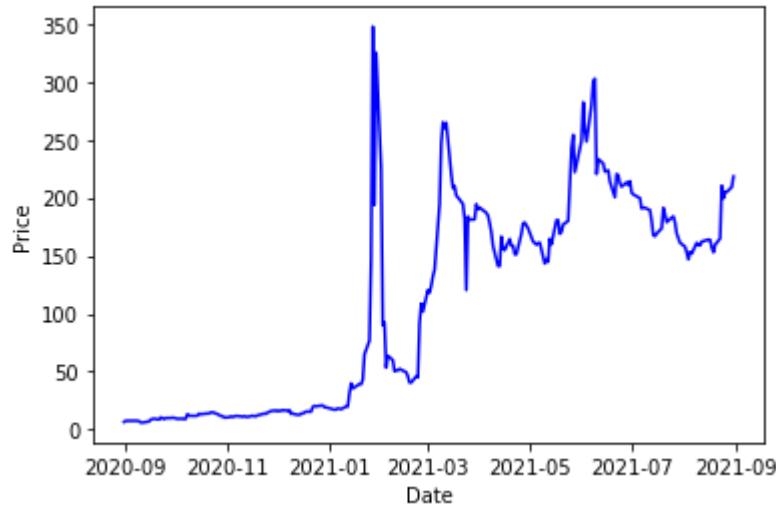


Figure 8.3 Prices of GameStop between September 2020 to September 2021

Meme stocks can grow up to 400% in a single week and it is possible to drop up to 50% overnight as well, as the saying goes; ‘high risk, high return’. These stocks are extremely volatile compared to the general market as it is almost impossible to know the exact movement even with technical indicators.

8.2 Implementation of Recurrent Neuro-Fuzzy Parallel System (RNFPS)

We will be using the same architecture introduced in the previous chapter, Recurrent Neuro-Fuzzy Parallel System (RNFPS).

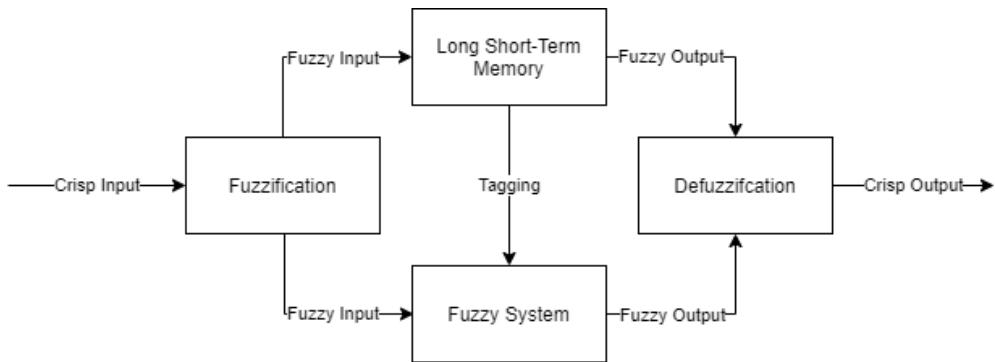


Figure 8.4 The architecture of Recurrent Neuro-Fuzzy Parallel System (RNFPS)

The clustering technique used will be UDCT as well, as seen from the previous experiments, it managed to give the best recall accuracy. The activation function for the input layer, hidden layers, and output layer will be all set to relu, and Adam optimizer will be used. The number of hidden layers used will be set to three, while the number of nodes for the hidden layer is two hundred.

8.3 Look forward

As mentioned in the previous chapter, the LSTM within RNFPS is able to look forward, where it predicts the output of the next day, given the input from the prior few days.

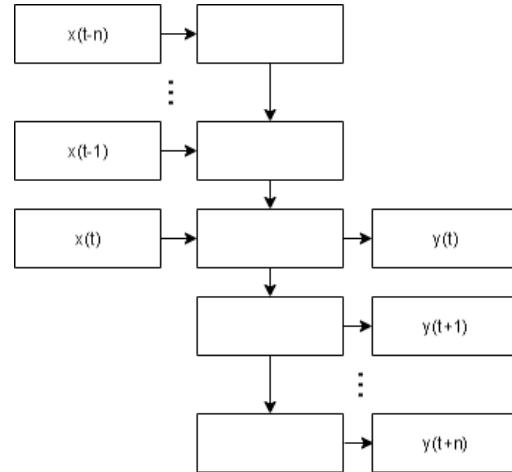
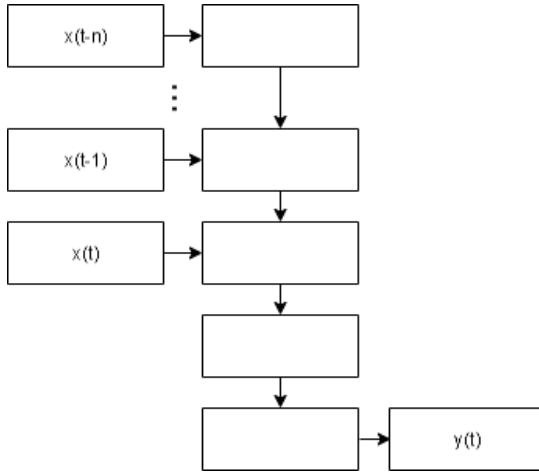


Figure 8.5 Architecture of hindsight LSTM

Figure 8.6 Architecture of look forward LSTM

The input data used would be the historical data of open price, high price, low price, close price, and volume of the previous 10 days to predict the close price of time t.

	Input (n=5)	Output
Look forward LSTM (look forward=5)	open(t-5), open(t-4), open(t-3), open(t-2), open(t-1) high(t-5), high(t-4), high(t-3), high(t-2), high(t-1) low(t-5), low(t-4), low(t-3), low(t-2), low(t-1) volume(t-5), volume(t-4), volume(t-3), volume(t-2), volume(t-1) close(t-5), close(t-4), close(t-3), close(t-2), close(t-1)	close(t+5)

Table 8.1 Input and output structure of look forward lstm

8.5 Accuracy of Recurrent Neuro-Fuzzy Parallel System used to predict Stock Market prices

This section presents the accuracy of RNFPS when used to predict stock market closing price instead of traffic data compared to the previous chapters. We will be experimenting with the accuracy of the different amounts of look ahead as well.

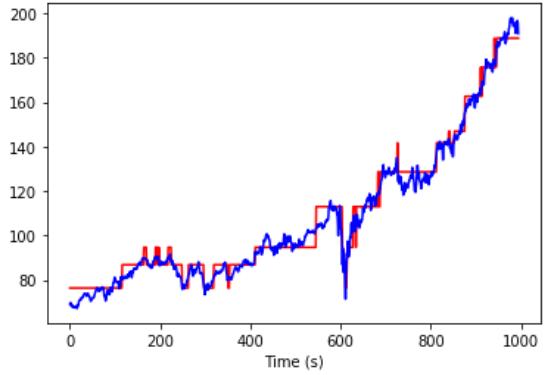


Figure 8.7 Result of Stock Market prediction (NDAQ) tested with RNFPS (look forward = 1)

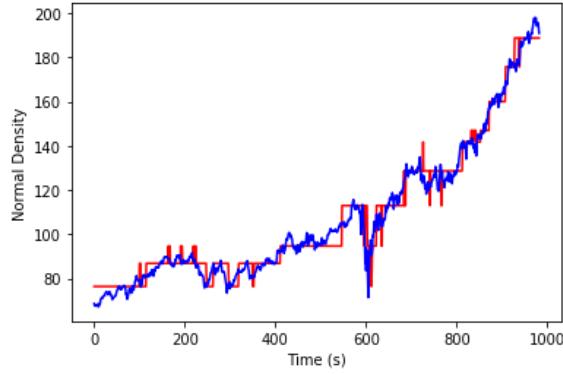


Figure 8.8 Result of Stock Market prediction (NDAQ) tested with RNFPS (look forward = 7)

Type of stocks	Training / Testing data	Average	Look forward						
			1	2	3	4	5	6	7
Bull	TSLA	0.9721	0.9806	0.9851	0.9853	0.9640	0.9675	0.9598	0.9622
	AAPL	0.9831	0.9869	0.9870	0.9875	0.9862	0.9843	0.9820	0.9679
Bear	BBBY	0.9028	0.9371	0.9356	0.9050	0.8616	0.8955	0.8947	0.8898
	BB	0.9134	0.9541	0.9471	0.9250	0.9214	0.9097	0.8833	0.8529
Index	NDAQ	0.9740	0.9769	0.9787	0.9788	0.9725	0.9749	0.9670	0.9690
	DJI	0.8994	0.9249	0.9553	0.9097	0.9116	0.9026	0.8489	0.8428
Meme	GME	0.7931	0.8093	0.8138	0.7473	0.7559	0.7645	0.8468	0.8143
	AMC	0.8294	0.9318	0.9126	0.8644	0.8438	0.8053	0.7017	0.7462

Table 8.2 Accuracy of look forward prediction

From Table 8.2 we are able to see that all accuracy does decrease when it is predicting an output with bigger forward window size. And meme stocks have the lowest accuracy among all the other types of stocks, this is probably because they are more dynamic and volatile compared to the rest.

9 Trading Strategies and Analysis

In this chapter, we will discuss the different trading strategies that investors use and the different analysis methods that are used to analyze the movement of the market. After being able to predict the future prices in advance, we still have to come up with a strategy and analysis method to automate trading when the signal is given.

9.1 Trading Strategies

Trading strategies refer to the different methods used to time profitable trades, each has its own advantages and disadvantages which will be discovered in the next few points.

End of day trading

End-of-day trading refers to traders trading when it is near the market closing time. Traders start operating when it is clear that the price is going to close or settle. End-of-day trading requires a clear understanding of the price action compared to the price movement of the previous day. With this knowledge, traders are able to predict the price's movement based on the price action.

There are mainly two advantages of end-of-day trading, firstly it is simple as traders do not need to enter multiple positions, and secondly, it is not as time-consuming as other strategies. However, this strategy involves overnight risk as well, which can be avoided by placing a stop-loss order. [59]

Swing trading

Swing traders utilize the market's oscillation, where they buy when they expect the market to rise and sell when they expect the price to fall. This strategy heavily depends on the prediction of the length and duration of each swing. The level of supply or demand and momentum is important to swing trading as well. During strong trends, to enter in the trend's direction, it is possible to use retracement swings. When a new momentum high is formed, traders then look for the highest probability trade

which is typically to buy the first dip. And when a new momentum low is formed, traders tend to sell the first rally. [54], [55]

The advantages of swing trading include having several trade opportunities as it involves trading both sides of the market and it is not as time-consuming compared to the other strategies. The disadvantage involves overnight risk and requires intensive research on how to analyze the market.

Day trading

Day trading refers to buying and selling within the same day. Positions are closed on the same day and no position is held overnight. Traders utilize the price fluctuation between market open and close hours. They also tend to hold several positions open within a day. Positions are not open overnight, as traders want to mitigate overnight risk. To properly utilize this strategy, traders have to understand the support levels, resistance levels, and possible reactions. [56]

The advantage of day trading includes having no overnight risk, minimized intraday risk, flexible trading and it creates several trade opportunities. Day trading includes minimized intraday risk as short-term trades highly reduce the probability of risk that might exist in longer-term trades. There are several trade opportunities as traders are able to open and close multiple positions within a day.

The disadvantages of day trading involve the requirement of a regimen and possible flat trades. A regimen is required to mitigate risk, where traders should have a predetermined strategy indicating the entry and exit levels. While flat trades refer to positions that do not move within a day.

Position trading

Position trading seeks to hold positions for a long period of time, hoping to profit from long-term trends instead of profiting from minor price fluctuations. Traders using this strategy tend to benefit from both the upside and downside of the market movements. Fundamental analysis is required to assess the potential price trends within the market, along with consideration of other factors like

market trends and historical patterns. When a market trend has settled, traders jump onto the trend, and when the trend breaks, traders will exit the position. [57], [58]

The advantages of position trading involve high profits and require a lesser time commitment. While the disadvantages of position trading include consequential loss.

Trading method selected

End-of-day trading will be utilized as the Recurrent Neuro-Fuzzy Parallel System is able to predict closing price, which is the best fit with this trading method which sells at the end of the day.

9.2 Trading Analysis

There are two methods investors use to analyze stocks, technical and intrinsic analysis. Both methods use past data to predict the future behavior assuming that the stock market will repeat trend movements. We will be focusing on technical analysis which tries to evaluate a stock's movement based on variables like prices, and volume. There are several types of technical indicators; trend, volatility, volume, and momentum indicators. Within each type of indicator, there are several formulas that contribute to the prediction. The different kinds of indicators were discussed in the literature review.

9.3 Automated Trading Algorithms

With so many different technical indicators, it has been proven that the most accurate predictions use two to three technical indicators. Any more will lead to confusing and contradicting signals and if only one technical indicator is used, there is no confirmation of signals given. [62] The following techniques discussed do not use any deep learning to predict future prices but instead, they use technical indicators to predict the price movements.

Data Trader strategy

This strategy is adapted from [60]. The first technique uses three technical indicators namely; Stochastic Oscillator (SO), Relative Strength Index (RSI), and Moving Average Convergence Divergence

(MACD). This technique buys when all SO hits an oversold signal, RSI is above the middle line and when MACD is above the signal line and sells when SO hits an overbought signal, RSI is below the middle line, and when MACD is below the signal line.

```

For i in range(date):
    If SO>=oversold and RSI>=middle and MACD>signal and buy
    conditions met:
        Buy
    If SO>overbought and RSI<middle and MACD<signal and sell
    conditions met:
        Sell

```

Figure 9.1 Pseudocode of Data Trader trading strategy

The strategy was tested with a capital of \$1,000 and gained a profit of \$28,041 over 134 days, which equates to a 28% gain.

Dourra strategy

This strategy is adapted from [61]. For the second technique, they use three indicators as well namely; Rate of Change (ROC), Stochastic Oscillator (SO), and Bollinger Bands (BB). The paper uses fuzzy values as well, where the technical indicators are fuzzified into four different levels; low, medium, big, and large. The paper concluded their research with the following strategy:

- Buy if the thirty-day ROC is big, two-day ROC is big, a 3-period moving average of %K is large and the support line is low.
- Sell if thirty-day ROC is low, two-day ROC is medium, a 3-period moving average of %K is low and the resistance line is low.

```

For i in range(date):
    If 30-ROC==big and 2-ROC==big and %K==large and support==low
    and buy conditions met:
        Buy
    If 30-ROC==low and 2-ROC==medium and %K==low and
    resistance==low and buy conditions met:
        Sell

```

Figure 9.2 Pseudocode of Dourra trading strategy

The strategy was tested against a few different stocks and over different periods of time as well. It uses a capital of \$10,000 and no other leverage. This strategy has two levels; low-risk investment and high-risk investment. When Compaq computer stock was tested using this strategy, the period; 3rd January to 30th July 1999 was used. The low-risk investment amounted to \$24,709, while the high risk amounted to

\$33,916. When General Motors stock tested using this strategy, the period; 3rd January to 10th May 1999 was used. The low-risk investment amounted to \$15,813, while the high risk amounted to \$17,081.

10 Implementation of Automated Trading Algorithm with Recurrent Neuro-Fuzzy Parallel System

In this chapter, we will be using the Recurrent Neuro-Fuzzy Parallel System with an automated trading algorithm created with the predicted value of closing prices of specific stocks in the near future. The automated trading algorithm relies on the future closing price that is predicted by the Recurrent Neuro-Fuzzy Parallel System. The algorithm buys, sells, and holds according to an algorithm that has been tested.

10.1 Automated Trading Algorithm

As mentioned above, the trading system will buy, sell and hold accordingly. To find the best trading strategy, a genetic algorithm is used to optimize the buy and sell strategy. Where the genetic algorithm finds the best day to sell when the system predicts that the closing price is going to fall in the future, and finds the best day to buy when the system predicts that the closing price is going to rise in the future.

```
Set parameters
Generate the initial population
While i< MaxIteration and BestFitness<MaxFitness do:
    Fitness calculation
    Selection
    Crossover
    Mutation
End while
Decode the individual with maximum fitness
Return the best solution
```

Figure 10.1 Pseudocode of Genetic Algorithm

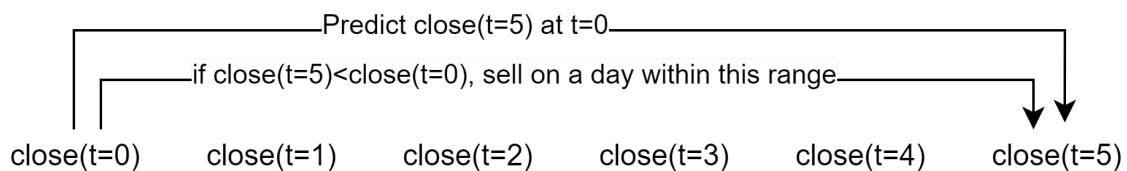


Figure 10.2 Diagram of Function used within Genetic Algorithm

```

Def test(a,b):
    For i in range(date):
        If currentClosePrice<predictedClosePrice[i+a] and buy conditions met:
            Buy
        If currentClosePrice>predictedClosePrice[i+b] and sell conditions met:
            Sell
    Return -total assets

```

Figure 10.3 Pseudocode of Function used within Genetic Algorithm

Since we are predicting the closing price of the stock, we will be using end-of-day trading as mentioned in the previous chapter. The trading strategy starts off with an asset of \$100,000 and a risk percentage is designated as well, for testing purposes, there are three levels; low, medium, high risk. The risk taken here represents the percentage of leverage that is going to be used.

```

For i in range(date):
    If currentClosePrice<predictedClosePrice[i+a] and buy conditions met:
        Buy
    If currentClosePrice>predictedClosePrice[i+b] and sell conditions met:
        Sell

```

Figure 10.4 Pseudocode of Trading Algorithm

10.2 Result of Automated Trading Algorithm with Recurrent Neuro-Fuzzy Parallel System

The automated trading algorithm has been explained above, along with the prediction method. Here, we will see the strategy tested across several types of stocks; bull, bear, index, and meme stocks. The initial capital is \$100,000 and leverage depends on the risk level taken. Leverage is assumed to have 6% interest annually which will be paid at the end of the simulation and is deducted before the final profit.

	avg % gained		Bull				Bear				
			TSLA		AAPL		BBBY		BB		
			\$	%	\$	%	\$	%	\$	%	
Low Risk with NO leverage of 0%	SO, RSI and MACD	Vanilla	4	267,600	3	336,525	3	197,558	2	107,806	1
			4	632,835	6	510,560	5	200,868	2	106,712	1
	Recurrent Neuro-Fuzzy	LF - 5	7	477,796	5	221,824	2	144,644	1	798,288	8
		LF - 10	6	386,114	4	207,109	2	1,396,891	14	401,723	4

Medium Risk with leverage of 3%	Parallel System	LF - 15	12	872,191	9	247,989	2	626,797	6	251,193	3	
		LF - 20	7	476,486	5	339,999	3	241,119	2	189,563	2	
	Pure LSTM	LF - 5	16	1,113,449	11	315,331	3	669,502	7	1,668,854	17	
		LF - 10	27	1,154,153	12	546,186	5	1,016,875	10	484,714	5	
		LF - 15	58	803,464	8	490,293	5	951,896	10	536,887	5	
		LF - 20	25	1,568,087	16	609,765	6	805,917	8	280,562	3	
		SO, RSI and MACD	Vanilla	82	7,680,866	77	3,904,852	39	5,764,074	58	1,825,689	18
		ROC, SO, BB		43	6,626,598	66	1,704,470	17	1,185,723	12	536,573	5
	Recurrent Neuro-Fuzzy Parallel System	LF - 5	36	1,376,673	14	936,759	9	1,230,695	12	5,423,818	54	
		LF - 10	74	1,521,785	15	1,346,426	13	22,043,191	220	6,570,931	66	
		LF - 15	856	8,962,493	90	1,757,494	18	48,083,853	481	1,993,787	20	
		LF - 20	297	2,990,996	30	2,637,193	26	20,544,024	205	2,017,587	20	
	Pure LSTM	LF - 5	873	22,452,374	225	3,557,729	36	5,438,805	54	15,730,289	157	
		LF - 10	292	19,728,169	197	3,015,628	30	8,372,398	84	639,032	6	
		LF - 15	2,365	34,806,161	348	1,952,284	20	4,177,497	42	2,125,935	21	
		LF - 20	143	15,819,210	158	2,344,212	23	3,458,681	35	2,287,699	23	
High Risk with leverage of 9%	Parallel System	SO, RSI and MACD	Vanilla	134	2,078,703	21	9,892,612	99	3,028,485	30	1,069,525	11
		ROC, SO, BB		72	12,260,899	123	3,440,535	34	2,003,807	20	2,027,610	20
	Recurrent Neuro-Fuzzy Parallel System	LF - 5	90	3,211,529	32	2,126,216	21	10,798,398	108	30,524,927	305	
		LF - 10	406	1,730,824	17	1,649,149	16	223,872,726	2,239	17,291,106	173	
		LF - 15	3,780	6,523,163	65	2,111,681	21	295,211,216	2,952	6,733,276	67	
		LF - 20	1,072	13,414,541	134	1,608,599	16	27,536,636	275	5,772,973	58	
	Pure LSTM	LF - 5	181	16,645,824	166	2,222,909	22	7,272,267	73	2,264,005	23	
		LF - 10	346	9,039,151	90	2,297,153	23	84,089,582	841	1,910,051	19	
		LF - 15	506	32,941,654	329	3,185,190	32	13,578,525	136	2,281,888	23	
		LF - 20	193	6,496,215	65	2,269,430	23	930,288	9	812,220	8	

Table 10.1 Profit of trading strategy applied with predicted close values generated by RNFPS

Low Risk with NO leverage (0%)	SO, RSI and MACD	Vanilla	avg %	Index				meme stocks			
				NDAQ		DJI		GME		AMC	
				\$	%	\$	%	\$	%	\$	%
Low Risk with NO leverage (0%)	SO, RSI and MACD	Vanilla	4	256,786	3	177,779	2	608,397	6	1,465,140	15
			4	203,979	2	160,438	2	999,248	10	344,228	3
	Recurrent Neuro-Fuzzy	LF - 5	7	221,278	2	150,726	2	2,041,105	20	1,760,707	18
		LF - 10	6	165,703	2	153,509	2	832,060	8	1,473,683	15

	Parallel System	LF - 15	12	208,379	2	178,053	2	663,848	7	6,339,437	63	
		LF - 20	7	204,885	2	155,009	2	1,601,115	16	2,275,361	23	
Medium Risk with leverage of 3% (30%)	Pure LSTM	LF - 5	16	219,679	2	116,619	1	983,521	10	7,778,916	78	
		LF - 10	27	189,640	2	167,165	2	5,775,624	58	12,328,579	123	
		LF - 15	58	223,757	2	180,811	2	24,062,376	241	18,970,970	190	
		LF - 20	25	215,888	2	167,559	2	11,804,471	118	4,379,347	44	
		SO, RSI and MACD	Vanilla	82	1,364,515	14	117,563	1	19,314,541	193	25,681,734	257
		ROC, SO, BB		43	460,370	5	115,230	1	12,282,480	123	11,777,261	118
High Risk with leverage of 9% (90%)	Recurrent Neuro-Fuzzy Parallel System	LF - 5	36	773,846	8	375,862	4	11,872,360	119	7,161,045	72	
		LF - 10	74	865,657	9	371,002	4	5,050,576	51	21,169,259	212	
		LF - 15	856	1,219,416	12	362,019	4	517,170,423	5,172	105,584,237	1,056	
		LF - 20	297	1,834,595	18	412,378	4	18,141,556	181	188,972,769	1,890	
	Pure LSTM	LF - 5	873	2,000,023	20	231,179	2	494,122,033	4,941	154,606,785	1,546	
		LF - 10	292	1,268,843	13	187,725	2	59,410,022	594	141,028,657	1,410	
		LF - 15	2,365	944,714	9	240,651	2	1,698,235,347	16,982	149,500,446	1,495	
		LF - 20	143	1,169,853	12	128,957	1	80,655,145	807	8,571,123	86	
	SO, RSI and MACD	Vanilla	134	3,432,373	34	1,030,988	10	51,754,676	518	34,731,708	347	
			72	1,526,339	15	662,779	7	25,405,245	254	10,334,676	103	
	Recurrent Neuro-Fuzzy Parallel System	LF - 5	90	782,933	8	675,778	7	2,820,117	28	20,699,107	207	
		LF - 10	406	1,152,380	12	593,565	6	9,582,141	96	69,034,548	690	
		LF - 15	3,780	1,226,261	12	568,161	6	1,856,287,207	18,563	855,717,016	8,557	
		LF - 20	1,072	1,362,705	14	1,034,356	10	76,729,157	767	729,809,853	7,298	
	Pure LSTM	LF - 5	181	1,829,610	18	300,003	3	31,429,169	314	82,712,697	827	
		LF - 10	346	1,496,864	15	126,671	1	60,160,848	602	117,479,628	1,175	
		LF - 15	506	1,613,485	16	158,207	2	58,944,643	589	291,732,177	2,917	
		LF - 20	193	627,837	6	143,196	1	20,072,106	201	122,687,363	1,227	

Table 10.2 Profit of trading strategy applied with predicted close values generated by RNFPS

Throughout the different risk levels and stocks, it can be seen that look forward to close(t=15) has the best profit. GME stocks tend to have the highest profit as well, even though Table 8.2 shows that GME has the lowest average accuracy compared to the rest of the stocks. This can be due to the dynamic and volatility of GME stocks as with so much movement, there is a lot of opportunities for traders to gain higher profits.

11 Conclusion

In this paper, we have introduced the different ways to incorporate fuzzy neural networks to predict various problems. Through fuzzy logic, we are also able to understand the decision-making reasons behind the neural network's prediction. However, with fuzzy logic being introduced, we gained human understanding of the trade-off of accuracy loss.

In the first few chapters, Yager-PC FNN and Modified Yager FNN were being used to experiment on the Iris dataset and traffic dataset. UDCT, MVLQ, PFKP were used to experiment with the two architecture as well.

With several new neural network models and libraries being introduced, MLP was introduced to integrate with both Yager-PC FNN and Modified Yager FNN to compare the accuracy of the integrated system. Both Yager-PC FNN and Modified Yager FNN integrated with MLP gave favorable accuracy that surpasses the vanilla version of Yager-PC FNN and Modified Yager FNN.

In chapter 7, Recurrent Neuro-Fuzzy Parallel System (RNFPS) was introduced, which was created by integrating Modified Yager FNN with LSTM. RNFPS was tested with traffic datasets and with the stock market as well. Where an automated trading system was integrated with RNFPS which utilized the prediction of close prices generated by the prediction system. The system then experimented with the various trading strategy that has been utilized by others.

11.1 Recommendations for future works

The current architecture is able to predict the future close price of the chosen stock with an average accuracy of 92%. The result from the automated trading strategy introduced in this paper has gains that range from 1% to 18,563%. In chapter 8, we have gone through the topic of meme stocks which were heavily dependent on the sentiment of the stocks on a subreddit called r/Wallstreetbets where they discuss stocks and options trading.

To further optimize the automated trading system, sentiment analysis should be incorporated. As the world is moving and changing at a rapid speed, sentiment analysis on news and social media is

deemed to be important and necessary to take into consideration. Covid market crash is another example that can be used as well.

12 References

- [1] Brownlee, J. (2019, May 22). Difference Between Classification and Regression in Machine Learning. *Machine Learning Mastery*.
<https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/>
- [2] Heung-II Suk, Chapter 1 - An Introduction to Neural Networks and Deep Learning, Editor(s): S. Kevin Zhou, Hayit Greenspan, Dinggang Shen, *Deep Learning for Medical Image Analysis*, Academic Press, 2017, Pages 3-24, ISBN 9780128104088
- [3] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
- [4] Shi Dong, Ping Wang, Khushnood Abbas, A survey on deep learning and its applications, *Computer Science Review*, Volume 40, 2021, 100379, ISSN 1574-0137,
- [5] Nikolaus Kriegeskorte, Tal Golan, Neural network models and deep learning, *Current Biology*, Volume 29, Issue 7, 2019, Pages R231-R236, ISSN 0960-9822,
- [6] E. Nishani and B. Çiço, "Computer vision approaches based on deep learning and neural networks: Deep neural networks for video analysis of human pose estimation," 2017 6th Mediterranean Conference on Embedded Computing (MECO), 2017, pp. 1-4, doi: 10.1109/MECO.2017.7977207.
- [7] Noriega, L. (2005). Multilayer perceptron tutorial. School of Computing. Staffordshire University.
- [8] Dumoulin, V. (2016, March 23). A guide to convolution arithmetic for deep learning. ArXiv.Org.
<https://arxiv.org/abs/1603.07285>
- [9] Tch, A. (2021, February 16). The mostly complete chart of Neural Networks, explained. Medium.
<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>
- [10] Eduardo Ramirez, Patricia Melin, German Prado-Arechiga, Hybrid model based on neural networks, type-1 and type-2 fuzzy systems for 2-lead cardiac arrhythmia classification, *Expert Systems with Applications*, Volume 126, 2019, Pages 295-307, ISSN 0957-4174

- [11] Zhenyu Yuan, Handong Huang, Yuxin Jiang, Jingjing Li, Hybrid deep neural networks for reservoir production prediction, Journal of Petroleum Science and Engineering, Volume 197, 2021, 108111, ISSN 0920-4105
- [12] Yuan, Z. (2020, May 18). Hybrid-DNNs: Hybrid Deep Neural Networks for Mixed Inputs. ArXiv.Org. <https://arxiv.org/abs/2005.08419>
- [13] Wikipedia contributors. (2021, July 5). Fuzzy control system. Wikipedia. https://en.wikipedia.org/wiki/Fuzzy_control_system
- [14] MATLAB® and Its Applications in Engineering: [Based on MATLAB 7.5 (R2007b)]. (2021). https://www.oreilly.com/library/view/matlab-and-its/9788131716816/9788131716816_ch11lev1sec10.html
- [15] F. Es-Sabery, A. Hair, J. Qadir, B. Sainz-De-Abajo, B. García-Zapirain and I. D. L. Torre-Díez, "Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier," in IEEE Access, vol. 9, pp. 17943-17985, 2021, doi: 10.1109/ACCESS.2021.3053917.
- [16] Vandana Bhatia, Rinkle Rani, "A parallel fuzzy clustering algorithm for large graphs using Pregel", Expert Systems with Applications, Volume 78, 2017, Pages 135-144, ISSN 0957-4174
- [17] Y. Deng, Z. Ren, Y. Kong, F. Bao and Q. Dai, "A Hierarchical Fused Fuzzy Deep Neural Network for Data Classification," in IEEE Transactions on Fuzzy Systems, vol. 25, no. 4, pp. 1006-1012, Aug. 2017, doi: 10.1109/TFUZZ.2016.2574915.
- [18] A. Wibowo, Sutikno, Kushartantya, H. A. Wibawa and A. Wibisono, "Parallel Cascade Fuzzy Inference System at the environment changes, case study: Automated Guide Vehicle Robot using ultrasonic sensor," 2012 International Conference on Advanced Computer Science and Information Systems (ICACSIS), pp. 319-324.
- [19] A. Loria and E. Panteley, "Cascaded nonlinear time-varying systems: Analysis and design," in Advanced Topics in Control Systems Theory. Berlin, Germany: Springer-Verlag, pp. 23–64.
- [20] Zhang, Ren & Shen, Furao & Zhao, Jinxi. (2014). A model with Fuzzy Granulation and Deep Belief Networks for exchange rate forecasting. Proceedings of the International Joint Conference on Neural Networks. 366-373. 10.1109/IJCNN.2014.6889448.

- [21] H. C. Quek and K. K. Ang (2011), Computational Modeling and Simulation of Intellect: Current State and Future Perspectives (pp.485-509), MLVQ: a modified learning vector quantization algorithm for identifying centroids of fuzzy membership functions 10.4018/978-1-60960-551-3.ch019
- [22] White, D. and P. Ligomenides (1993). GANNet: A genetic algorithm for optimizing topology and weights in neural network design, Berlin, Heidelberg, Springer Berlin Heidelberg.
- [23] G. Iovane, A. Amorosia, M. Leone, M. Nappi, G. Tortora, Multi indicator approach via mathematical inference for price dynamics in information fusion context, *Information Sciences*, Volume 373, 2016, Pages 183-199, ISSN 0020-0255, <https://doi.org/10.1016/j.ins.2016.08.063>.
- [24] Gudelek, M. U., Boluk, S. A., & Ozbayoglu, A. M. (2017). A deep learning based stock trading model with 2-D CNN trend detection. 2017 IEEE Symposium Series on Computational Intelligence (SSCI). Published. <https://doi.org/10.1109/ssci.2017.8285188>
- [25] Yang H., Zhu Y., Huang Q. (2018) A Multi-indicator Feature Selection for CNN-Driven Stock Index Prediction. In: Cheng L., Leung A., Ozawa S. (eds) Neural Information Processing. ICONIP 2018. Lecture Notes in Computer Science, vol 11305. Springer, Cham.
https://doi.org/10.1007/978-3-030-04221-9_4
- [26] Z. Liu and D. Xiao, "An Automated Trading System with Multi-indicator Fusion Based on D-S Evidence Theory in Forex Market," 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 2009, pp. 239-243, doi: 10.1109/FSKD.2009.395.
- [27] Dourra, H., & Siy, P. (2002). Investment using technical analysis and fuzzy logic. *Fuzzy Sets and Systems*, 127(2), 221–240. [https://doi.org/10.1016/s0165-0114\(01\)00169-5](https://doi.org/10.1016/s0165-0114(01)00169-5)
- [28] Chandima, A. (2020). Use Stock Market Data to assist investors on getting more accurate transaction decision. *International Journal of Scientific and Research Publications (IJSRP)*, 11(1), 762–770. <https://doi.org/10.29322/ijrsp.11.01.2021.p10993>
- [29] Deng, Y., Ren, Z., Kong, Y., Bao, F., & Dai, Q. (2016). A hierarchical fused fuzzy deep neural network for data classification. *IEEE Transactions on Fuzzy Systems*, 25(4), 1006-1012.
- [30] Wang, Y., Wu, Z., & Zhang, J. (2016, October). Damaged fingerprint classification by Deep Learning with fuzzy feature points. In 2016 9th international congress on image and signal processing, BioMedical engineering and informatics (CISP-BMEI) (pp. 280-285). IEEE.

- [31] Tabrizi, P. R., Mansoor, A., Cerrolaza, J. J., Jago, J., & Linguraru, M. G. (2018, April). Automatic kidney segmentation in 3D pediatric ultrasound images using deep neural networks and weighted fuzzy active shape model. In 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018) (pp. 1170-1173). IEEE.
- [32] Chopade, H. A., & Narvekar, M. (2017, November). Hybrid auto text summarization using deep neural network and fuzzy logic system. In 2017 International Conference on Inventive Computing and Informatics (ICICI) (pp. 52-56). IEEE.
- [33] Zhang, R., Shen, F., & Zhao, J. (2014, July). A model with fuzzy granulation and deep belief networks for exchange rate forecasting. In 2014 International Joint Conference on Neural Networks (IJCNN) (pp. 366-373). IEEE.
- [34] Zhou, S., Chen, Q., & Wang, X. (2014). Fuzzy deep belief networks for semi-supervised sentiment classification. Neurocomputing, 131, 312-322.
- [35] A. Singh, C. Quek and S. - Cho, "DCT-Yager FNN: A Novel Yager-Based Fuzzy Neural Network With the Discrete Clustering Technique," in IEEE Transactions on Neural Networks, vol. 19, no. 4, pp. 625-644, April 2008, doi: 10.1109/TNN.2007.911709.
- [36] Keller, J. M., Yager, R. R., & Tahani, H. (1992). Neural network implementation of fuzzy logic. Fuzzy Sets and Systems, 45(1), 1–12. [https://doi.org/10.1016/0165-0114\(92\)90086-j](https://doi.org/10.1016/0165-0114(92)90086-j)
- [37] Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446.
- [38] Anderson, E. (1936). The species problem in Iris. Annals of the Missouri Botanical Garden, 23(3), 457-509.
- [39] Gopalkrishnan, V., Steier, D., Lewis, H., & Guszcza, J. (2012, August). Big data, big business: bridging the gap. In Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (pp. 7-11).
- [40] Jang, J. S. (1993). ANFIS: adaptive-network-based fuzzy inference system. IEEE transactions on systems, man, and cybernetics, 23(3), 665-685.

- [41] Tung, W. L., & Quek, C. (2002). GenSoFNN: A generic self-organizing fuzzy neural network. *IEEE Transactions on Neural Networks*, 13(5), 1075-1086.
- [42] Kim, J., & Kasabov, N. (1999). HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems. *Neural networks*, 12(9), 1301-1319.
- [43] Zhou, R. W., & Quek, C. (1996). POPFNN: A pseudo outer-product based fuzzy neural network. *Neural Networks*, 9(9), 1569-1581.
- [44] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), e00938.
- [45] Tan, G. K. (1997). Feasibility of predicting congestion states with neural network models. Final Year Project Report, School of Civil and Structural Engineering, Nanyang Technological University, Singapore.
- [46] Gad, A. F. (2021, June 11). PyGAD: An Intuitive Genetic Algorithm Python Library. ArXiv.Org. <https://arxiv.org/abs/2106.06158>
- [47] Chen, Y. Y. (2021). Recurrent Neural Network Embedded Fuzzy System (RNNEFS) with its Applications in Stock Market Forecasting & MACD Trading Strategies. Final Year Project Report, School of Computer Science and Engineering, Nanyang Technological University, Singapore.
- [48] L. (2017, August 27). Neural network approach to Iris dataset. Kaggle. <https://www.kaggle.com/louisong97/neural-network-approach-to-iris-dataset>
- [49] Lemos, V. (2019, September 4). Using Multilayer Perceptron in Iris Flower DataSet - Computing Science. Medium. <https://medium.com/computing-science/using-multilayer-perceptron-in-classification-problems-iris-flower-6fc9fbf36040>
- [50] Chen, N. F., Roll, R., & Ross, S. A. (1986). Economic forces and the stock market. *Journal of business*, 383-403.
- [51] Wikipedia contributors. (2021, September 6). *Cryptocurrency bubble*. Wikipedia. https://en.wikipedia.org/wiki/Cryptocurrency_bubble
- [52] Wikipedia contributors. (2021b, September 25). *2020 stock market crash*. Wikipedia. https://en.wikipedia.org/wiki/2020_stock_market_crash

- [53] Aloosh, A., Choi, H. E., & Ouzan, S. (2021). On the Efficiency of Meme Stocks. *Available at SSRN 3839832*.
- [54] Chung, J. M., Choe, H., & Kho, B. C. (2009). The impact of day-trading on volatility and liquidity. *Asia-Pacific Journal of Financial Studies*, 38(2), 237-275.
- [55] Omar Bassal, C. F. A. (2008). *Swing trading for dummies*. John Wiley & Sons.
- [56] Turner, T. (2007). *A Beginner's Guide To Day Trading Online 2nd Edition*. Simon and Schuster.
- [57] Griffis, M., & Epstein, L. (2009). *Trading For Dummies®*. John Wiley & Sons.
- [58] Bulkowski, T. N. (2012). *Fundamental Analysis and Position Trading: Evolution of a Trader* (Vol. 605). John Wiley & Sons.
- [59] Aitken, M., Cumming, D., & Zhan, F. (2015). High frequency trading and end-of-day price dislocation. *Journal of Banking & Finance*, 59, 330-349.
- [60] Highly Profitable Stochastic + RSI + MACD Trading Strategy (Proven 100x). (2021, March 24). [Video]. YouTube.
https://www.youtube.com/watch?v=hh3BKTfE1dc&t=514s&ab_channel=DataTrader
- [61] Dourra, H., & Siy, P. (2002). Investment using technical analysis and fuzzy logic. *Fuzzy sets and systems*, 127(2), 221-240.
- [62] How To Combine Trading Indicators For Money-Making Systems & Strategies. (2020, January 20). [Video]. YouTube.
https://www.youtube.com/watch?v=QdbKApfwF-g&ab_channel=TheSecretMindset

13 Appendix

	Training data	Testing data	Yager Network (DCT)	Fuzzy-LSTM (DCT) 100-100-100	Fuzzy-LSTM (DCT) Optimized for fold 1 162-189-170	Fuzzy-LSTM (MLVQ) Optimized for fold 1 162-189-170	Fuzzy-LSTM (PFKP) Optimized for fold 1 162-189-170	Pure LSTM Optimized for fold 1 162-189-170	Fuzzy-MLP (DCT) Optimized for fold 1 173-187-18	Fuzzy-MLP (MLVQ) Optimized for fold 1 173-187-18			
			RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	
Recall ability training - 100%, testing - 100%	100% (fold 1)	100% (fold 1)	0.1596	0.6945	0.0483	0.8950	0.0391	0.9150	0.0529	0.8865	0.0406	0.9117	0.0050
	100% (fold 2)	100% (fold 2)	0.2673	0.5872	0.0816	0.8129	0.0655	0.8459	0.0881	0.7981	0.0869	0.8008	0.0036
	100% (fold 3)	100% (fold 3)	0.0959	0.6640	0.0631	0.7752	0.0589	0.7899	0.0681	0.7574	0.0937	0.6662	0.0044
	100% (fold 4)	100% (fold 4)	0.1910	0.6480	0.0794	0.8527	0.0667	0.8762	0.0764	0.8583	0.0667	0.8762	0.0041
	100% (fold 5)	100% (fold 5)	0.2537	0.5133	0.0872	0.8339	0.0701	0.8665	0.0866	0.8694	0.0828	0.8422	0.0051
	100% (fold 6)	100% (fold 6)	0.0766	0.8105	0.0637	0.8422	0.0732	0.8188	0.0479	0.8815	0.0526	0.8658	0.0008
Verification training - 100%, testing - 100%													
Generalization ability training - 20%, testing - 80%	20% (fold 1)	80%	0.2363	0.4879	0.1528	0.6692	0.1368	0.7038	0.1304	0.6176	0.1506	0.6739	0.0054
	20% (fold 2)	80%	0.1848	0.6078	0.1050	0.7762	0.1056	0.7748	0.1136	0.7579	0.1660	0.6462	0.0039
	20% (fold 3)	80%	0.2765	0.5368	0.1927	0.6089	0.1886	0.6168	0.163	0.6691	0.1763	0.6422	0.0085
	20% (fold 4)	80%	0.2442	0.4357	0.1430	0.6673	0.1156	0.7309	0.1195	0.7238	0.1179	0.7257	0.0017
	20% (fold 5)	80%	0.2071	0.5425	0.1662	0.6310	0.1591	0.6467	0.2206	0.5103	0.1588	0.6474	0.0033
	Verfication training - 16.6%, testing - 83.4%	16.6% (fold 6)	83.4%	0.2454	0.4705	0.1210	0.7378	0.1306	0.7099	0.2137	0.6537	0.1193	0.7414
Verification training - 83.4%, testing - 16.6%													
Generalization ability training - 80%, testing - 20%	80%	20% (fold 1)	0.2191	0.5307	0.0395	0.9139	0.0375	0.9183	0.1370	0.8194	0.1358	0.8221	0.0002
	80%	20% (fold 2)	0.1793	0.5898	0.0470	0.8921	0.0453	0.8951	0.1390	0.8105	0.1438	0.7994	0.0059
	80%	20% (fold 3)	0.2092	0.5615	0.0375	0.8661	0.04010	0.8571	0.1319	0.7862	0.1393	0.7559	0.0003
	80%	20% (fold 4)	0.2480	0.5887	0.0749	0.8610	0.0496	0.9079	0.0618	0.8540	0.1417	0.7727	0.0004
	80%	20% (fold 5)	0.2800	0.4630	0.0472	0.9100	0.0465	0.9113	0.0501	0.9045	0.0474	0.8997	0.0005
	Verfication training - 83.4%, testing - 16.6%	16.6% (fold 6)	0.1697	0.5845	0.0470	0.8837	0.0374	0.9072	0.1309	0.8235	0.1344	0.8148	0.0002

Fuzzy-LSTM (DCT)	Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP	Fuzzy-LSTM (DCT)	Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP	Fuzzy-LSTM (DCT)	Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP
Optimized for fold 2	Optimized for fold 3	Optimized for fold 4									
149-190-182	149-190-182	70-81-15	70-81-15	77-145-132	77-145-132	155-199-32	155-199-32	125-183-168	125-183-168	125-183-168	125-183-168
RMSE	R2										
0.0530	0.8847	0.0005	0.9988	0.0543	0.8835	0.0446	0.9039	0.0536	0.8834	0.0052	0.9888
0.0639	0.8535	0.0025	0.9940	0.0981	0.7749	0.0802	0.8160	0.0742	0.8299	0.0040	0.9907
0.0713	0.7459	0.0007	0.9974	0.0478	0.8325	0.0369	0.8706	0.0615	0.7809	0.0028	0.9898
0.1019	0.8110	0.0015	0.9971	0.1294	0.7615	0.0846	0.8441	0.0727	0.8652	0.0191	0.9646
0.0964	0.8165	0.0018	0.9965	0.1604	0.6924	0.1069	0.7949	0.0786	0.8503	0.002	0.9961
0.0512	0.8732	0.0018	0.9955	0.0719	0.8239	0.0451	0.8894	0.0498	0.8766	0.0212	0.9481
0.1885	0.5919	0.0055	0.9879	0.1828	0.6039	0.1102	0.7610	0.1601	0.6533	0.0027	0.9940
0.1001	0.7866	0.0016	0.9964	0.1113	0.7638	0.0880	0.8132	0.2037	0.5659	0.0022	0.9951
0.1833	0.6281	0.0045	0.9908	0.2021	0.5914	0.1141	0.7693	0.1727	0.6495	0.0084	0.9829
0.1237	0.7122	0.0061	0.9656	0.1603	0.6296	0.0932	0.7846	0.1924	0.5524	0.0022	0.9948
0.1294	0.7126	0.0012	0.9972	0.1556	0.6563	0.0917	0.7974	0.2150	0.5226	0.0022	0.9951
0.1060	0.7702	0.0028	0.9938	0.1212	0.7383	0.0883	0.8094	0.1220	0.7365	0.0032	0.9929
0.0375	0.9183	0.0032	0.9930	0.0631	0.8218	0.0640	0.8629	0.0382	0.9689	0.0017	0.9963
0.0437	0.8998	0.0031	0.9927	0.1042	0.7609	0.0811	0.8139	0.0485	0.8988	0.0004	0.9988
0.0373	0.8659	0.0005	0.9981	0.0741	0.7404	0.0480	0.8318	0.0373	0.8670	0.0002	0.9992
0.0501	0.9071	0.0017	0.9968	0.1131	0.7915	0.0850	0.8432	0.0527	0.9021	0.0012	0.9976
0.0465	0.9114	0.0007	0.9986	0.1272	0.7560	0.1006	0.8069	0.0459	0.9125	0.0019	0.9962
0.0378	0.9063	0.0004	0.9989	0.0623	0.8475	0.0477	0.8832	0.0359	0.9111	0.0004	0.9967
0.0378	0.9063	0.0004	0.9989	0.0623	0.8475	0.0477	0.8832	0.0359	0.9111	0.0004	0.9967

Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP	Fuzzy-LSTM (DCT)	Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP	Fuzzy-LSTM (DCT)	Pure LSTM	Fuzzy-MLP (DCT)	Pure MLP	Fuzzy-MLP (DCT)	Pure MLP	
Optimized for fold 4 125-183-168	Optimized for fold 4 72-175-50	Optimized for fold 4 125-183-168	Optimized for fold 4 102-191-129	Optimized for fold 5 102-191-129	Optimized for fold 5 199-162-198	Optimized for fold 5 199-162-198	Optimized for fold 5 199-162-198	Optimized for fold 6 123-180-156	Optimized for fold 6 123-180-156	Optimized for fold 6 134-161-63			
RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
0.0021	0.9954	0.0620	0.8672	0.0419	0.9102	0.0507	0.8896	0.0026	0.9943	0.0653	0.8578	0.0414	0.9113
0.0016	0.9962	0.1104	0.7468	0.0795	0.8176	0.0643	0.8520	0.0020	0.9952	0.0984	0.7744	0.0879	0.7983
0.0006	0.9976	0.0833	0.7082	0.0362	0.8731	0.0822	0.7070	0.0008	0.9968	0.0455	0.8405	0.0332	0.8836
0.0006	0.9987	0.0962	0.8226	0.0759	0.8600	0.0857	0.8410	0.0022	0.9957	0.1389	0.7439	0.0759	0.8600
0.0083	0.9841	0.1343	0.7423	0.0989	0.8102	0.0678	0.8709	0.0036	0.993	0.1122	0.7848	0.0861	0.8348
0.0012	0.997	0.0632	0.8453	0.0387	0.9052	0.0766	0.8105	0.0009	0.9976	0.0701	0.8284	0.0398	0.9024
												0.0590	0.8540
												0.0004	0.9989
												0.0630	0.8457
												0.0383	0.9062
0.0043	0.9906	0.1812	0.6074	0.1399	0.6967	0.1494	0.6765	0.0058	0.9872	0.1728	0.6256	0.1284	0.7217
0.0065	0.9859	0.1944	0.5876	0.0787	0.8329	0.1018	0.7483	0.0033	0.9929	0.1489	0.6840	0.0862	0.8171
0.0087	0.9821	0.1380	0.7210	0.1136	0.7702	0.1903	0.6439	0.0072	0.9853	0.1417	0.7135	0.1216	0.7541
0.0006	0.9983	0.1387	0.6795	0.0955	0.7792	0.1241	0.7111	0.0023	0.9944	0.2268	0.4758	0.1029	0.7622
0.0016	0.9963	0.1599	0.6467	0.0823	0.8118	0.0472	0.9100	0.0021	0.9951	0.1219	0.7662	0.0898	0.8016
0.0022	0.9951	0.1321	0.7148	0.0902	0.8053	0.1309	0.7163	0.0031	0.9932	0.1222	0.7363	0.0863	0.8137
												0.1079	0.7661
												0.0028	0.9937
												0.1326	0.7137
												0.0929	0.7993
0.0002	0.9995	0.0771	0.8347	0.0568	0.8782	0.0525	0.8857	0.0014	0.9969	0.1321	0.7170	0.0516	0.8892
0.0006	0.9984	0.1074	0.7536	0.0724	0.8339	0.0451	0.8966	0.0002	0.9993	0.1098	0.7481	0.0771	0.8232
0.0012	0.9954	0.0711	0.7509	0.0437	0.8468	0.0381	0.8639	0.0001	0.9995	0.0797	0.7206	0.0408	0.8571
0.0003	0.9993	0.1087	0.7995	0.0809	0.8508	0.04605	0.9146	0.0007	0.9986	0.1134	0.7910	0.0842	0.8447
0.0011	0.9978	0.1303	0.7500	0.0911	0.8252	0.0458	0.9126	0.0009	0.9982	0.1174	0.7747	0.0943	0.8190
												0.0458	0.9126
												0.0018	0.9964
												0.1279	0.7546
												0.0863	0.8344
0.0000	0.9997	0.1019	0.7504	0.0409	0.8998	0.0379	0.9062	0.0003	0.9991	0.0747	0.8171	0.0405	0.9008
												0.0282	0.9301
												0.0017	0.9957
												0.0888	0.7826
												0.0424	0.896

Figure 13 Result of traffic flow dataset optimized for all 6 folds