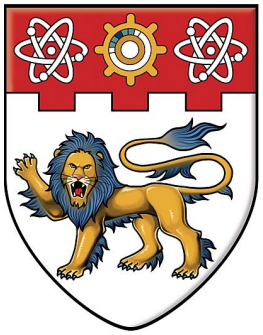


NANYANG TECHNOLOGICAL UNIVERSITY

Recurrent Neuro-Fuzzy Parallel System (RNFPS)

by
Koh Liang Jing

SCSE20-0787
School of Computer Science and Engineering
2020 - 2021

NANYANG TECHNOLOGICAL UNIVERSITY

Recurrent Neuro-Fuzzy Parallel System (RNFPS)

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

In this paper,

Table of Contents

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 Modified Yager FNN tagged with LSTM tested with traffic data
- Chapter 8: Implementation of Modified Yager FNN tagged with LSTM tested with stock market data (WIP)
- Chapter 9: Experiment of the different trading strategies (WIP)
- Chapter 10: Implementation of Modified Yager FNN tagged with LSTM and the selected trading strategy tested with stock market data (WIP)
- Chapter 11: Conclusion (WIP)

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:

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

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:

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

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]

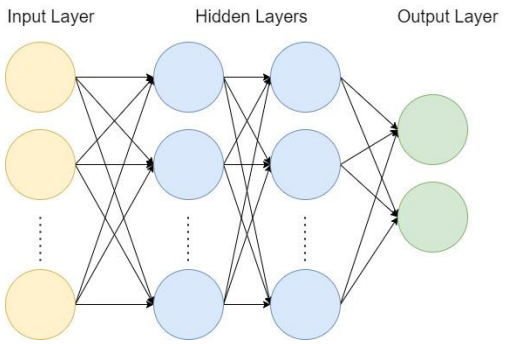


Figure 2.1 Multi-Layer Perceptron with 2 hidden layers

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.

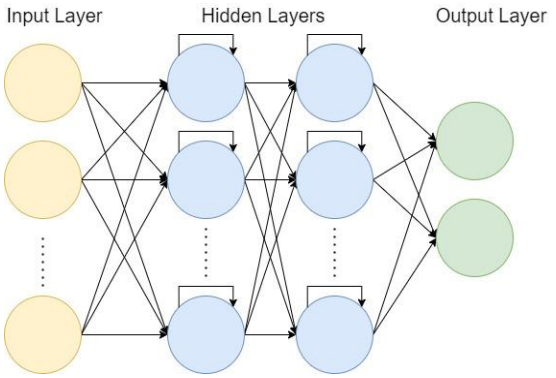


Figure 2.2 Recurrent Neural Network with 2 hidden layers - adapted from [9]

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.

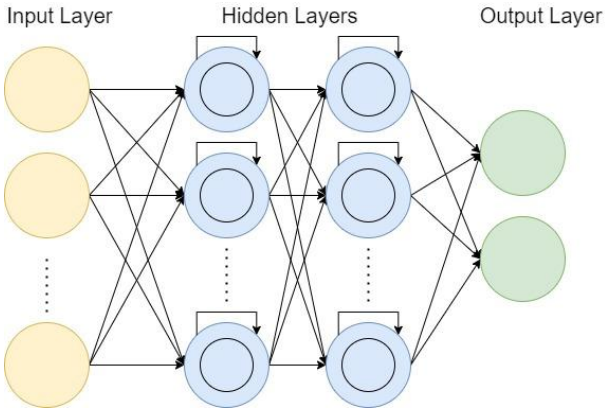


Figure 2.3 Long Short Term Memory with 2 hidden layers - adapted from [9]

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 fate 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

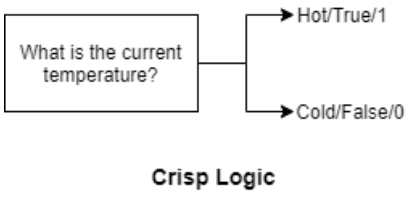


Figure 2.4 An example of Crisp Logic

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

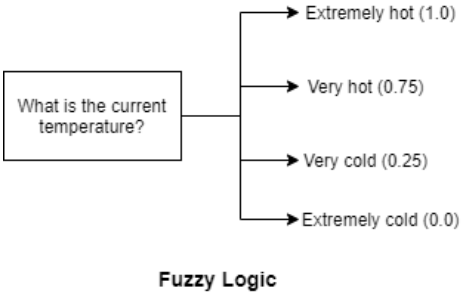


Figure 2.5 An example of Fuzzy Logic

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.

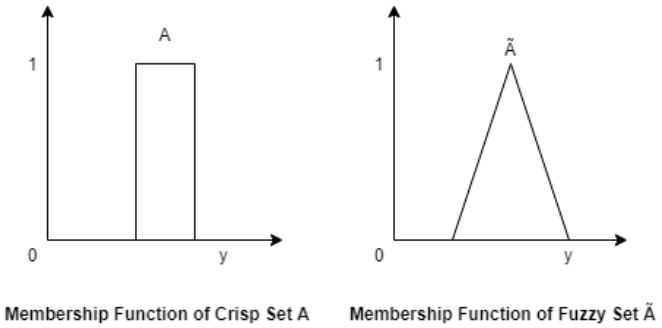


Figure 2.6 Graph of membership function for Crisp Set and Fuzzy Set

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.

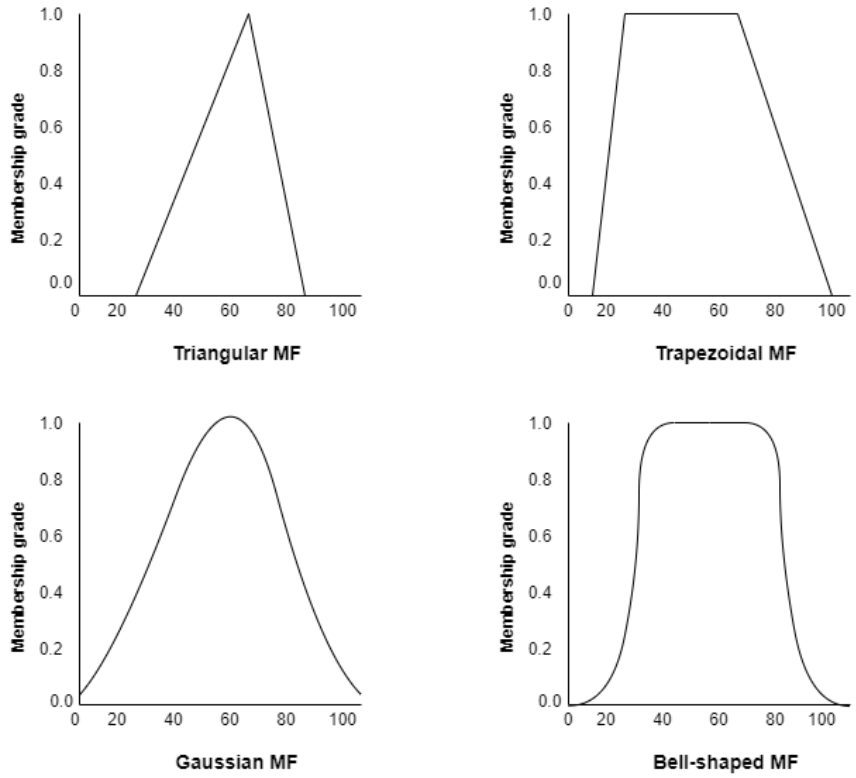


Figure 2.7 Graph of Triangular, Trapezoidal, Gaussian, and Bell-Shaped membership function

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	A triangular MF is specified by three parameters {a, b, c} as follows: $0, \quad x \leq a.$

	$$triangle(x; a,\ b,\ c) = \begin{cases} \frac{x-a}{b-a}, & a \le x \le b. \\ \frac{c-x}{c-b}, & b \le x \le c. \\ 0, & c \le x. \end{cases} \quad (2\text{-}6)$$
	An alternative expression is $$triangle(x; a,\ b,\ c) = max(min(\frac{x-a}{b-a},\ \frac{c-x}{c-b}),\ 0) \quad (2\text{-}7)$$
Trapezoidal	A trapezoidal MF is specified by four parameters {a, b, c, d} as follows: $$trapezoid(x; a,\ b,\ c,\ d) = \begin{cases} 0, & x \le a. \\ \frac{x-a}{b-a}, & a \le x \le b. \\ 1, & b \le x \le c. \\ \frac{d-x}{d-c}, & c \le x \le d. \\ 0, & d \le x. \end{cases} \quad (2\text{-}8)$$
	An alternative expression is $$trapezoid(x; a,\ b,\ c,\ d) = max(min(\frac{x-a}{b-a},\ 1,\ \frac{d-x}{d-c}),\ 0) \quad (2\text{-}9)$$
Gaussian	A gaussian MF is specified by two parameters {c,σ} as follows:

	$gaussian(x; c, \sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$ (2-10)
Bell-shaped	A bell-shaped Function is specified by three parameters {a,b,c} as follows: $bell(x; a, b, c) = \frac{1}{1+\left\|\frac{x-c}{a}\right\|^{2b}}$ (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

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

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

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

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.

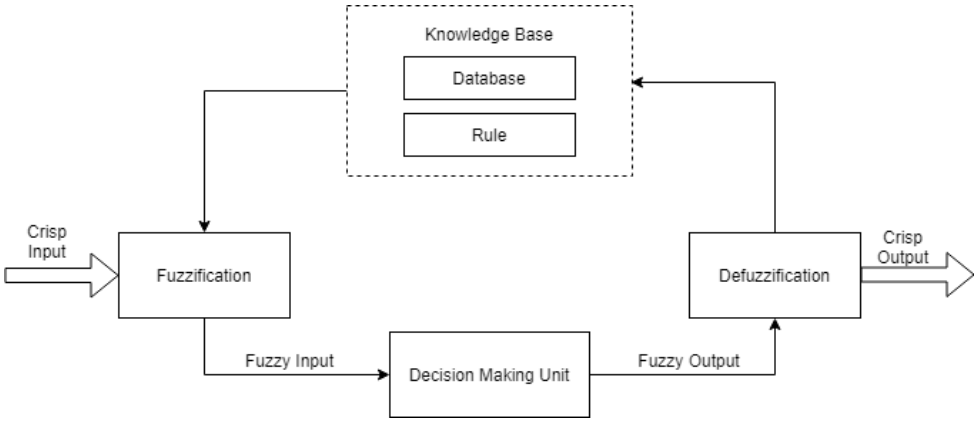


Figure 2.8 Flowchart of a Fuzzy Inference System

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

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

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 f 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

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

where

n refers to the number of data vectors

2. $for\ k = 1..n$

2.1. Find the winner i

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

2.2. Update the weight of the winner i

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

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

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

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

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

where

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

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

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

$$w_i = \frac{v_i - v_{closest}}{\sigma} \tag{2-20}$$

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

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

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 (PFKP)

Pseudo Fuzzy Kohonen Partition (PFKP) [21] was created based on MLVQ algorithm, where it is able to directly create fitting membership functions from the training data. PFKP generates a triangular and trapezoidal-shaped membership function, which is highly utilized due to its simplicity in computation and storage. PFKP 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 PFKP algorithm is as follows:

1. Initialize training iteration and weight

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

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

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

3.2. Update the weight of the winner

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

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

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

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. $for\ all\ i = 1..c$ 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

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

8.2. Update pseudo weight of winner i

$$\varphi_i = \varphi_i + \eta(x_k + \varphi_i) \qquad (2\text{-}23)$$

8.3. Update the four points of Trapezoidal Fuzzy Number

$$\alpha_i = min(\alpha_i, x_k) \qquad (2\text{-}24)$$

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

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

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

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.

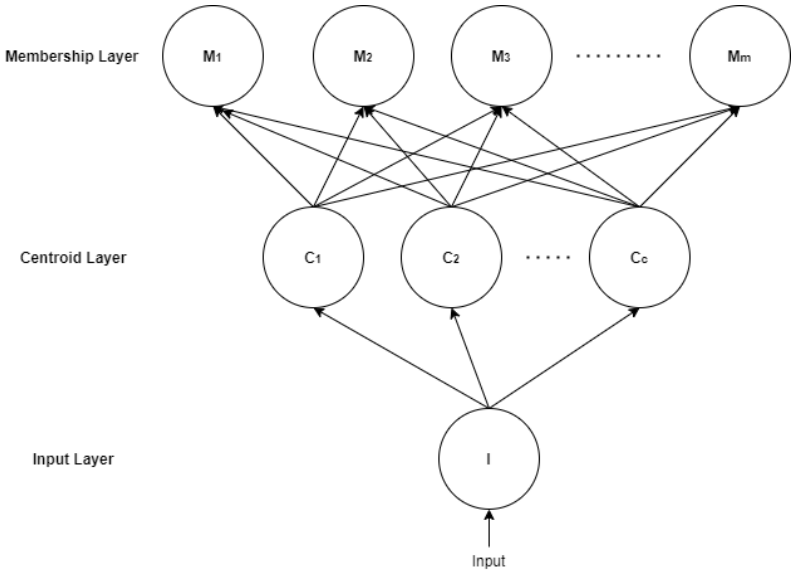


Figure 2.9 Architecture of Neural Network for Discrete Clustering Technique

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

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

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

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

3.2. Update the weight of the winner

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

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

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

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

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

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

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

10. Compute the learning rate for each individual cluster

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

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

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

11.2. Find the winning neuron j in the Membership Layer

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

11.3. Compute the gravitational ratio g_{jk}

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

$$g_{jk} = \frac{|x_k - X_j|}{\sigma} \tag{2-29}$$

where

σ refers to the interval between any two adjacent discrete points

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

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

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

11.5.1. Compute the gravitational ratio g_{pk}

$$d = |j - p| \tag{2-31}$$

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

where

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

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

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

12. Normalize all weights

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

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

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

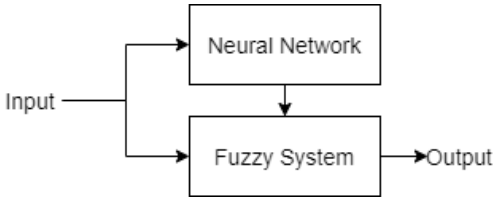


Figure 2.10 Parallel fuzzy neural network system - adapted from [14]

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

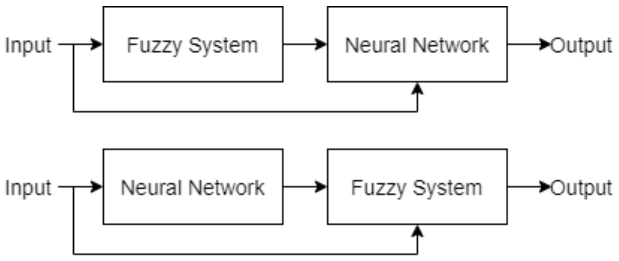


Figure 2.11 Cascading fuzzy neural network systems - adapted from [14]

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:

$$If\ X_1\ is\ A_1\ and\ X_2\ is\ A_2\ and\ \dots\ and\ X_n\ is\ A_n\ then\ Y\ is\ B$$

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]

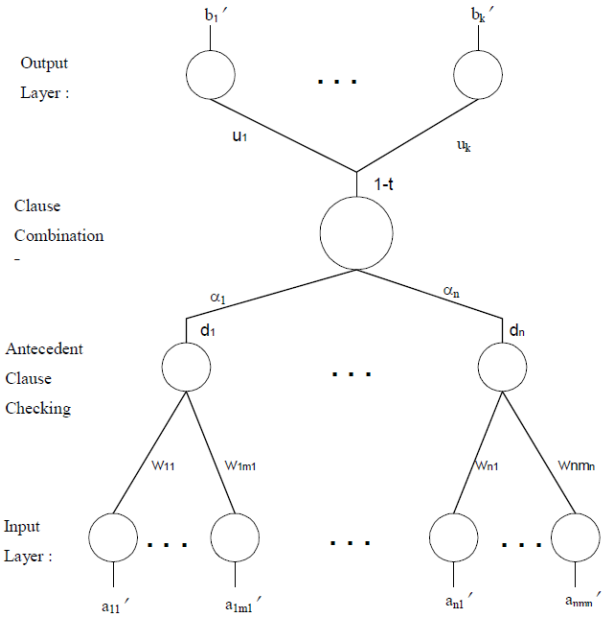


Figure 2.12 Structure of Fuzzy Logic Inference Neural Network - adapted from [36]

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.

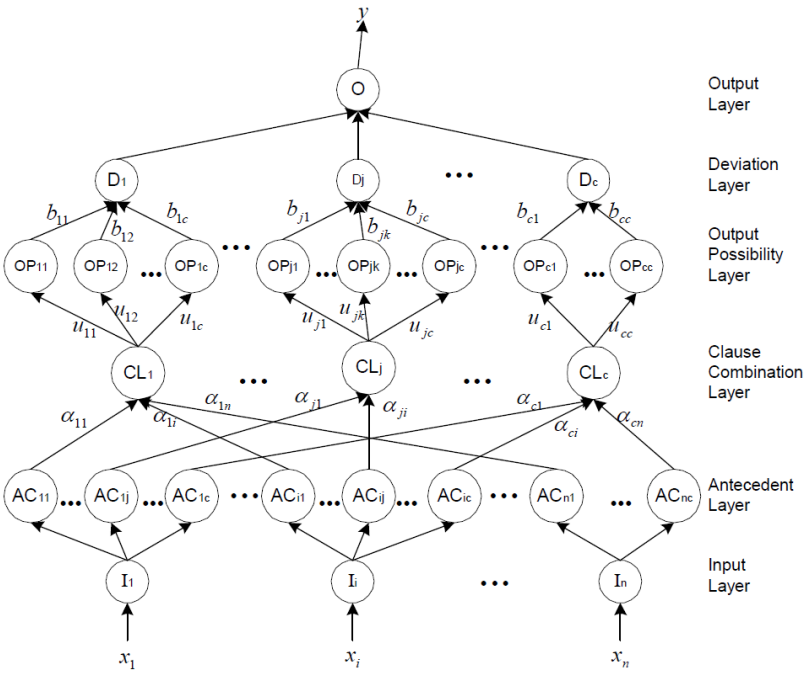


Figure 2.13 Structure of Yager-PC FNN - adapted from [35]

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.

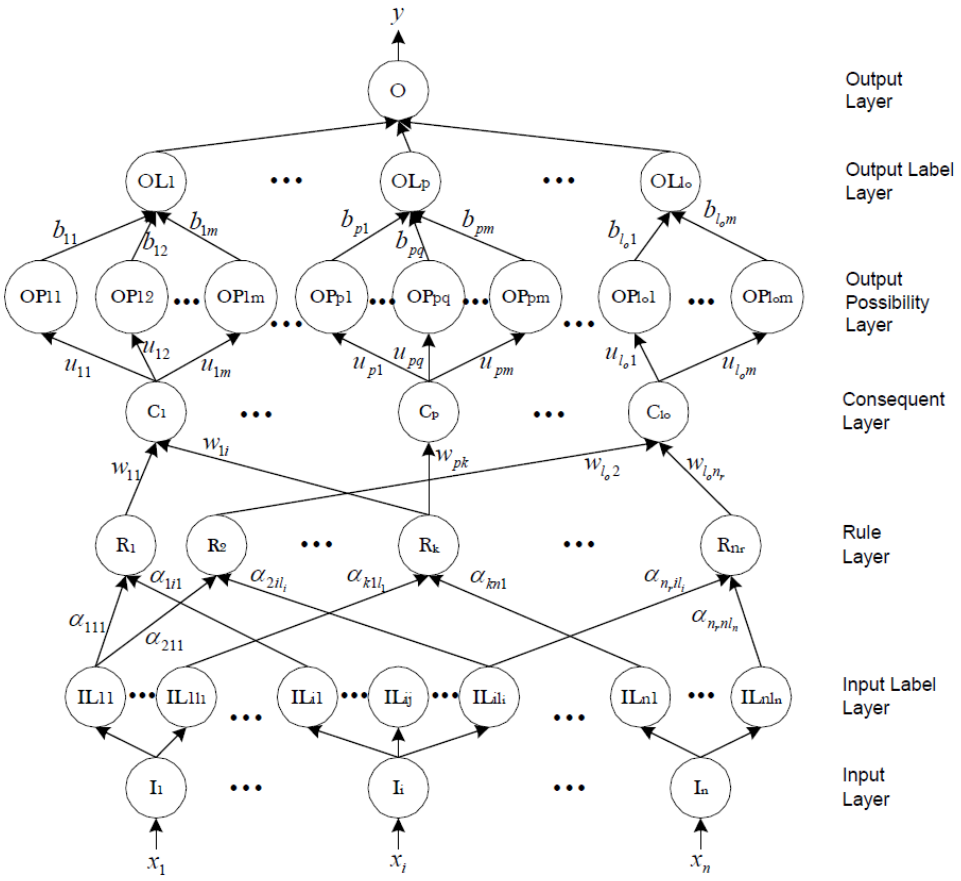


Figure 2.14 Structure of Modified Yager FNN - adapted from [35]

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]

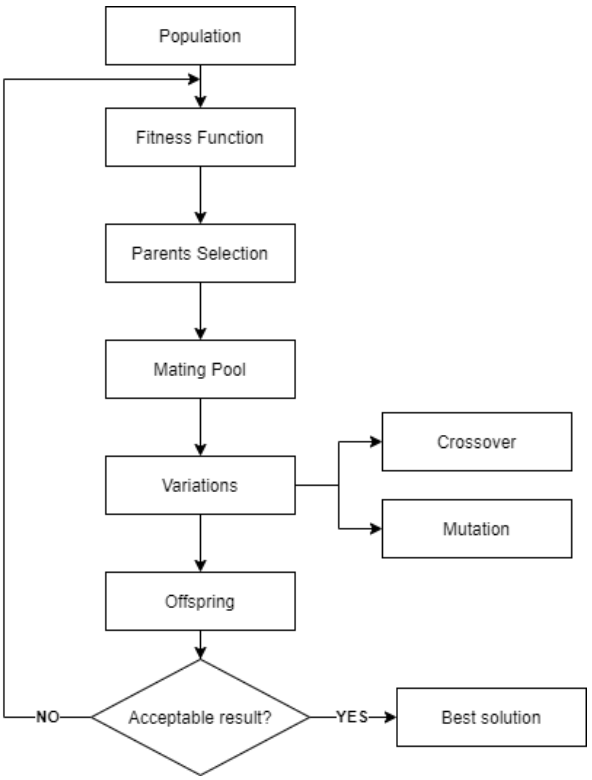


Figure 2.15 Flowchart of Genetic Algorithm

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.

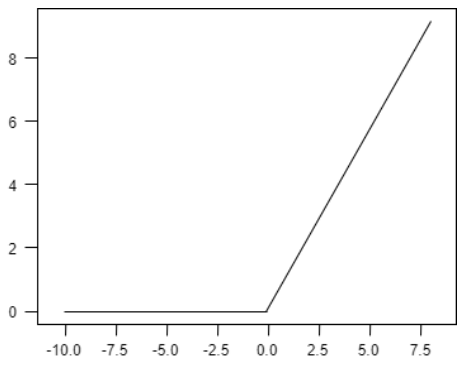


Figure 2.16 Graph of input vs output for a ReLu activation function

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:

$$max(0, x) \tag{2-34}$$

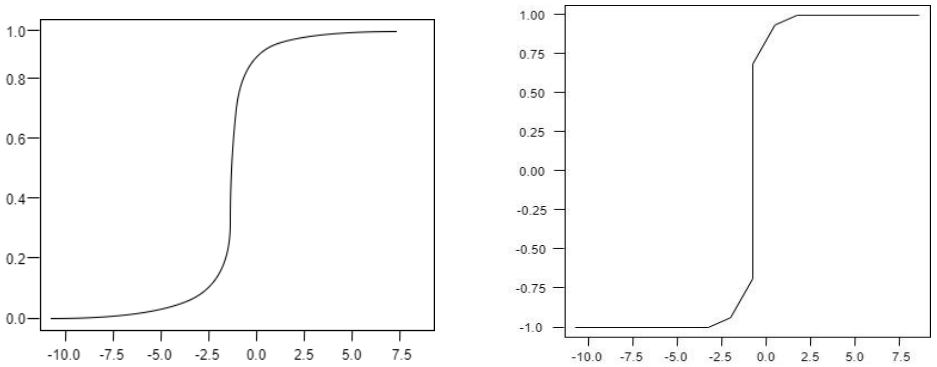


Figure 2.17 Graph of input vs output for a Sigmoid and a Tanh activation function

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:

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

The Tanh activation function is calculated as follows:

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

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. 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.

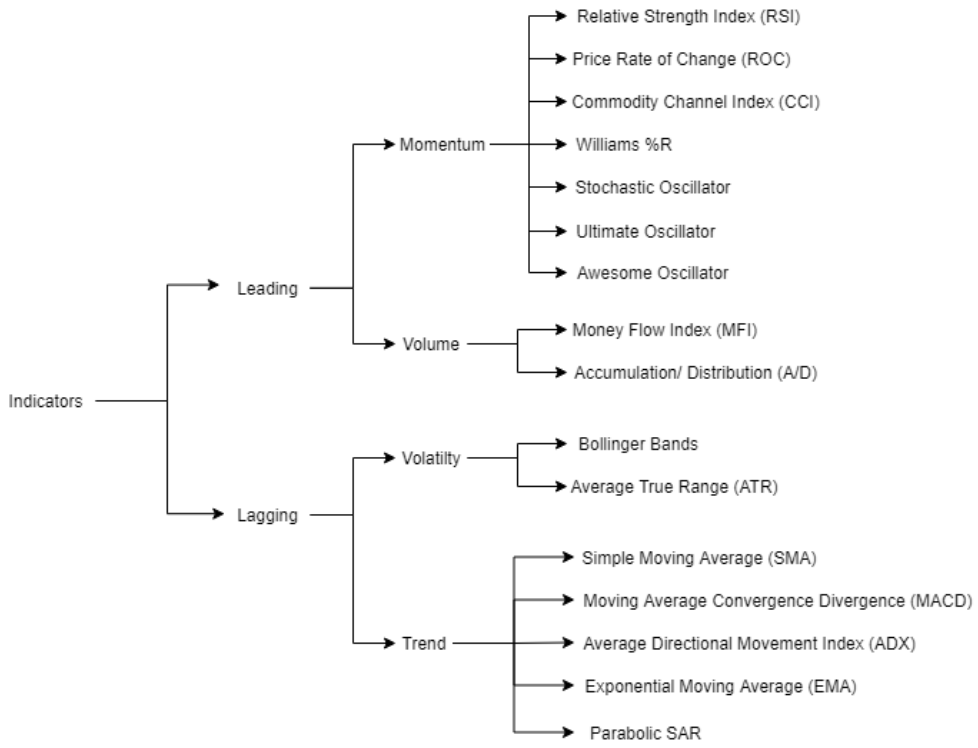


Figure 2.18 Technical indicators separated by categories

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:

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

where

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

$closingprice_{p-n}$ 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:

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

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:

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

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

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

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:

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

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:

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

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:

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

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:

$$k = \frac{2}{N+1} \tag{2-45}$$

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

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:

$$MACD = 12\,period\,EMA - 26\,period\,EMA \tag{2-47}$$

where

Nrefers 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, Virginca) 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.

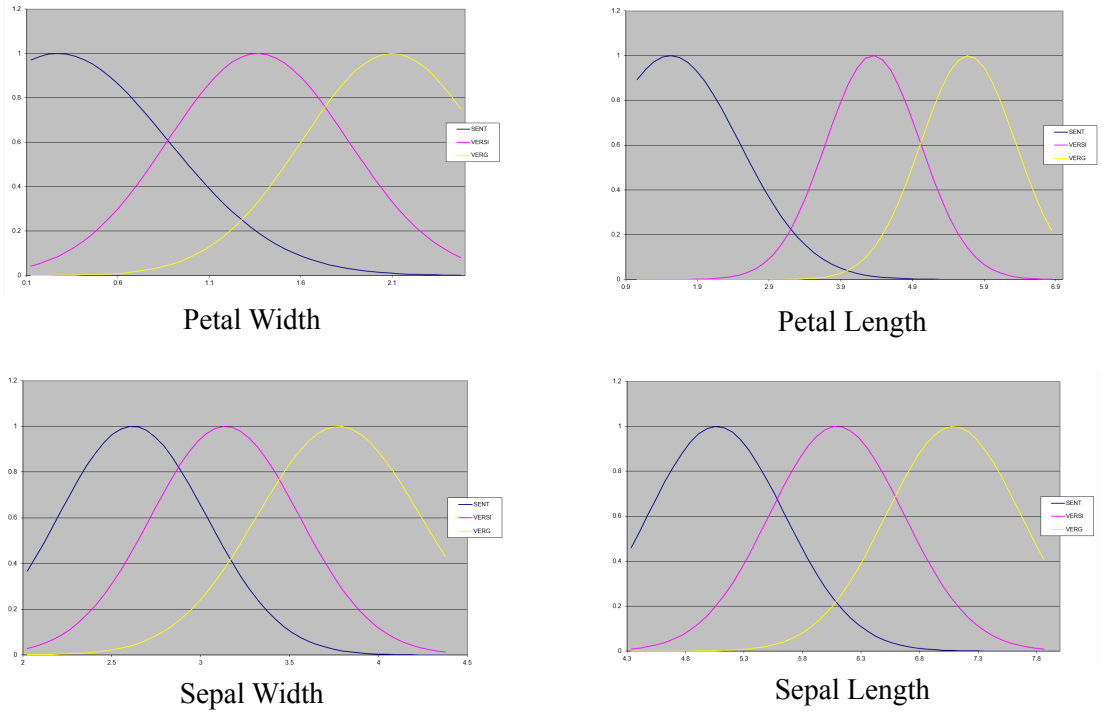


Figure 3.1 Membership Functions of Iris dataset identified by MLVQ

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.

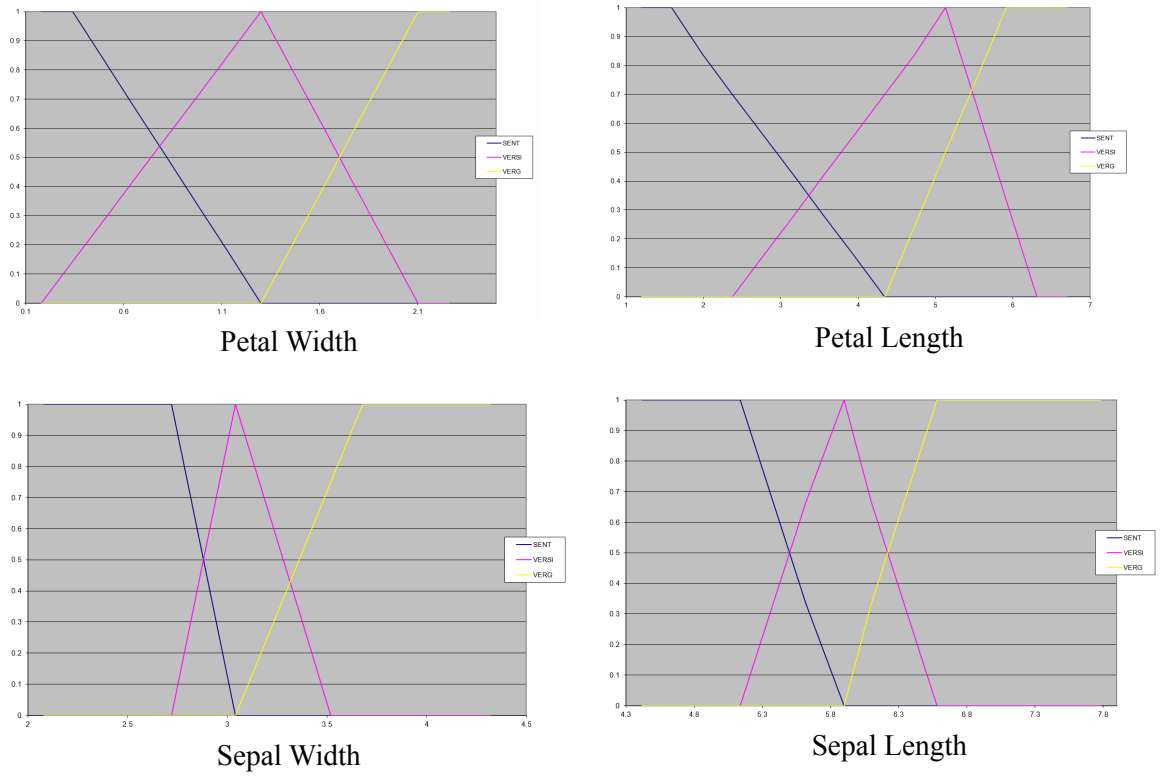


Figure 3.2 Membership Functions of Iris dataset identified by PFKP

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.

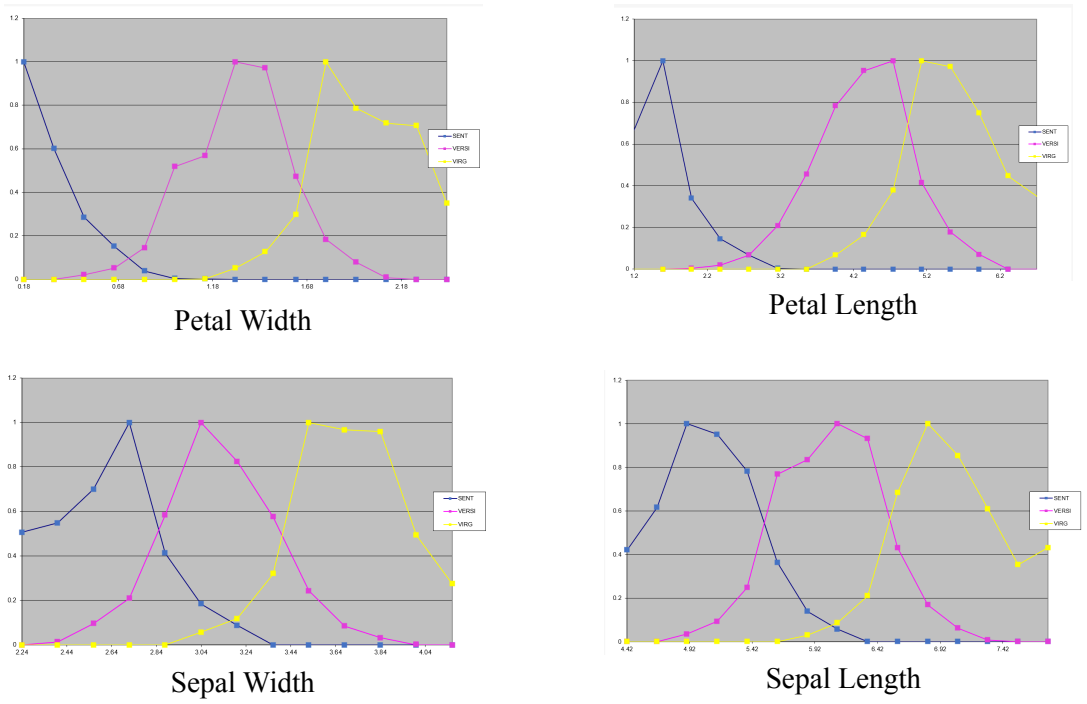


Figure 3.3 Membership Functions of Iris dataset identified by UDCT

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:

$$accuracy = classified / 150$$

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.

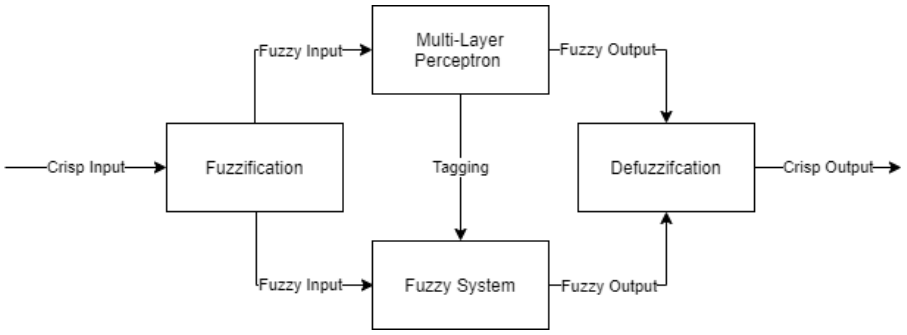


Figure 4.1 Architecture of Yager-PC FNN tagged with Multi-Layer Perceptron

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:

$$if\ x_1\ is\ A\ \wedge\ x_2\ is\ B\ \wedge\ \dots\ \wedge\ x_n\ is\ N\ then\ y\ is\ Z \tag{4.1}$$

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

$$Consequents\ =\ \{y\}$$

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:

$$w_0 = 0 \tag{4-1}$$

$$w(t+1) = w(t) + \Delta w \tag{4-2}$$

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

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.

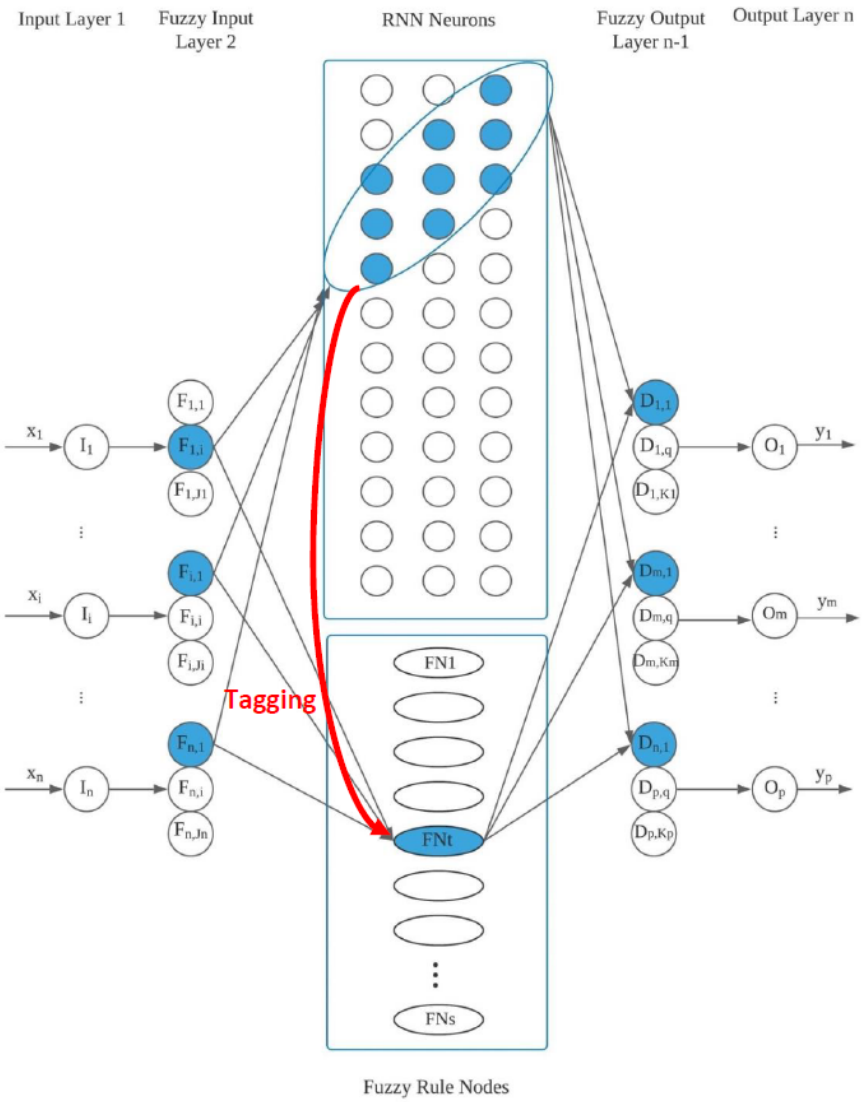


Figure 4.1 Tagging Mechanism - adapted from [47]

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, a 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 $\varepsilon = 0.0005$

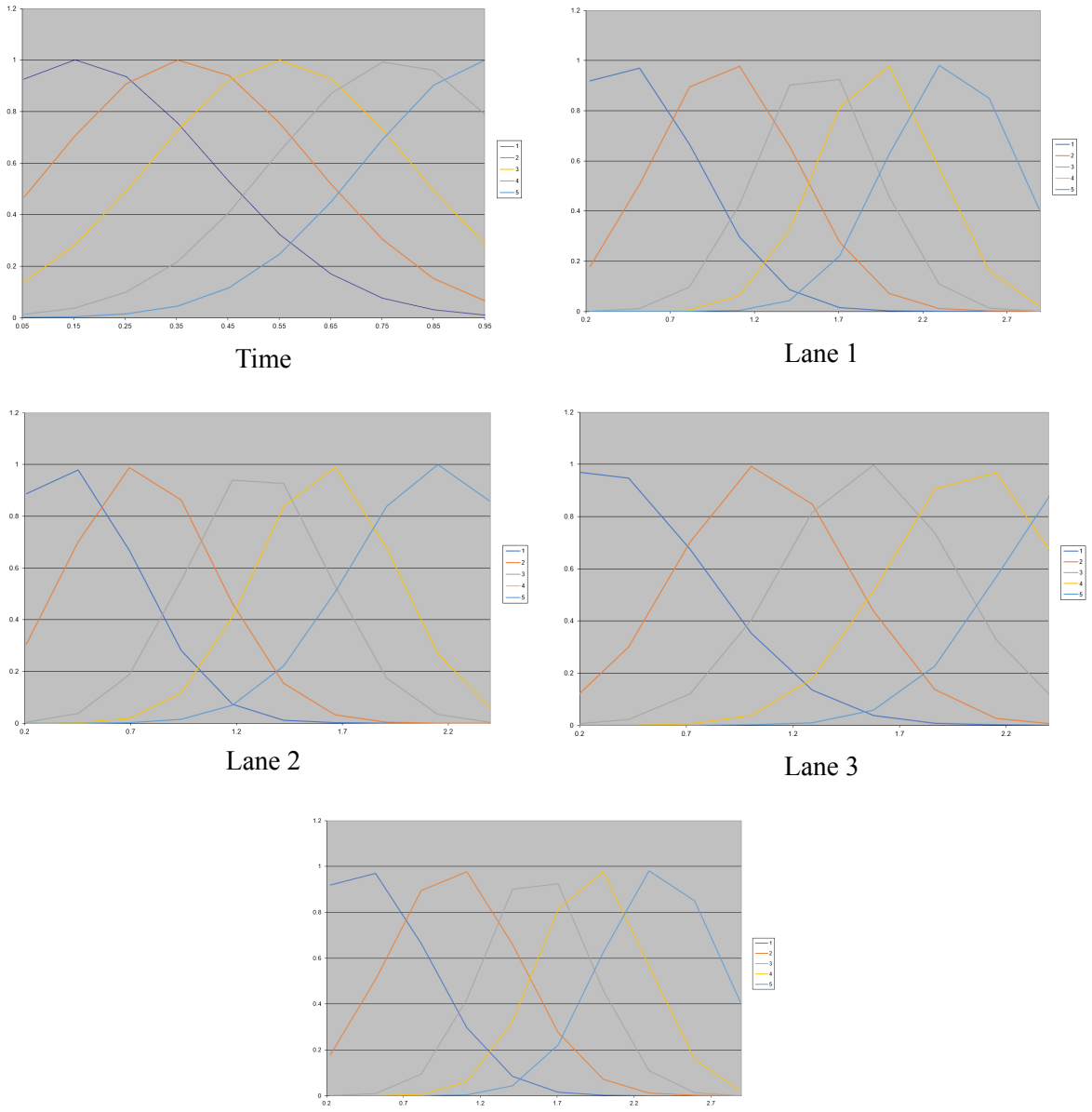


Figure 5.1 Membership Functions of Traffic flow dataset identified by MLVQ

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 $\varepsilon = 0.0005$

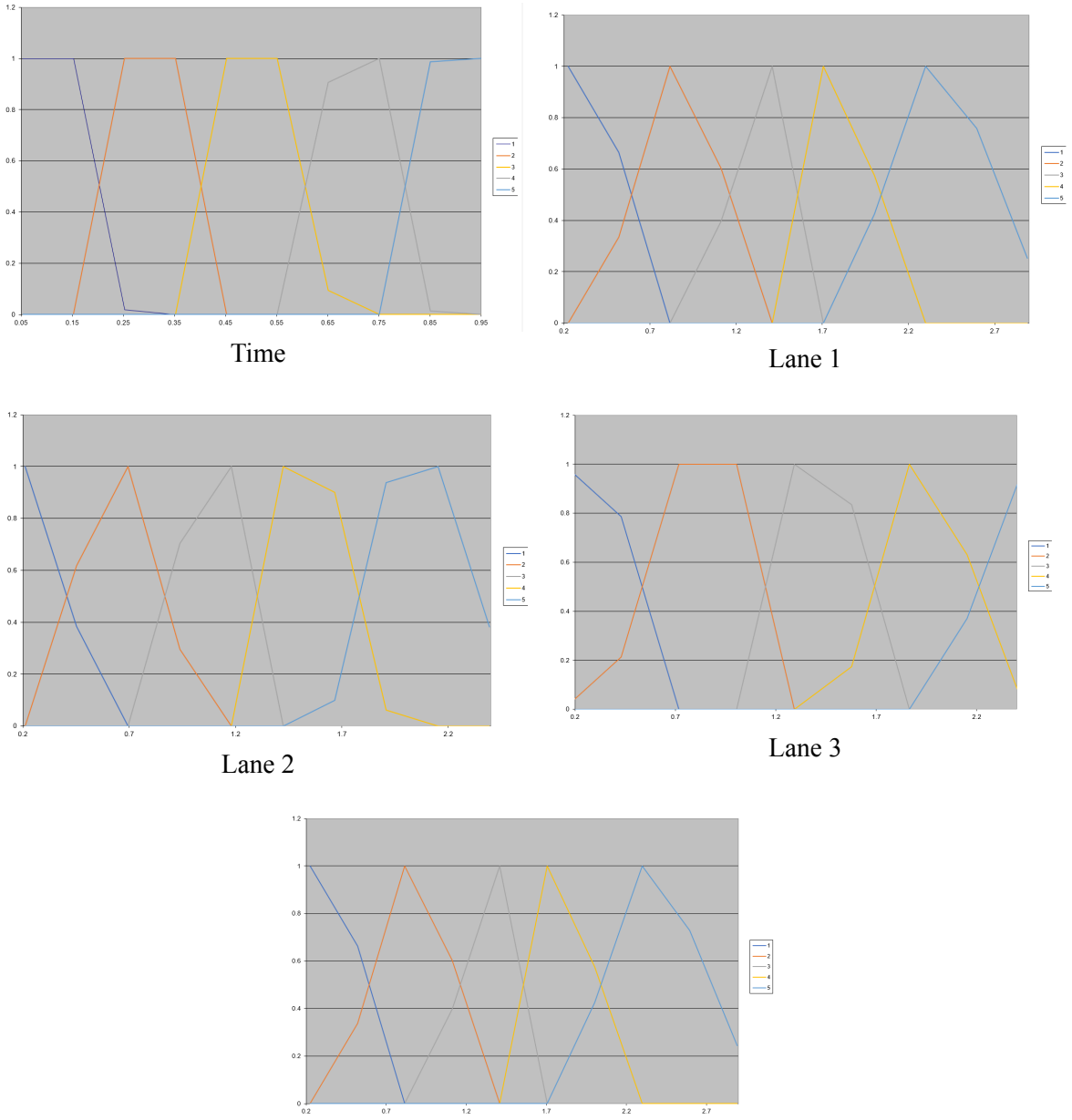


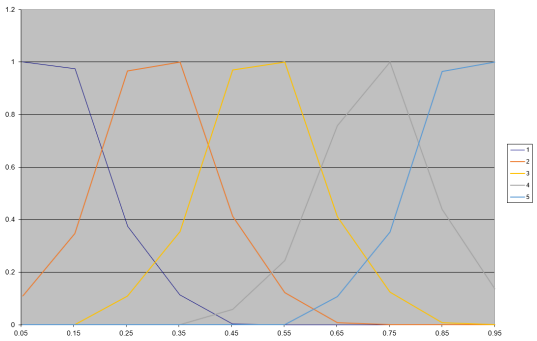
Figure 5.2 Membership Functions of Traffic flow dataset identified by PFKP

5.1.3 Results using UDCT

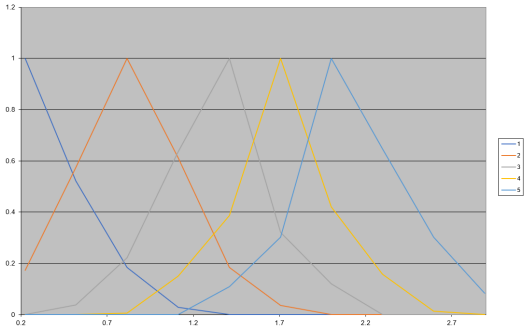
During this experiment, the UDCT algorithm was used on the whole set of data. The following variable was assigned the respective values:

- Learning constant $\lambda = 0.0018$
- Number of cluster 5

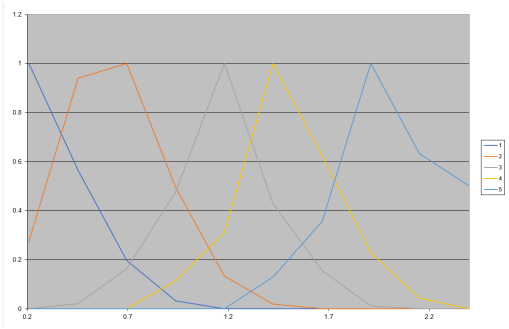
- Discrete point sample point $m = 15$
- Stopping criterion $\varepsilon = 0.0005$



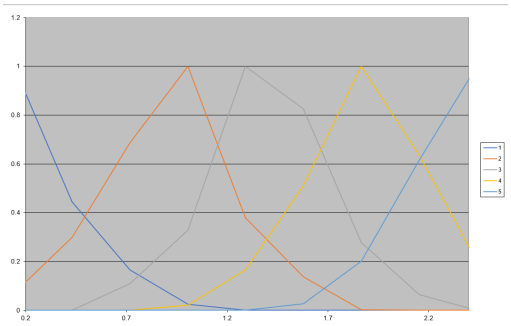
Time



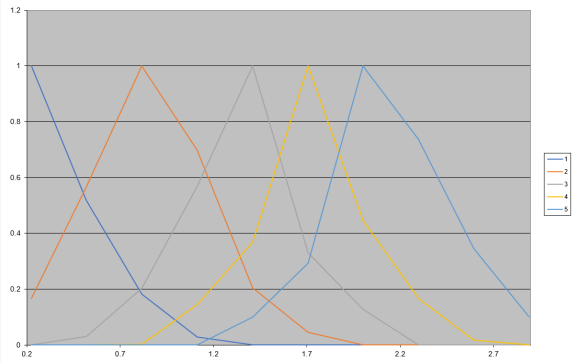
Lane 1



Lane 2



Lane 3



Output

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:

$$accuracy = classified / 150$$

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.

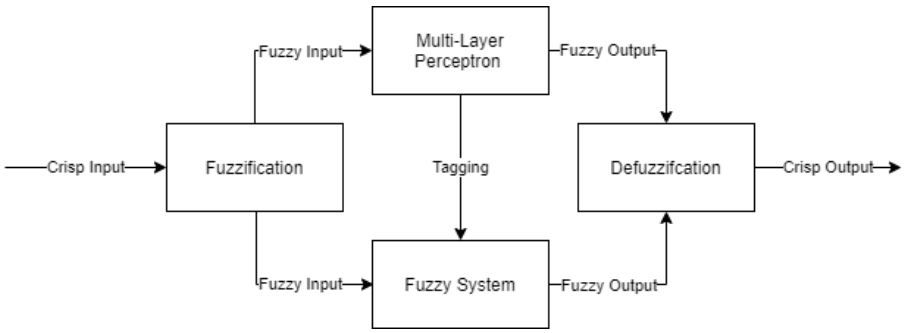


Figure 6.1 Architecture of Modified Yager FNN tagged with Multi-Layer Perceptron

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.

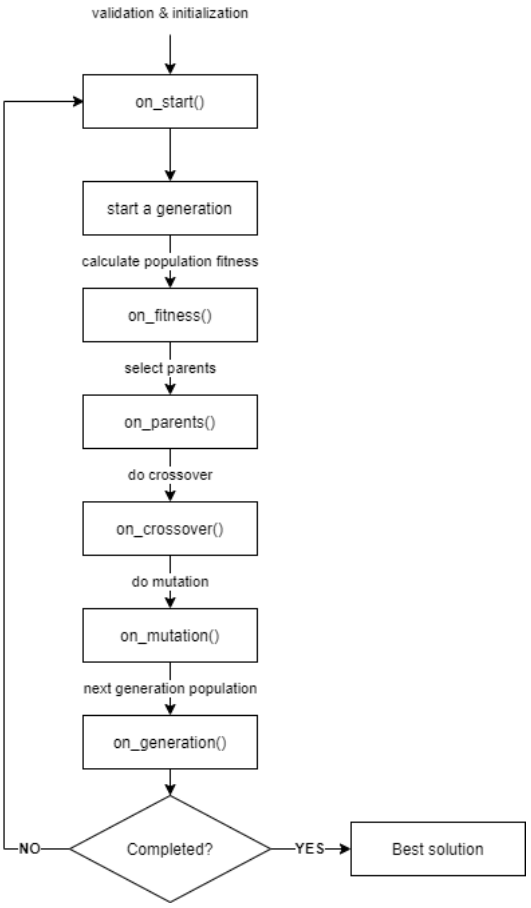


Figure 5.2 flowchart of geneticalgorithm from python library - adapted from [46]

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 a verification.

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**
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 table 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 literature review that neural networks does 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.

Training data	Testing data	Pure MLP	Fuzzy-MLP (UDCT)	Fuzzy-MLP (UDCT)
		Optimized for fold 1	100-100-100	Optimized for fold 1

		173-187-18				173-187-18	
		RMSE	**R2**	**RMSE**	**R2**	**RMSE**	**R2**
fold 1	fold 1	**0.0402**	**0.9138**	0.0790	0.8307	**0.0757**	**0.8377**
fold 2	fold 2	0.0802	0.8160	0.1109	0.7456	0.1053	0.7585
fold 3	fold 3	0.0343	0.8799	0.0810	0.7164	0.0738	0.7414
fold 4	fold 4	0.0863	0.8409	0.1149	0.7883	0.1102	0.7969
fold 5	fold 5	0.093	0.8215	0.1253	0.7596	0.1206	0.7687
fold 6	fold 6	0.0426	0.8956	0.1065	0.7393	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

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.

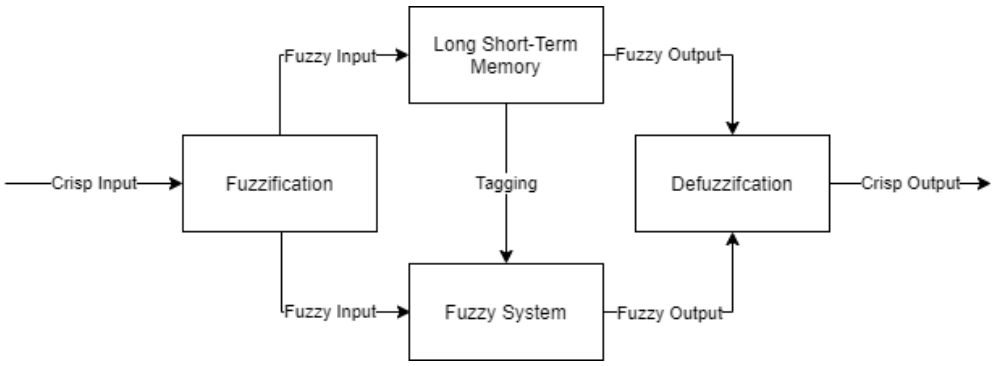


Figure 7.1 Architecture of Recurrent Neuro-Fuzzy Parallel System (RNFPS)

type of recurrent neural network that can learn the order dependence between items in a sequence Long Short-Term Memory (LSTM) is being used instead of MLP as 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. With this, LSTM is more effective when used to predict time-series data compared to the other artificial neural network.

7.2 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.

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

In this section, the result of 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 a verification.

Training data	**Testing data**	**Yager Network (UDCT)**		**RNFPS (UDCT)**		**RNFPS (UDCT)**		**RNFPS (MLVQ)**		**RNFPS (PFKP)**	
				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 7.1 Result of recall ability

Training data	**Testing data**	**Yager Network (UDCT)**		**RNFPS (UDCT)**		**RNFPS (UDCT)**		**RNFPS (MLVQ)**		**RNFPS (PFKP)**	
				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 7.2 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 (PFKP)**	
				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 7.3 Result of the generalization ability (80% training data and 20% testing data)

From table 7.1, 7.2 and 7.3, 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 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 literature review that neural networks does better when trained with more amount of data.

7.4 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.

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 7.4 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 7.5 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 7.6 Result of the generalization ability (80% training data and 20% testing data)

From Table 7.4, 7.5 and 7.6, 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 7.4 and 7.6 that PNFPS compared to pure LSTM have an average of just 8% decrease of accuracy but in Table 7.5, it has an average of 25% decrease. It indicates that PNFPS do not work well with small amounts of training data.

7.5 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	

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-Il 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

[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/ijsrp.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

[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

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	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**	**0.9892**	**0.0531**	**0.8861**	0.0606	0.8682
	100% (fold 2)	100% (fold 2)	0.2673	0.5872	0.0816	0.8129	0.0655	0.8499	0.0881	0.7981	0.0869	0.8008	0.0036	0.9916	0.1257	0.7117	0.0998	0.7710
	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	0.9840	0.0356	0.8751	0.0703	0.7536
	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	0.9922	0.1017	0.8125	0.0764	0.8592
	100% (fold 5)	100% (fold 5)	0.2537	0.5133	0.0872	0.8339	0.0701	0.8665	0.0686	0.8694	0.0828	0.8422	0.0051	0.9901	0.1275	0.7553	0.1105	0.7879
Verification training - 100%, testing - 100%	100% (fold 6)	100% (fold 6)	0.0766	0.8105	0.0637	0.8422	0.0732	0.8188	0.0479	0.8815	0.0526	0.8698	0.0008	0.9978	0.0606	0.8515	0.0444	0.8913
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**	**0.9881**	**0.1513**	**0.6720**	0.1676	0.6367
	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	0.9916	0.1209	0.7435	0.1327	0.7185
	20% (fold 3)	80%	0.2765	0.5368	0.1927	0.6089	0.1888	0.6168	0.163	0.6691	0.1763	0.6422	0.0085	0.9827	0.1404	0.7161	0.1378	0.7213
	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	0.9959	0.1696	0.6079	0.1121	0.7392
	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	0.9924	0.1773	0.6082	0.1248	0.7242
Verification 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	0.0090	0.9800	0.1141	0.7479	0.1021	0.7795
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**	**0.9993**	**0.0757**	**0.8377**	0.1513	0.7900
	80%	20% (fold 2)	0.1793	0.5888	0.0470	0.8921	0.0453	0.8961	0.1390	0.8105	0.1438	0.7994	0.0009	0.9978	0.1053	0.7585	0.1521	0.7803
	80%	20% (fold 3)	0.2092	0.5675	0.0375	0.8661	0.04010	0.8571	0.1319	0.7862	0.1393	0.7599	0.0003	0.9987	0.0738	0.7414	0.1484	0.7304
	80%	20% (fold 4)	0.2480	0.5587	0.0749	0.8610	0.0496	0.9079	0.0618	0.8540	0.1417	0.7227	0.0004	0.9991	0.1102	0.7969	0.1804	0.7518
	80%	20% (fold 5)	0.2800	0.4630	0.0472	0.9100	0.0465	0.9113	0.0501	0.9045	0.0474	0.8097	0.0005	0.9989	0.1206	0.7687	0.1587	0.7873
Verification training - 83.4%, testing - 16.6%	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	0.0002	0.9994	0.0729	0.8214	0.1400	0.8019

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	
Optimized for fold 2 149-190-182		Optimized for fold 2 149-190-182		Optimized for fold 2 70-81-15		Optimized for fold 2 70-81-15		Optimized for fold 3 77-145-132		Optimized for fold 3 77-145-132		Optimized for fold 3 155-199-32		Optimized for fold 3 155-199-32		Optimized for fold 4 125-183-168		Optimized for fold 4 125-183-168	
RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2
0.0530	0.8847	0.0005	0.9988	0.0543	0.8835	0.0448	0.9039	0.0536	0.8834	0.0052	0.9888	0.0598	0.8718	0.0495	0.8938	0.0634	0.8620	0.0021	0.9954
0.0639	0.8535	0.0025	0.9940	0.0981	0.7749	0.0802	0.8160	0.0742	0.8299	0.0040	0.9907	0.1012	0.7679	0.0825	0.8108	0.0830	0.8097	0.0016	0.9962
0.0713	0.7459	0.0007	0.9974	0.0478	0.8325	0.0369	0.8706	0.0615	0.7809	0.0028	0.9898	0.0445	0.8440	0.0347	0.8782	0.0847	0.6981	0.0006	0.9976
0.1019	0.8110	0.0015	0.9971	0.1294	0.7615	0.0846	0.8441	0.0727	0.8652	0.0191	0.9646	0.1622	0.7011	0.0769	0.8582	0.0657	0.8782	0.0006	0.9987
0.0964	0.8165	0.0018	0.9965	0.1604	0.6924	0.1069	0.7949	0.0786	0.8503	0.002	0.9961	0.1274	0.7555	0.0942	0.8193	0.1065	0.7972	0.0083	0.9841
0.0512	0.8732	0.0018	0.9955	0.0719	0.8239	0.0451	0.8894	0.0498	0.8766	0.0212	0.9481	0.0597	0.8536	0.0424	0.8962	0.0844	0.7911	0.0012	0.997
0.1885	0.5919	0.0055	0.9879	0.1828	0.6039	0.1102	0.7610	0.1601	0.6533	0.0027	0.9940	0.1734	0.6241	0.1250	0.7291	0.1588	0.6562	0.0043	0.9906
0.1001	0.7866	0.0016	0.9964	0.1113	0.7638	0.0880	0.8132	0.2037	0.5659	0.0022	0.9951	0.2109	0.5527	0.0771	0.8363	0.2527	0.4615	0.0065	0.9859
0.1833	0.6281	0.0045	0.9908	0.2021	0.5914	0.1141	0.7693	0.1727	0.6495	0.0084	0.9829	0.1337	0.7296	0.1111	0.7753	0.1906	0.6133	0.0087	0.9821
0.1237	0.7122	0.0061	0.9856	0.1603	0.6296	0.0932	0.7846	0.1924	0.5524	0.0022	0.9948	0.1621	0.6254	0.1111	0.7431	0.1104	0.7431	0.0006	0.9983
0.1294	0.7126	0.0012	0.9972	0.1556	0.6563	0.0917	0.7974	0.2150	0.5226	0.0022	0.9951	0.1929	0.5739	0.0801	0.8230	0.1502	0.6664	0.0016	0.9963
0.1060	0.7702	0.0028	0.9938	0.1212	0.7383	0.0883	0.8094	0.1220	0.7355	0.0032	0.9929	0.1398	0.6982	0.0965	0.7917	0.1161	0.7483	0.0022	0.9951
0.0375	0.9183	0.0032	0.9930	0.0831	0.8218	0.06400	0.8629	0.0382	0.9169	0.0017	0.9963	0.0777	0.8334	0.0550	0.8821	0.0414	0.9099	0.0002	0.9995
0.0437	0.8998	0.0031	0.9927	0.1042	0.7609	0.0811	0.8139	0.0485	0.8888	0.0004	0.9988	0.1078	0.7528	0.0729	0.8327	0.0506	0.8839	0.0006	0.9984
0.0373	0.8669	0.0005	0.9981	0.0741	0.7404	0.0480	0.8318	0.0373	0.8670	0.0002	0.9992	0.0733	0.7432	0.0445	0.8441	0.0403	0.8561	0.0012	0.9954
0.0501	0.9071	0.0017	0.9968	0.1131	0.7915	0.0850	0.8432	0.0527	0.9021	0.0012	0.9976	0.1114	0.7946	0.0848	0.8437	0.0435	0.9193	0.0003	0.9993
0.0465	0.9114	0.0007	0.9986	0.1272	0.7560	0.1006	0.8069	0.0459	0.9125	0.0019	0.9962	0.1257	0.7589	0.0866	0.8338	0.0482	0.9081	0.0011	0.9978
0.0378	0.9063	0.0004	0.9989	0.0623	0.8475	0.0477	0.8832	0.0359	0.9111	0.0004	0.9987	0.0721	0.8235	0.0416	0.8979	0.0566	0.8597	0.0000	0.9997

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 4 125-183-168		Optimized for fold 4 72-175-50		Optimized for fold 4 125-183-168		Optimized for fold 5 102-191-129		Optimized for fold 5 102-191-129		Optimized for fold 5 199-162-198		Optimized for fold 5 199-162-198		Optimized for fold 6 123-180-156		Optimized for fold 6 123-180-156		Optimized for fold 6 134-161-63		Optimized for fold 6 134-161-63	
RMSE	R2	RMSE	R2	RMSE	R2	RMSE	R2	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	0.0663	0.8578	0.0414	0.9113	0.9067	0.9067	0.0014	0.9969	0.0832	0.8216	0.0407	0.9126
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	0.0628	0.8561	0.0009	0.9977	0.0997	0.7713	0.0752	0.8275
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	0.0832	0.7036	0.0008	0.9968	0.0460	0.8387	0.0354	0.8759
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	0.0653	0.8788	0.0015	0.9971	0.0945	0.8258	0.0804	0.8517
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	0.0940	0.8210	0.0024	0.9953	0.1247	0.7608	0.0933	0.8210
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	0.1400	0.6969	0.0053	0.9884	0.1824	0.6047	0.1363	0.7046
0.0065	0.9859	0.1944	0.5876	0.0787	0.8329	0.1181	0.7483	0.0033	0.9929	0.1489	0.6840	0.0862	0.8171	0.0899	0.8083	0.0028	0.9939	0.1106	0.7652	0.0782	0.8341
0.0087	0.9821	0.1380	0.7210	0.1136	0.7702	0.1903	0.6139	0.0072	0.9853	0.1417	0.7135	0.1216	0.7541	0.1987	0.5967	0.0060	0.9877	0.1420	0.7129	0.1021	0.7934
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	0.1215	0.7172	0.0036	0.9914	0.1565	0.6384	0.1018	0.7647
0.0016	0.9963	0.1599	0.6467	0.0823	0.818	0.0472	0.9100	0.0021	0.9951	0.1219	0.7662	0.0898	0.8016	0.1845	0.5904	0.0020	0.9955	0.1837	0.5942	0.0817	0.8194
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	0.0400	0.9129	0.0025	0.9944	0.0958	0.7947	0.0557	0.8806
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	0.0473	0.8916	0.0033	0.9923	0.1075	0.7534	0.0788	0.8191
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	0.0423	0.8491	0.0007	0.9973	0.0768	0.7310	0.0385	0.865
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	0.0447	0.9170	0.0006	0.9988	0.1127	0.7923	0.0807	0.8513
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