

An Explainable Artificial Intelligence Approach Based on Deep Type-2 Fuzzy Logic System

Ravikiran Chimatapu



A thesis submitted for the degree of

Doctor of Philosophy in Computer Science

School of Computer Science and Electronic
Engineering

University of Essex

2021

Acknowledgements

I would like to thank the following people for their contributions to this PhD and my career, for which I will always be indebted:

Professor Hani Hagrass, the academic supervisor, I would like to extend my sincere gratitude and thanks for his supervision, exceptional guidance, and support.

Dr Gilbert Owusu, one of the industrial supervisors, I would like to thank for the opportunities given to me with respect to this PhD. I would also like to thank him for his guidance and support.

Dr Andrew Starkey, one of the industrial supervisors, I would like to thank him for his exceptional guidance on evolutionary algorithms and fuzzy logic. I would also like to thank him for his supervision, guidance and support.

I would like to thank British Telecom for supporting this PhD.

Finally, I would like to thank my family and friends, for their love, support and guidance.

Publications Arising from this Work

Journal Papers

- J. M. Mendel, R. Chimatapu, and H. Hagra, "Comparing the performance potentials of singleton and non-singleton type-1 and interval type-2 fuzzy systems in terms of sculpting the state space," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 4, pp. 783-794, 2019.

Conference Papers

- R. Chimatapu, H. Hagra, M. Kern, and G. Owusu, "Hybrid Deep Learning Type-2 Fuzzy Logic Systems For Explainable AI," *proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2020: IEEE*, pp. 1-6.
- R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Enhancing Human Decision Making for Workforce Optimisation Using a Stacked Auto Encoder Based Hybrid Genetic Algorithm," *proceedings of the International Conference on Innovative Techniques and Applications of Artificial Intelligence, 2018: Springer*, pp. 63-75.
- R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Stacked Auto Encoder Based Hybrid Genetic Algorithm for Workforce Optimization," *proceedings of the 10th Computer Science and Electronic Engineering (CEEC), 2018: IEEE*, pp. 236-241.
- R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Interval type-2 fuzzy logic based stacked autoencoder deep neural network for generating explainable ai models in workforce optimization," *Proceedings of the 2018*

IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2018: IEEE, pp. 1-8. (Shortlisted for the Best Student Paper Award in FUZZ-IEEE 2018)

- R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "A Big-Bang Big-Crunch Type-2 Fuzzy Logic System for Generating Interpretable Models in Workforce Optimization," *proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2018: IEEE, pp. 1-8.*

Book Chapters

- R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Explainable AI and Fuzzy Logic Systems," in Theory and Practice of Natural Computing, Lecture Notes in Computer Science book series, Springer, pp.3-20, November 2018.

Patents

R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Explainable Machine Learning Systems for Telecom Optimisation", PCT/EP2020/057529, https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2020193329&_cid=P20-KIP8FK-18085-1

Abstract

Artificial intelligence (AI) systems have benefitted from the easy availability of computing power and the rapid increase in the quantity and quality of data which has led to the widespread adoption of AI techniques across a wide variety of fields. However, the use of complex (or Black box) AI systems such as Deep Neural Networks, support vector machines, etc., could lead to a lack of transparency. This lack of transparency is not specific to deep learning or complex AI algorithms; other interpretable AI algorithms such as kernel machines, logistic regressions, decision trees, or rules-based algorithms can also become difficult to interpret for high dimensional inputs. The lack of transparency or explainability reduces the effectiveness of AI models in regulated applications (such as medical, financial, etc.), where it is essential to explain the model operation and how it arrived at a given prediction.

The need for explainability in AI has led to a new line of research that focuses on developing Explainable AI techniques. There are three main avenues of research that are being explored to achieve explainability; first, Deep Explanations, which involves the modification of existing Deep learning models to add explainability. The methods proposed to do Deep explanations generally provide details about all the input features that affect the output, generally in a visual format as there might be a large number of features. This type of explanation is useful for tasks such as image recognition, but in other tasks, it might be hard to distinguish the most important features. Second, Model induction, which involves methods that are model agnostic, but these methods might not be suitable for use in regulated applications. The third method is to use existing interpretable models such as decision trees, fuzzy logic, etc., but the problem with them is that they can also become opaque for high dimensional data.

Hence, this thesis presents a novel AI system by combining the predictive power of Deep Learning with the interpretability of Interval Type-2 Fuzzy Logic Systems. The advantages of such a system are, first, the ability to be trained via labelled and unlabelled data (i.e., mixing supervised and unsupervised learning). Second, having embedded feature selection abilities (i.e., can be trained by hundreds and thousands of inputs with no need for feature selection) while delivering explainable models with small rules bases composed of short rules to maximize the model's interpretability.

The proposed model was developed with data from British Telecom (BT). It achieved comparable performance to the deep models such as Stacked Autoencoder (SAE) and Convolution Neural Networks (CNN). In categorical datasets, the model outperformed the SAE by 2%, performed within 2-3% of the CNN and outperformed Multi-Layer Perceptron (MLP) and IT2FLS by 4%. In the regression datasets, the model performed slightly worse than the SAE, MLP and CNN models, but it outperformed the IT2FLS with a 15% lower error. The proposed model achieved excellent interpretability in a survey where it was rated within 2% of the highly interpretable IT2FLS. It was also rated 20% and 17% better than Deep learning XAI tools LIME and SHAP, respectively. The proposed model shows a small loss in performance for significantly higher interpretability, making it a suitable replacement for the other AI models in applications with many features where interpretability is paramount.

Table of Contents

Acknowledgements	ii
Publications Arising from this Work	iii
Abstract.....	v
List of Figures.....	xiv
List of Tables	xx
List of Acronyms	xxiv
Chapter 1. Introduction	1
1.1 Aims of the Thesis	4
1.2 Thesis Layout.....	5
Chapter 2. Explainable Artificial Intelligence	7
2.1 Deep Explanations	9
2.1.1 Model Simplification	10
2.1.2 Feature Relevance	12
2.2 Interpretable Models	15
2.2.1 Linear/Logistic Regression	16
2.2.2 Decision Trees	17
2.2.3 Bayesian Models	19
2.2.4 K-Nearest Neighbours	20
2.2.5 Fuzzy Logic	22
2.3 Model Induction.....	24

2.3.1	Local Interpretable Model-Agnostic Explanations (LIME).....	24
2.3.2	Anchor Local Interpretable Model-Agnostic Explanations.....	27
2.3.3	SHAP	29
2.3.4	Partial Dependence Plot.....	32
2.3.5	Individual Conditional Expectation	33
2.3.6	Interpretable Mimic Learning	34
2.4	Hybrid Deep Learning and Fuzzy Logic Systems	36
2.4.1	Fuzzy Restricted Boltzmann Machine	37
2.4.2	Fuzzy Deep Neural Network.....	38
2.4.3	Fuzzy Deep Learning.....	40
2.4.4	Takagi Sugeno Deep Fuzzy Network	41
2.4.5	Fuzzy Deep Belief Network.....	42
2.4.6	Active Fuzzy Deep Belief Network.....	43
2.4.7	Pythagorean Fuzzy Deep Boltzmann Machine	43
2.5	Summary	45
Chapter 3.	An Overview on Fuzzy Logic	47
3.1	Uncertainty.....	48
3.2	Type-1 Fuzzy Logic Systems.....	50
3.2.1	Linguistic Variables	51
3.2.2	Membership Functions.....	52
3.2.3	Fuzzy Set Theoretic Operations.....	54

3.2.4	Rules	56
3.2.5	Fuzzifier	59
3.2.6	Fuzzy Inference Engine	60
3.2.7	Defuzzifier	61
3.3	Type-2 Fuzzy Logic Systems.....	63
3.3.1	General Type-2 Fuzzy Sets	63
3.3.2	z-slices Based General Type-2 Fuzzy Sets	66
3.3.3	Interval Type-2 Fuzzy Sets	68
3.3.4	Interval Type-2 Fuzzy Logic Systems	70
3.3.5	Type-Reduction + Defuzzification	71
3.3.6	Direct Defuzzification.....	76
3.4	Fuzzy Rule-Based Classification Systems.....	78
3.4.1	Scaled Support and Scaled Confidence	78
3.4.2	Calculate Output Class.....	80
3.4.3	Similarity Metric	81
3.5	Design Methods for Fuzzy Logic Systems	84
3.5.1	Wang Mendel Method	84
3.5.2	Enhanced Wang-Mendel Method	91
3.6	Summary	94
Chapter 4.	An Overview of Selected Optimization Algorithms	95
4.1	Big Bang Big Crunch (BB-BC).....	95

4.1.1	Implementation of the Big Bang-Big Crunch Algorithm	99
4.2	Genetic Algorithms	99
4.2.1	Genetic Operators	100
4.2.2	Implementation of Genetic Algorithm	102
4.3	Gradient Descent Optimization Algorithm	103
4.3.1	Adaptive Learning Rate	105
4.3.2	Implementation of Gradient Descent Algorithm	106
4.4	Summary	107
Chapter 5.	Overview of the Datasets used in the Research	108
5.1	Classification Problems	109
5.1.1	BT Customer Service (BTCS)	109
5.1.2	CLL Identification (CLL)	110
5.1.3	IDA 2016 (IDA).....	110
5.1.4	Epileptic Seizure (ES).....	111
5.1.5	PD Speech (PDS).....	112
5.1.6	Santander CTP (SCTP)	112
5.2	Regression Problems.....	113
5.2.1	Wi-fi Localization (WL)	113
5.2.2	Swiss Premium (SP)	114
5.2.3	CT Scan Region (CTSR)	115
5.2.4	BT PWA (BTP).....	116

5.2.5	Song year (SY).....	117
5.3	Summary	118
Chapter 6.	The Proposed Deep Type-2 Fuzzy Logic System.....	119
6.1	Model Representation	122
6.1.1	Representation of the Type-1 models	122
6.1.2	Representation of IT2 Models.....	128
6.2	Layer Wise Training of The D2FLS	133
6.2.1	Fitness Function	133
6.2.2	Hidden Layer Training.....	134
6.2.3	Optimization Method for the Final Layer	140
6.3	Experiments and Results	145
6.3.1	Training Parameters	145
6.3.2	Deep Type 2 Fuzzy Logic system Vs Deep Neural Networks	148
6.3.3	D2FLS vs Shallow Neural Networks and an IT2FLS	153
6.3.4	Comparison between the Three-Step Training process and Single Step Training process	155
6.3.5	Comparison Between D2FLS trained using the various Membership Function Types	157
6.3.6	Comparison of the performance of the BB-BC against Genetic Algorithms	160
6.3.7	Deep Type-2 FLS vs Deep Type-1 FLS	163
6.4	Summary	165

Chapter 7. Deep Type-2 FLS trained using a Stacked Autoencoder	168
7.1 Autoencoder Training	169
7.2 Pretraining D2FLS	171
7.3 Optimization Method for the Final Layer	176
7.4 Experiments and Results	180
7.4.1 D2FLS vs Fuzzy Stacked Autoencoder	180
7.4.2 D2FLS Comparison between The Two Training Methods.....	183
7.4.3 Effectiveness of Pre-training on the D2FLS	187
7.5 Summary	189
Chapter 8. Local Interpretability Enhancement for Deep Type-2 Fuzzy	
Logic Systems 191	
8.1 Rule-Based Explanations	192
8.1.1 Regression Datasets	195
8.1.2 Classification Datasets	196
8.2 Feature Importance Explanations.....	196
8.2.1 Regression Datasets	198
8.2.2 Categorical Datasets.....	199
8.3 Survey	201
8.4 Summary	204
Chapter 9. Enhanced Deep Type-2 Fuzzy Logic System for Global	
Interpretability 205	

9.1 Global Interpretability at Modular Level	208
9.1.1 Modular Rule-Based Explanation.....	208
9.1.2 Modular Feature Importance Explanations.....	211
9.2 Enhanced Deep Type-2 Fuzzy Logic System	214
9.2.1 Hidden layer training	215
9.2.2 Optimization Method for the Final Layer	217
9.2.3 Extracting rules	218
9.3 Experiments and Results	220
9.3.1 Comparison between a D2FLS pre-trained as an FAE vs Enhanced D2FLS pre-trained as an FAE	220
9.3.2 Comparison between a D2FLS pre-trained as FAE and Enhanced D2FLS pre-trained using SAE.....	224
9.4 Summary	227
Chapter 10. Conclusions and Future Work	229
10.1 Conclusions	229
10.2 Future work.....	234
Bibliography	235

List of Figures

Figure 2-1: Existing AI techniques and XAI methods [16]	8
Figure 2-2: CNN explanation via decision trees [23]	11
Figure 2-3: Deep Taylor Decomposition [28]	13
Figure 2-4: Linear Regression with one independent variable [32]	16
Figure 2-5: Decision Tree [32]	18
Figure 2-6: Bayesian network for the car start problem [36].....	19
Figure 2-7: K-Nearest neighbour [37]	21
Figure 2-8: A Typical Interval Type-2 Fuzzy Logic System [1]	22
Figure 2-9: Interval Type-2 Fuzzy Set [1]	23
Figure 2-10: LIME Algorithm: Generated sample data for training local model [32].....	26
Figure 2-11: Anchors generated for keywords “not” and “bad” [41].....	27
Figure 2-12: SHAP explanations [43].....	30
Figure 2-13: ICE plot of survival probability by Age [32].....	34
Figure 2-14: Interpretable Mimic Learning	35
Figure 2-15: Fuzzy Restricted Boltzmann machine [54]	38

Figure 2-16: Fuzzy Deep Neural Network [58].....	39
Figure 2-17: Fuzzy Deep Learning [59]	40
Figure 2-18: Takagi Sugeno Deep Fuzzy Network [60].....	41
Figure 2-19: Fuzzy Deep Belief Network [62]	42
Figure 2-20: Pythagorean Fuzzy Deep Boltzmann Machine [65]	44
Figure 3-1: Type-1 Fuzzy Logic System [38].....	50
Figure 3-2: Membership Functions for Pressure [38].....	52
Figure 3-3: Types of Membership Function a) Triangular b) Trapezoidal c) Gaussian d) Singleton	53
Figure 3-4: Fuzzy set theoretical operations a) Fuzzy sets A and B b) A union B c) A intersect B d) B complement [38].....	56
Figure 3-5: a) Singleton Fuzzification b) Non-singleton Fuzzification [76]	60
Figure 3-6: a) Type-1 Membership Function b) Blurred Type-1 Membership Function c) Footprint of Uncertainty [38]	64
Figure 3-7: (a) Side view of a general type-2 fuzzy set, with three zLevels on the third dimension (b) Tilted rear/below view of the same set, indicating the position of the three zSlices. (c) Side view of the same zSlices version in (a), with $I=3$. (d) Tilted rear/below view of the same set, showing the zSlices [80].	65

Figure 3-8: (a) Front view of a general type-2 set \tilde{F} . (b) Third dimension at x' of a zSlices-based type-2 fuzzy set with $I=4$ [80]	66
Figure 3-9: Type -2 Fuzzy Logic System	70
Figure 3-10: An Example to Illustrate Similarity Metric.....	82
Figure 3-11: Division of Domain Intervals [93]	85
Figure 3-12: The form of a Fuzzy Rule Base [93].....	89
Figure 4-1: 2D depiction of the initial population of candidate solutions in BB-BC algorithm.....	96
Figure 4-2: 2D depiction of the candidate solutions in the second Big Bang Phase in BB-BC algorithm.....	97
Figure 4-3: Flow Chart for the Big Bang Big Crunch Algorithm.....	98
Figure 4-4: Pseudocode for the BB-BC algorithm	99
Figure 4-5: Single Point Crossover [69]	101
Figure 4-6: Multi-Point Crossover [69]	102
Figure 4-7: Flow Chart of a Genetic Algorithm	103
Figure 4-8: Genetic Algorithm Pseudo Code.....	103
Figure 4-9: Gradient Descent Pseudo Code [101]	106
Figure 6-1: A Deep Type-2 Fuzzy Logic System Architecture	120

Figure 6-2: Layer Wise Training D2FLS.....	121
Figure 6-3: Fuzzy Autoencoder Architecture	122
Figure 6-4: Representation of a Trapezoidal Type-1 Membership Function	124
Figure 6-5: Representation of a Triangular Type-1 Membership Function.....	125
Figure 6-6: Representation of a Gaussian Type-1 Membership Function	126
Figure 6-7: Representation of a Footprint of uncertainty (FOU) for Trapezoidal MFs.....	129
Figure 6-8: Representation of a Footprint of uncertainty (FOU) for Triangular MFs	130
Figure 6-9: Representation of the FOU of a Gaussian IT2 MF	131
Figure 6-10: Training Algorithm for IT2FLS	134
Figure 6-11: Fuzzy Set Generated by D2FLS Training for MSLCL1 2 feature of the BT PWA dataset	152
Figure 6-12: Fuzzy Set Generated by D2FLS Training for Contractor 2 feature of the BT PWA dataset	152
Figure 6-13: Examples of IT2 Fuzzy Sets Generated for D2FLS during Training where (a) (d) are Trapezoidal MF, (b) (e) are Triangular MF, and (c) (f) are Gaussian MF for Contractor 0 and MSLCL1 2 features respectively of the BT PWA dataset	159
Figure 7-1: Fuzzy Stacked Autoencoder [116].....	168

Figure 7-2: Stacked Autoencoder Training.....	169
Figure 7-3: Fuzzy Sets Generated by D2FLS SAE Training for (a) Loans 1 feature and (b) MSLCL2 2 feature of the BT PWA dataset	185
Figure 8-1: Rule-Based Explanation depicting the Rule contributions to the outputs of the D2FLS layers.....	193
Figure 8-2: Rule Contributions for the Intermediate Variable H00.....	193
Figure 8-3: Feature Importance Scores.....	197
Figure 8-4: SHAP explanation for Sparse Stacked Autoencoder	201
Figure 8-5: LIME explanation for Sparse Stacked Autoencoder.....	202
Figure 8-6: Survey Plot.....	202
Figure 9-1: Rule-Based Explanation.....	208
Figure 9-2: Rule-Based Explanation for one hidden output	209
Figure 9-3: Rule Base Explanation for ILU HH Actual is High module.....	209
Figure 9-4: Modular Simple Explanation for Positive Outputs	212
Figure 9-5: Modular Simple Explanation for Negative Outputs	213
Figure 9-6: Enhanced Deep Type-2 Fuzzy Logic System. The Dotted Rectangles indicate the two FLSs that are constrained to use the same Linguistic Labels for the Consequents as the Antecedents of subsequent FLS.....	214

Figure 9-7: Fuzzy Sets Generated by Enhanced D2FLS FAE Training for (a)
Loans 0 feature and (b) Rank 0 feature of the BT PWA dataset 223

List of Tables

Table 5-1: Summary of Datasets used in the Experiments	108
Table 6-1: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Categorical Datasets with Average Recall as Fitness function	147
Table 6-2: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Regression Datasets using Mean Absolute Error as the Fitness Function	148
Table 6-3: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Regression Datasets using Root Mean Square Error as the Fitness Function	149
Table 6-4: Snapshot of Rule base of the Hidden Layer of the D2FLS on the BT PWA Dataset	150
Table 6-5: Snapshot of Rule base of the Output Layer of the D2FLS on the BT PWA Dataset	151
Table 6-6: Comparison of the performance of the D2FLS vs Multi-layer perceptron vs IT2FLS in Classification Datasets using Average Recall as the Fitness Function	154
Table 6-7: Comparison of the performance of the D2FLS vs Multi-layer perceptron vs IT2FLS in Regression Datasets using MAE as the Fitness Function	155

Table 6-8: Comparison for the Three-Step Training Process and the Single Step Training Process for training the D2FLS with Mean Absolute Error as the Fitness Function	155
Table 6-9: Comparison of performance of D2FLS for difference types of MFs on Categorical Datasets with Average Recall as the Fitness Function.....	157
Table 6-10: Comparison of performance of D2FLS for difference types of MFs on Regression Datasets with Mean Absolute Error as the Fitness Function	158
Table 6-11: Comparison Between D2FLS Trained using BB-BC and Genetic Algorithm on Categorical Datasets with Average Recall as the Fitness Function ...	160
Table 6-12: Comparison Between D2FLS Trained Using BB-BC and Genetic Algorithm on Regression Datasets with Mean Absolute Error as the Fitness Function	161
Table 6-13: Impact of number of Generations on D2FLS trained using Genetic Algorithm.....	162
Table 6-14: D2FLS optimized by Genetic Algorithm using a single step training with MAE as fitness function	163
Table 6-15: Deep Type-2 FLS vs Deep Type-1 FLS on Classification Datasets	164
Table 6-16: Deep Type-2 FLS vs Deep Type-1 FLS on Regression Datasets	165
Table 7-1: Comparison between D2FLS and FSAE on Categorical Datasets with Average Recall as Fitness Functions	181

Table 7-2: Comparison between D2FLS and FSAE on Regression Datasets with Mean Absolute Error as the fitness function	181
Table 7-3 : D2FLS Pretrained using FAE vs D2FLS Pretrained using SAE on Classification Datasets with Average Recall as the fitness function.....	182
Table 7-4: D2FLS Pretrained using FAE vs D2FLS Pretrained using SAE on Regression datasets with MAE as the fitness function.....	183
Table 7-5: Snapshot of Rule base of the Hidden Layer of the D2FLS trained using SAE on the BT PWA Dataset.....	184
Table 7-6: Snapshot of Rule base of the Output Layer of the D2FLS pre-trained using SAE on the BT PWA Dataset	186
Table 7-7: Comparison of D2FLS with hidden layers pre trained using FAE against D2FLS without pretraining on Classification Datasets with fitness function Average Recall.....	187
Table 7-8: Comparison of D2FLS with hidden layers pre trained using FAE against D2FLS without pretraining on Regression datasets with MAE as the fitness function.....	188
Table 8-1: Survey Results.....	203
Table 9-1: Comparison of the D2FLS with the Enhanced D2FLS on Categorical Datasets with Average Recall as the fitness function.....	220

Table 9-2: Comparison of the D2FLS with Enhanced D2FLS on the Regression dataset with Mean average error as the fitness function.....	221
Table 9-3: Rules of the Hidden Layer of a D2FLS with five outputs.....	222
Table 9-4: Rules of the Final Layer of a D2FLS	224
Table 9-5: Comparison of the D2FLS pre-trained using FAE with the Enhanced D2FLS pre-trained using SAE on Categorical Datasets with Average Recall as the fitness function.....	225
Table 9-6: Comparison of the D2FLS FAE with Enhanced D2FLS SAE on the Regression dataset with Mean average error as the fitness function	226

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BB-BC	Big Bang Big Crunch
BTCS	BT Customer Service
BTP	BT Preferred Work Area
CART	Classification and Regression Trees
CLL	Chronic Lymphocytic Leukemia
CNN	Convolutional Neural Network
CRED	Cnn-Rnn Earthquake Detector
CTSR	Computerized Tomography Scan
DARPA	Defense Advanced Research Projects Agency
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
Deep LIFT	Deep Learning Important FeaTures
DNN	Deep Neural Network
EEG	electroencephalogram
ES	Epileptic Seizure
FAE	Fuzzy Autoencoder
FDBN	Fuzzy Deep Belief Network
FDL	Fuzzy Deep Learning
FDNN	Fuzzy Deep Neural Network
FL	Fuzzy Logic
FLC	Fuzzy Logic Controller
FOU	Footprint of Uncertainty
FRBCS	Fuzzy Rule Based Categorical System
FRBM	Fuzzy Restricted Boltzmann Machine
FS	Fuzzy System
FSAE	Fuzzy Stacked Autoencoder

GA	Genetic Algorithm
GBT	Gradient Boosted Trees
ICE	Individual Conditional Expectation
IDA	Intelligent Data Analysis
IEEE	Institute of Electrical and Electronics Engineers
IT2FLS	Interval Type-2 Fuzzy Logic System
KM	Karnik Mendel
KNN	K-Nearest Neighbour
LIME	Local Interpretable Model-Agnostic Explanations
LRP	Layer wise Relevance Propagation
MAE	Mean Absolute Error
MF	Membership Function
ML	Machine Learning
MLP	Multi Layer Perceptron
PDP	Partial Dependence Plot
PDS	Parkinson's Disease Speech
PFDBM	Pythagorean Fuzzy Deep Boltzmann Machine
RBM	Restricted Boltzmann Machine
RMSE	Root Mean Squared Error
SAE	Stacked Autoencoder
SCTP	Santander Customer Prediction
SHAP	SHapley Additive exPlanations
SP	Swiss Premium Prediction
SVM	Support Vector Machine
SY	Song Year
TSDFN	Takagi Sugeno Deep Fuzzy Network
TSFLS	Takagi Sugeno Fuzzy Logic System
UCI	University of California Irvine
VGG	Visual Geometry Group
WA	Work Area

WL	WiFi Localization
WM	Wang Mendel Method
XAI	Explainable Artificial Intelligence

Chapter 1. Introduction

Artificial Intelligence (AI) is the programmed ability of machines to mimic cognitive functions such as learning, problem-solving, etc., that usually require human-level intelligence [1]. AI comprised of all the Machine Learning (ML) techniques such as search and optimization, symbolic and logical reasoning, statistical learning methods and behaviour-based approaches. The recent explosion in computing power, coupled with the rapid rise in the quantity and quality of data available for research have led to the rapid adoption of AI techniques across a wide variety of fields. There are huge incentives for adopting AI, such as product/process cost reduction, consistency, repeatability, improved efficiency, enhanced decision making, as well as helping in the development of new products and services [1]. AI is a significant disruptor that is transforming many industries; it is also helping in spawning new technologies and trends that can transform our lives, such as self-driving cars, AI assistants, etc.

As AI technology matures, it can propel economic growth, transforming the way in which we work with computers [1]. Hence, regulators and participants hope that AI will be inclusive and beneficial to everyone. However, the use of complex AI algorithms such as deep learning, random forests, support vector machines (SVMs), etc., could result in a lack of transparency thus creating "black box" models [1, 2]. This lack of transparency is not specific to deep learning, or complex AI algorithms, other interpretable AI algorithms such as kernel machines, logistic regressions, decision trees or rules-based algorithms can also become very difficult to interpret for high dimensional inputs [3]. The difficulty in interpreting these AI models can be a huge barrier in the adoption of AI systems in regulated applications such as financial, medical, justice etc., where the reliability of the model must be guaranteed.

There is a growing consensus that there is a need to develop technologies that mitigate this problem[4]. The UK Parliament house of lords AI select committee, for example, in their report they mention that "We believe it is not acceptable to deploy any artificial intelligence system which could have a substantial impact on an individual's life unless it can generate a full and satisfactory explanation for the decisions it will take" [5]. The European Parliament has gone further by putting in a clause in the General Data Protection Regulation to address the lack of transparency in AI systems. The clause emphasizes the right of all individuals to obtain "meaningful explanations of the logic involved when automated decision making takes place" [6]. The National Institute of Standards and Technology, U.S. Department of Commerce, has emphasised the need for explainable AI while putting forth four principles that may be used to establish a framework to guide real-world applications of AI [7].

The Financial Stability Board, which is an international agency that monitors global financial systems, has warned that the use of opaque models (such as Deep Learning techniques) can lead to the lack of interpretability or 'auditability' which can contribute to macro-level risks [8]. The financial stability board stressed that further progress in the interpretation of algorithms outputs and decision is an essential condition not only for risk management but also for engendering greater trust from the general public as well as regulators [8].

The IEEE Global Initiative for Ethical Considerations in Artificial Intelligence (AI) and Autonomous Systems (AS) Drives, a program of the IEEE initiated to address ethical issues raised by using AI system, document on Ethically Aligned Design mentions that "A key concern over autonomous systems is that their operation must be transparent to a wide range of stakeholders, for different reasons. For users,

transparency is important because it builds trust in the system by providing a simple way for the user to understand what the system is doing and why. For validation and certification of an autonomous system, transparency is important because it exposes the system's processes for scrutiny"[9].

The European Banking Authority, in its report on Big data and advanced analytics, mentions that "Lack of explainability could represent an important risk in the case of AI/ML models developed by external third parties and then sold as opaque black box packages. The institution acquiring the package needs to have enough means, including explanations, to validate the results produced by the package without being strongly dependent on the external provider" [10].

The lack of transparency in AI has led to a new line of research which focuses on developing AI techniques that are explainable [11]. Although a large body of research has been conducted there are still many challenges in achieving explainable AI techniques, such as.

- The need to develop explainable models that would adapt according to the user profile (level of expertise, domain knowledge, cultural background, interests and preferences and other contextual variables) and the explanation request setting (justification, teaching, audit, etc.) [12] [13].
- The trade-off between performance and interpretability that is, interpretability reduces as the performance of the system increases [14].
- The need to develop explainable models that can deal with inputs with a large number of features as many, otherwise, explainable model falter in these situations [3] [15].

To overcome the first challenge, semantic representation of the predictions that use natural language seems to be the way forward [16]. Furthermore, to overcome the trade-off between performance and interpretability, a hybrid modelling approach that combines high-performance models with interpretable models has been suggested [14]. Embedding feature selection within the model, such as in Stacked autoencoders, which can be considered as being one way to overcome the third problem.

From the discussion above a possible approach to solve these challenges is to combine Deep Learning training techniques with Fuzzy Logic Systems (FLS). Where the Deep Learning will help improve the performance of the AI model and add embedded feature selection, the FLS can be used to model the uncertainties in the data and express the logic behind the AI model as human-understandable IF-Then rules with linguistic labels. Hence, this thesis will explore a possible method of achieving explainable AI using a combination of Deep Learning and Type-2 Fuzzy Logic Systems.

1.1 Aims of the Thesis

The thesis aims to investigate and implement an explainable AI technique to solve real-world problems. The core aim is to combine the predictive power and embedded feature selection capability of Deep Learning techniques with the interpretability of Fuzzy Logic Systems. The remaining aims of this thesis are as follows

- To investigate the suitability of the model to solve real-world problems.
- To investigate the most suitable training method for the new explainable AI technique

- To investigate the most suitable optimization algorithm for the new explainable AI technique
- To investigate and develop local explanations that are understandable to all audience that might use these explanations
- To investigate and develop global explanations that can provide a holistic understanding of the new Explainable AI Technique.

1.2 Thesis Layout

This thesis is structured as follows; Chapter 2 will give an overview of Explainable AI and the three main methods, Deep Explanations, Interpretable AI and Model Induction, that are being explored to achieve Explainable AI. It will present the case for combining Deep Learning and Fuzzy Logic. It will also explore the techniques that have been used to combine these two AI techniques.

Chapter 3 will give an overview of the Fuzzy logic system and the two major subtypes type-1 fuzzy logic and type-2 fuzzy logic and the various operators and components of these systems.

Chapter 4 will give an overview of optimization algorithms. It describes some of the algorithms such as Big bang big crunch, Genetic Algorithms and gradient descent algorithms.

Chapter 5 will give a list of problems or datasets used to evaluate the AI models explored in this thesis. The problems were chosen with an emphasis on choosing real-world datasets with a large number of features. The problems are divided into Classification and Regression datasets based on the type of outputs or targets for these problems.

Chapter 6 presents the proposed Deep Type-2 Fuzzy Logic system detailing the training method used to train this AI model. It will also evaluate this AI model against two deep learning techniques. We will also evaluate some of the optimization algorithms and training techniques that could be used to train the AI model.

Chapter 7 presents an alternative training method to train the Deep Type-2 Fuzzy Logic System using Stacked autoencoders. It will also present Fuzzy Stacked Autoencoders and evaluate them against the model presented in the previous chapter

Chapter 8 presents two methods for extracting locally interpretable explanations from the Deep Type-2 Fuzzy Logic system. It also evaluates these explanations by conducting a survey where these explanations are compared against the explanations provided by LIME and SHAP and an Interval Type-2 Fuzzy Logic System.

Chapter 9 expands the two methods presented in the previous chapter to provide global explanations at the modular level. It will also present an enhancement to the Deep type-2 fuzzy logic system to extract global rules from it.

Chapter 10 presents the conclusion of the thesis and discusses potential future work.

Chapter 2. Explainable Artificial Intelligence

An Explainable Artificial Intelligence (XAI) or transparent AI or interpretable AI is one which produces explanations or reason for its actions which can be clearly understood by its users. Explainability will allow AI systems to provide increased transparency and fairness by providing an auditable record of all factors related to a given prediction [6]. Explainability will also ensure that algorithmic decision making is fair and ethical while enabling businesses to meet compliance requirements [1].

XAI is at the intersection of several fields of active research with an emphasis on the following.

- **Transparency:** AI models are used to support decision making [14]; hence, it is vital to ensure that all parties involved in the decision making and the people affected by these decisions can understand them.
- **Fairness:** Explainability allows us to provide a clear understanding of the relations which affect the result, allowing for a fair analysis of the model by highlighting any bias in the data. Hence, XAI allows us to avoid unfair or unethical use of AI models.
- **Interactivity:** The model should be interactive, i.e., it allows the end-users the ability to tweak and interact with the model to ensure success [14].
- **Confidence:** Explanations provided by an AI system will allow us to assess the reliability of the AI system, thereby gaining confidence and trust in the system.
- **Causality:** Since an AI model learns from data, it can discover correlations among this data. Although this might not be enough to unveil the cause-and-

effect relationships, it could provide a first intuition of the causal relationships between the inputs and the outputs.

The transparency in AI rarely comes for free; there are often trade-offs between accuracy and transparency, and these trade-offs are likely to grow as AI systems become even more complex. Hence, the goal of XAI systems should be to create explainable models that provide interpretability while maintaining high accuracy [1, 14].

The explanations provided by these systems should not be restricted to AI experts; they should provide explanations that can be easily understood by the lay user which will allow domain experts to test and augment the AI systems with their expert knowledge. Empowering them to determine when to trust or distrust a given AI model [1, 12] [13].

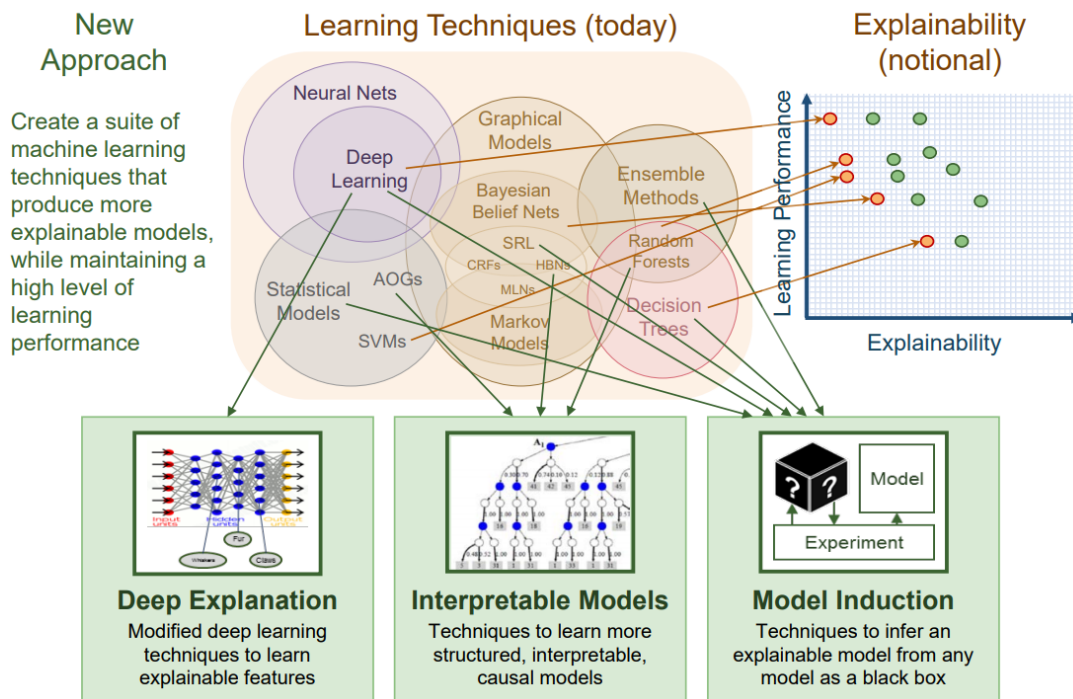


Figure 2-1: Existing AI techniques and XAI methods [4]

As we can see from Figure 2-1, complex AI models such as Deep Neural networks have historically performed better than simple interpretable models such as decision trees or rule-based systems [14]. XAI is a DARPA program that is expected to enable “third-wave AI systems” [4]. The goal of the XAI program is to enable a new suite of techniques that are more explainable while retaining the high level of performance of the AI models [4]. According to a DARPA report [4], the XAI explains individual decisions, enables understanding of overall strengths and weaknesses, and conveys an understanding of how the system will behave in the future and how to correct the system’s mistakes. From Figure 2-1, there are three main methods suggested in the report. In the following sections, we provide an introduction to the three methods suggested to achieve XAI as well as some of the representative works that use these methods. We will also present a fourth method to achieve XAI, in which we propose to combine high-performance complex AI systems such as Deep learning with interpretable systems such as Fuzzy Logic.

2.1 Deep Explanations

Deep Explanations involve the modification of existing Deep learning models to add explainability. The goal here is to increase the explainability of the Deep Learning models without losing their accuracy. Several approaches have been proposed to achieve deep explanations [14]; these can be classified into two broad categories which are presented below.

2.1.1 Model Simplification

2.1.1.1 Rule Extraction

The most common method used to achieve model simplification is by rule extraction; one work that uses this method is Deep Red [17] where the behaviour of the neural network is modelled as decision rules. This method is based on CRED [18], which uses the C4.5 algorithm to induce rules in a shallow neural network with one hidden layer. The authors of Deep Red extend the approach presented in CRED to Deep neural networks by using the C4.5 algorithm to extract rules for each hidden layer based on the preceding layer. Once the rules are extracted for all the layers, they are merged to obtain the rule set.

Some of the other methods that use rule extraction include RxREN [19], where, in the first step, insignificant neurons are removed from the network and rules are extracted using the remaining neurons, In [20] the neural network is trained using a Genetic Algorithm[21]. The significant neurons are then identified, and the rules are extracted using these neurons, [22], where rules are extracted using a sampling and query-based approach.

One point to note here is that the rules can become complex as the number of input attributes and the number of layers increases. Pruning is used to simplify these rules, but if the rules are aggressively pruned the model behaviour, and the rules diverge, i.e., the rules can not accurately predict the model behaviour.

2.1.1.2 Interpreting Neural Networks via Decision Trees

Another method that is used to achieve model simplification is to extract Decision Trees from neural networks. In [23], for example, the authors propose a method to

construct a decision tree (depicted in Figure 2-2) to explain the CNN predictions semantically and quantitatively. The rationale behind the CNN predictions is summarised into decision modes (tree nodes on the decision tree). Each decision mode represents predictions that trigger the same filters and have the same contribution to the output [23].

The decision tree represents all possible decision modes of the CNN in a coarse to fine manner. Nodes near the tree root represent the most common decision modes or characteristics that are shared by many inputs (Most generic rationales as shown in Figure 2-2). Nodes near leaves represent decision modes that are shared by fewer inputs (Most specific rationales as shown in Figure 2-2) [23].

To do this, CNN is trained using filter loss functions that push the filter to represent

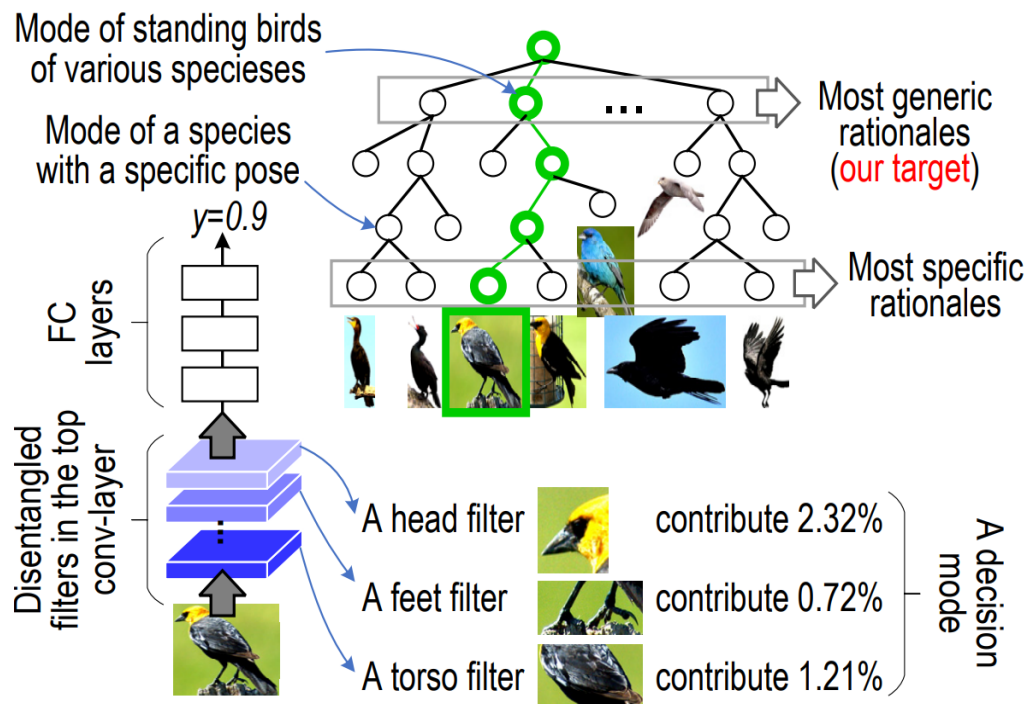


Figure 2-2: CNN explanation via decision trees [23]

object parts of images. Then, each filter is assigned a specific part name. Finally, the

decision modes are mined to explain how CNN uses the parts/filters for predictions, and a decision tree is constructed [23].

- Some of the other methods that use decision trees to interpret neural networks are TREPAN [24] which uses queries to induce decision trees that approximate the neural network,
- Tree Regularization [25] is a regularisation technique that favours models whose decision boundaries can be approximated by decision trees.
- Soft Decision Trees [26] is a technique where the inputs and output of a neural network are used to train a decision tree.

One point to note here is that the decision tree only provides an approximate explanation for the neural network prediction. Furthermore, as the number of inputs increases the number of nodes in the decision tree increases rapidly, which might lead to decision trees that are difficult to interpret.

2.1.2 Feature Relevance

Feature Relevance involves calculating the importance of the input features to the final output. There are a variety of methods that have been proposed for calculating feature relevance. Some of the most popular methods are presented below.

2.1.2.1 Deep Taylor Decomposition

Deep Taylor Decomposition [27] produces a decomposition of the neural network output on the input features. The idea here is to calculate how much each input contributes to the output. This is done by redistributing the predicted output into the input features by performing a backward pass on the network using a predefined set of rules [27, 28]. i.e., by redistributing the output to the neuron in the previous layer, then

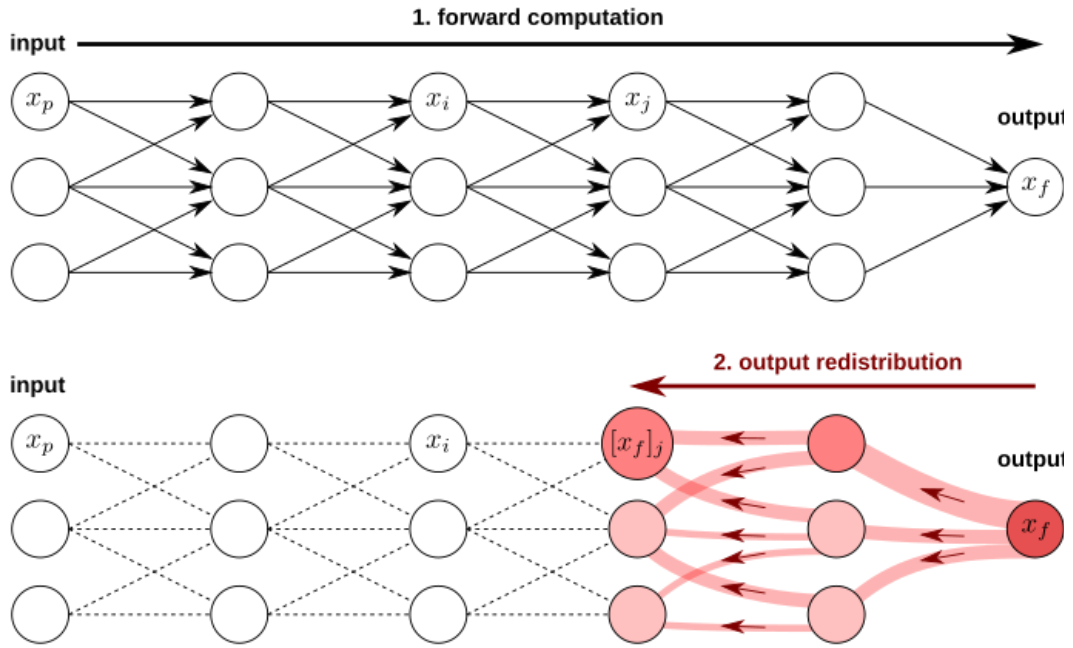


Figure 2-3: Deep Taylor Decomposition [28]

redistributing this value to its input neurons in the previous layer and so on, (depicted in Figure 2-3).

In Figure 2-3, the output that can be redistributed from arbitrary neuron x_j to the collection of neurons $(x_i)_i$ that are inputs to this neuron can be defined as a first-order Taylor expansion. Hence, the propagation rule for the Deep Taylor expansion can be expressed as follows.

$$[x_f]_i = \sum_j \frac{x_i w_{ij}^+}{\sum_{i'} x_{i'} w_{i'j}^+} [x_f]_j \quad (2.1)$$

Where w_{ij}^+ are the weights between the neurons i in layer l and neuron j in layer $l+1$, the + sign indicates that all the weights are forced to be positive.

One point to note here is that this method shows all the features and their contribution. In cases where there are many input features, it can become tough to

distinguish the most important features. The second point to note here is that methods such as Deep Taylor Decomposition which rely on gradients, which only measure the susceptibility of the output to changes in the input and might not necessarily coincide with those areas on which the network bases its decision [29]. The last point here is that the algorithm can be challenging to understand to a lay user and they would need the help of an expert to understand it.

2.1.2.2 Layer wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) [30] uses the idea of relevance redistribution. It starts with a relevance score at the output and backpropagates it through the network by calculating the relevance score for each subsequent layer of the network using the relevance score and parameters from the previous layer. The formula for calculating the relevance score is given below

$$R_i = \sum_j \frac{x_i w_{ij}}{\sum_i x_i w_{ij} + \epsilon} R_j \quad (2.2)$$

Where, x_i is the activation value of the neuron in layer l , w_{ij} weight of the connection between neuron i in layers l and neuron j in layer $l+1$, R_j is the relevance score for each neuron in layer $l+1$ and R_i is the relevance score for each neuron in layer l .

Like the Deep Taylor decomposition, LRP shows all the relevant input features and their relevance scores. This means that it can become difficult to distinguish the most important feature when there are many relevant input features [15].

2.1.2.3 Deep LIFT

Another example of feature relevance is Deep LIFT [31], where the activation of each neuron is compared to a reference activation, and the difference between these two values is used to calculate a score. This score is used to calculate the feature importance by backpropagating these values from the output to the input features. The reference values for all hidden units are calculating during a forward pass of the neural network, using a baseline input. The relevance values for each of the neuron layers is calculated using the following formula.

$$R_i = \sum_j \frac{x_i w_{ij} - \hat{x}_i w_{ij}}{\sum_i x_i w_{ij} + \sum_i \hat{x}_i w_{ij}} R_j \quad (2.3)$$

Where x_i is the activation value of the neuron in layer l , \hat{x}_i is the reference activation value of the neuron in layer l , w_{ij} weight of the connection between neuron i in layers l and neuron j in layer $l+1$, R_j is the relevance score for each neuron in layer $l+1$ and R_i is the relevance score for each neuron in layer l .

One point to note here is that this method shows all the features and their contribution. In cases where there are many input features, it can become extremely hard to distinguish the most important features. Another point here is that the algorithm can be difficult to understand to a lay user and they would need the help of an expert to understand it.

2.2 Interpretable Models

The second approach to achieving XAI is to use models that are inherently interpretable or explainable. Some of these models are listed below.

2.2.1 Linear/Logistic Regression

Linear Regression is used to predict the target output as a weighted sum of the feature inputs. Logistic Regression is an extension of linear regression for classification model used to predict a dependent variable which is binary in nature [32].

Linear Regression can be used to model the relationship between a target y and the input features $X = (x_1, x_2, \dots, x_p)$. The learned relationship between them can be written as follows [32] [33].

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \varepsilon \quad (2.4)$$

Where the predicted output is the weighted sum of its p input features, the coefficients represent the learned feature weights. An example of linear regression with a single independent variable ($p=1$) is depicted in Figure 2-4. For a single independent variable equation (2.4) can be simplified to the equation of a line.

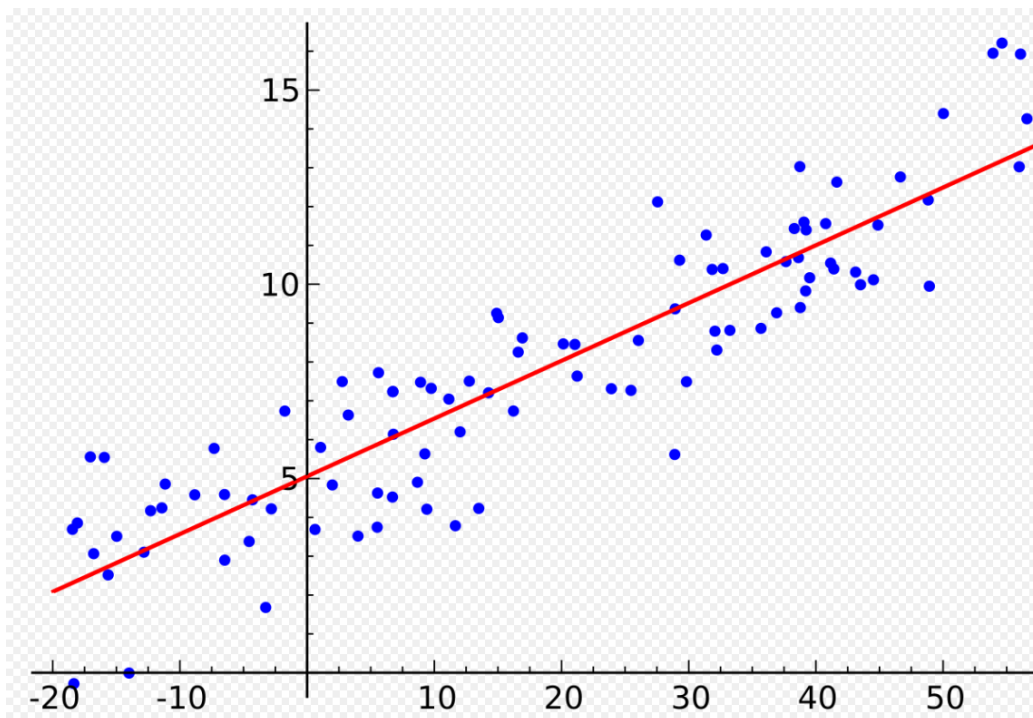


Figure 2-4: Linear Regression with one independent variable [32]

For logistic regression/classification, the right side of the equation is wrapped into the logistic function. This forces the output to assume only values between 0 and 1 [32] [33].

$$P(y^{(i)} = 1) = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}))} \quad (2.5)$$

The model assumes that the input and the output variable have a linear relationship. This makes it easy to create explanations for these models as long as the size of the inputs is limited. However, as the number of input features increases, it becomes difficult to understand the explanations. Hence, linear/logistic regression models require feature selection for inputs with a large number of features. Another problem with these models is that they can only be used to represent linear relationships. Moreover, any nonlinear input or interaction must be handcrafted and provided to the model.

2.2.2 Decision Trees

Decision trees are hierarchical structures where the data is split multiple times based on certain cut off values for the input features. These split-off data are formed into subsets which form the nodes of the decision tree. The final subsets are called terminal, or leaf nodes and the intermediate subsets are called intermediate nodes (depicted in Figure 2-5). The average outcome of these nodes is used to predict the outcome of the leaf nodes.

There are various algorithms that can be used to train a decision tree. The algorithm that is most commonly used to train decision trees is the classification and regression trees (CART) algorithm [34]. In CART, the following formula describes the relationship between the output y and the input features x [32].

$$\hat{y} = \hat{f}(x) = \sum_{i=1}^n c_i I\{x \in R_i\} \quad (2.6)$$

Where each instance falls into exactly one leaf node (=subset R_i). $I\{x \in R_i\}$ is the identity function that returns one if x is in the subset R_i otherwise it returns 0. If an instance falls into a leaf node R_i , the predicted outcome is $\hat{y} = c_i$, where c_i is the average of all training instances in leaf node R_i [32].

Decision trees have been used in a variety of context due to their interpretability, even experts from fields other than computation and AI are comfortable interpreting their outputs [35]. However, their poor generalization ability in comparison to other AI models makes them difficult to recommend in scenarios where predictive performance is a design driver. Tree ensembles can overcome this problem, but they have poor

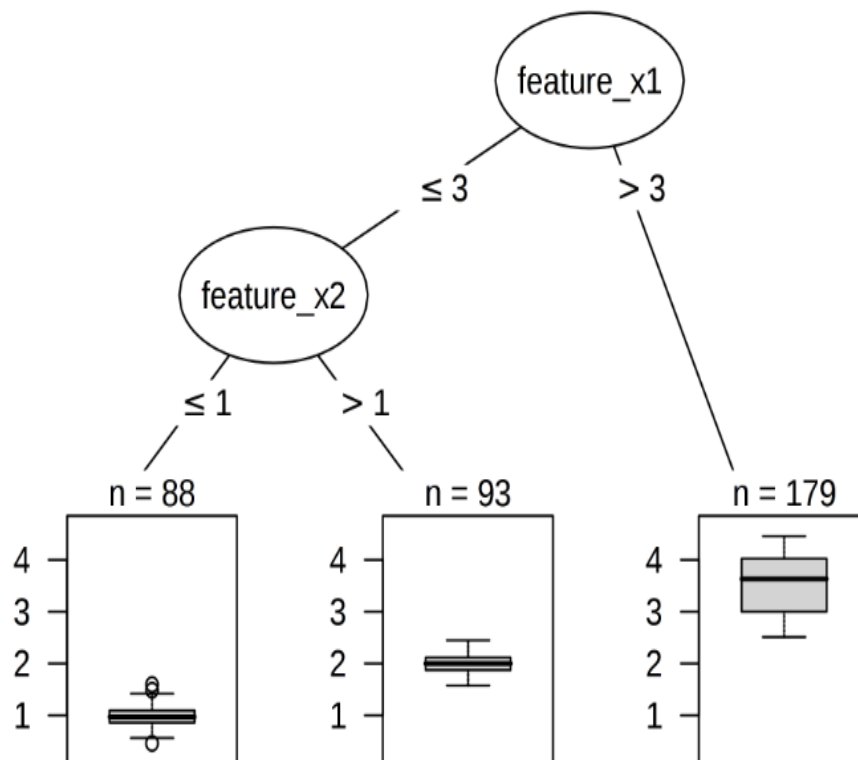


Figure 2-5: Decision Tree [32]

interpretability which means post hoc models have to be used. Another problem with decision trees is that as the number of input features increases the number of nodes in the tree increases, which means that decision trees can become very hard to interpret for inputs with a large number of features[3]. Hence, decision trees require feature selection for inputs with a large number of features.

2.2.3 Bayesian Models

A Bayesian model represents the probabilistic relationship between a set of variables. This means that Bayesian models can convey a clear representation of the relationship between the input features and the target outputs [14]. They usually take the form of a probabilistic acyclic graphical model whose edges represent the probabilistic relationship between a set of variables [14]. Formally, if an edge (A,B) exists in the graph connecting random variables A and B, it means that $P(B|A)$ is a

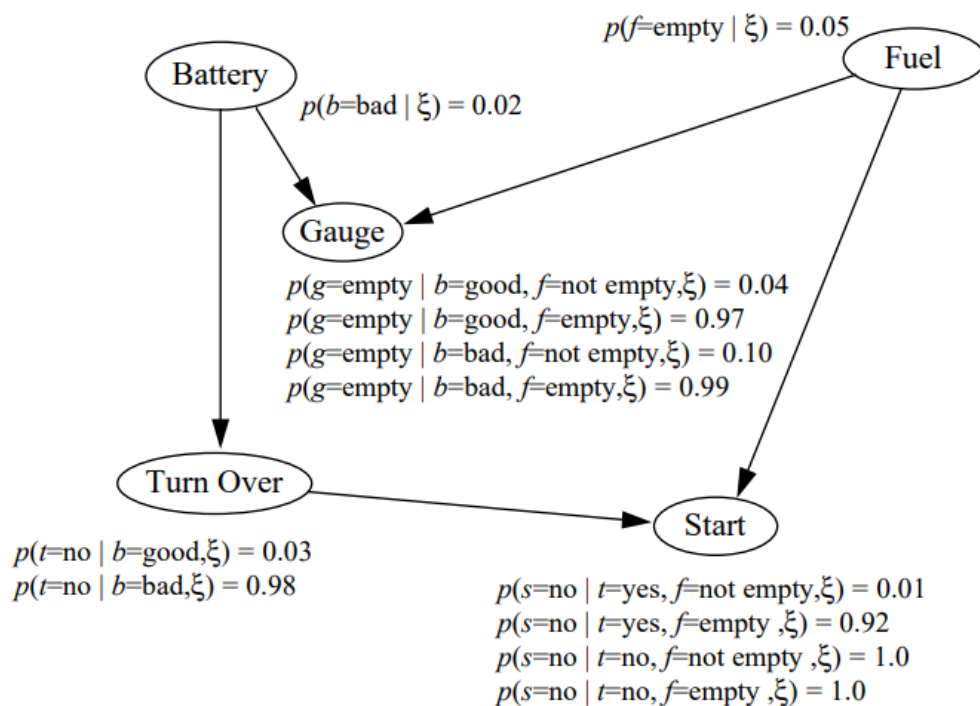


Figure 2-6: Bayesian network for the car start problem [36]

factor in the joint probability distribution. Bayesian networks also satisfy the local Markov property, which means that a variable is conditionally independent of its non-descendent variables. Hence, the Joint probability distribution can be written as follows [36].

$$p(x_1, \dots, x_n | \xi) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}, \xi) = p(x_i | \Pi_i, \xi) \quad (2.7)$$

Where for every variable x_i there will be a subset $\Pi_i \subseteq \{x_1, \dots, x_n\}$ such that x_i and x_1, \dots, x_n are conditionally independent.

For example, Figure 2-6 represents a Bayesian network for the car start problem. Where $P(\text{Turn Over} | \text{Battery}, \text{Gauge}) = P(\text{Turn Over} | \text{Battery})$ since Turnover is conditionally independent of Gauge, given Battery. $P(\text{Start} | \text{Fuel}, \text{Turn Over}, \text{Battery}, \text{Gauge}) = P(\text{Start} | \text{Fuel}, \text{Turn Over})$ since Start is conditionally independent of Battery and Gauge given Fuel and Turn over, etc.

This means that Bayesian Networks are easy to understand when the number of variables is small, but as the number of variables increases, the network becomes more and more complex. This means that explanations extracted from very large Bayesian networks can be hard to understand. Hence, Bayesian networks require feature selection for inputs with a large number of features.

2.2.4 K-Nearest Neighbours

K-Nearest Neighbours (KNN) deals with classification problems in a methodologically simple way: it predicts the class of a test sample by voting the classes of its K nearest neighbours (where the neighbourhood relation is deduced by a measure of the distance between samples). When used in the context of regression problems,

the voting is replaced by an aggregation (e.g. average) of the target values associated with the nearest neighbours [14]. The assumption here is that similar objects exist in close proximity to each other. A commonly used metric for regression problems is Euclidean distance. For classification problems, other metrics such as the overlap metric or Hamming distance can be used [37].

For example, in Figure 2-7 we are trying to identify the class of green square(unknown), assuming that $k=4$, among the four nearest neighbours of the unknown, three members are class A and only one member who belongs to class B is close to the unknown. Hence, the unknown object is assigned to Class A.

In terms of model explainability, it is important to observe that predictions generated by KNN models rely on the notion of distance and similarity between examples, which

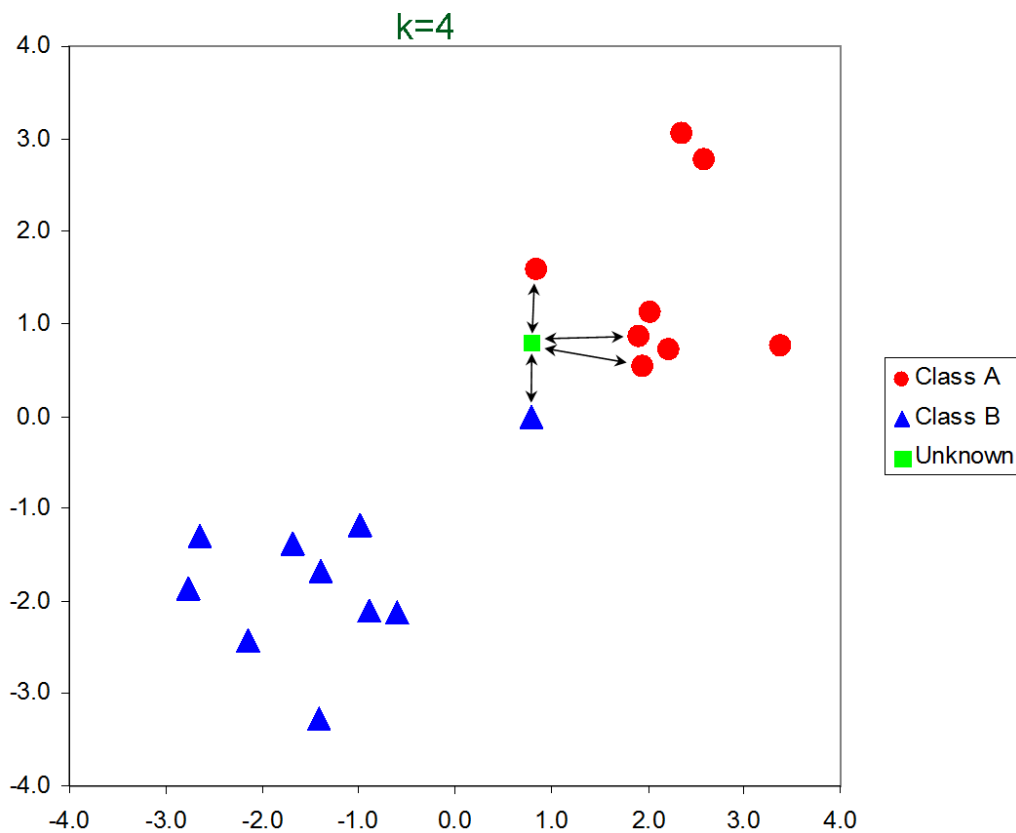


Figure 2-7: K-Nearest neighbour [37]

can be tailored depending on the specific problem being tackled. One must keep in mind that KNN's class of transparency depends on the features, the number of neighbours and the distance function used to measure the similarity between data instances [14]. A very high K impedes a full simulation of the model performance by a human user. Similarly, the usage of complex features or distance functions would hinder the decomposability of the model, restricting its interpretability [14].

2.2.5 Fuzzy Logic

Fuzzy Logic systems are rule-based systems which can be used to model imprecise and uncertain data. They try to mimic human thinking, although rather than trying to represent the brain's architecture as is done with a neural network, the focus is on how humans think in an approximate rather than precise way [1]. This is done by modelling uncertainty into if-then rules that describe a given behaviour into a human-readable format.

A good example would be a decision that a human might take while driving a car which could be the following rule *"If the distance to the car ahead is low and the road is slightly slippery Then slow down"*. The numerical meanings of "low", "close" and "slow down" will differ from driver to driver. Furthermore, if a driver were to be interviewed about the numerical values associated with these linguistic labels, they

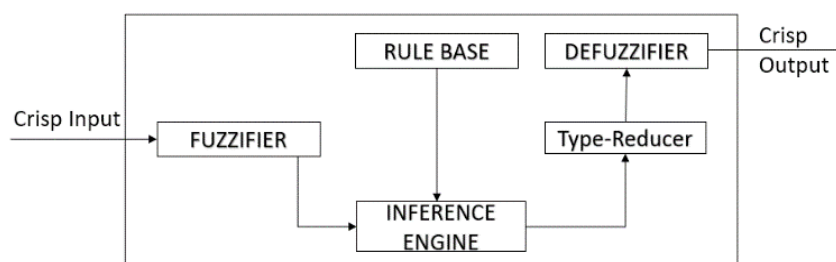


Figure 2-8: A Typical Interval Type-2 Fuzzy Logic System [1]

would struggle to quantify them. Amazingly, humans are nevertheless able to communicate with these ill-defined and vague linguistic labels and do not query the exact values when they discuss them. In fact, these uncertain concepts allow humans to be able to perform very sophisticated tasks such as driving cars or underwriting financial applications [1].

A Typical Fuzzy Logic System (FLS) is depicted in Figure 2-8; it contains five components: fuzzifier, rule base, inference engine, type-Reducer and a defuzzifier. A T1FLS is very similar to the system depicted in Figure 2-8, the only difference being that there is no type-Reducer in a T1FLS and it employs type-1 fuzzy sets in the input and output of the FLS [38].

The IT2FLS works in the following way: the crisp inputs in the data are first fuzzified into an input type-2 fuzzy set. A type-2 fuzzy set is also characterized by a membership function, but unlike a type-1 MF, the type-2 fuzzy sets are three dimensional and include a Footprint of Uncertainty (FOU). An interval type-2 fuzzy set [39], depicted in Figure 2-9, is used to represent the inputs and outputs of the IT2FLS. As seen in Figure 2-9, the membership for an Interval Type-2 fuzzy set outputs an interval, $[0.6, 0.8]$ rather than the crisp number produced by Type-1 fuzzy sets.

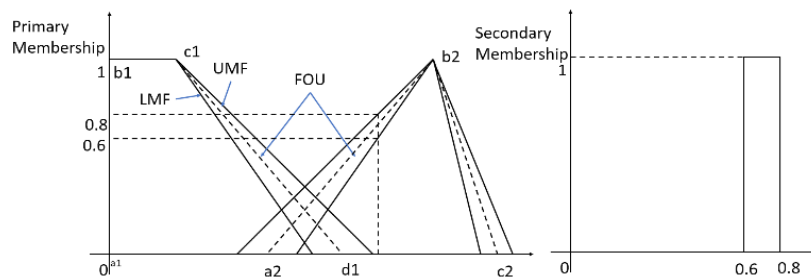


Figure 2-9: Interval Type-2 Fuzzy Set [1]

Once the inputs are fuzzified, the inference engine then activates the rule base using the input type-2 fuzzy sets and produces the output type-2 fuzzy sets. There is no

difference between the rule base of a T1FLS and a type-2 FLS except that the fuzzy sets are interval type-2 fuzzy sets instead of type-1 fuzzy sets.

In the final step, the output type-2 sets produced in the previous steps are converted into a crisp number. There are two methods for doing this; the first method is the conventional two-step process where the output type-2 sets are converted into type-reduced interval type-1 sets followed by defuzzification of the type reduced sets. The second method is the direct defuzzification process which was introduced because of the computational complexity of the first method. There are different types of type reduction and direct defuzzification [38].

It is important to note that the number of rules and the number of antecedents per rule have a bearing on the interpretability of the model. As the number of rules and antecedents increase the model becomes less and less interpretable and reducing the number and size of rules by optimisation can improve the interpretability but sometimes at the cost of accuracy. Fuzzy Logic will be discussed in much more detail in Chapter 3.

2.3 Model Induction

For AI models that are not inherently interpretable, one method to make them interpretable is to use model induction through model agnostic methods. These methods are designed such that any model can be plugged in with the intent of extracting explanations.

2.3.1 Local Interpretable Model-Agnostic Explanations (LIME)

LIME [40] is a locally interpretable surrogate model, where an interpretable AI model is trained to approximate the predictions of a black-box model. LIME

explanations are locally interpretable, i.e., the model is trained to provide good explanations for individual predictions, but it does not have to provide good explanations globally. This kind of accuracy is also called local fidelity [32]. Mathematically, local surrogate models with interpretability constraint can be expressed as follows [32].

$$\text{explanation}(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g) \quad (2.8)$$

Where f is the prediction of the original model (e.g. Deep Neural Network), g is the prediction of the interpretable model (e.g. decision tree) with π_x as the size of the neighbourhood around instance x , $\Omega(g)$ is the model complexity and L is the loss function. LIME is used to optimize the loss part with the complexity and size of neighbourhood determined by the user. The complexity of the model is generally kept low by using a low number of features to create the interpretable model.

For training the local surrogate model LIME models the behaviour of the underlying black-box model by inducing perturbations on the input, i.e., add small variation to the input data. LIME generates a new sample dataset (depicted as black dots in Figure 2-10) using these new inputs for the corresponding original black box model prediction (depicted as a yellow dot in Figure 2-10). The new sample dataset is weighted based on the proximity to the original target input, in the figure the size of the dots increases as they get closer to the black box model prediction. These weighted samples are then used to train an interpretable model to provide explanations. The interpretable model can be anything such as decision trees, fuzzy logic, etc. The interpretable model is trained on a reduced the number of features to reduce the complexity of the model.

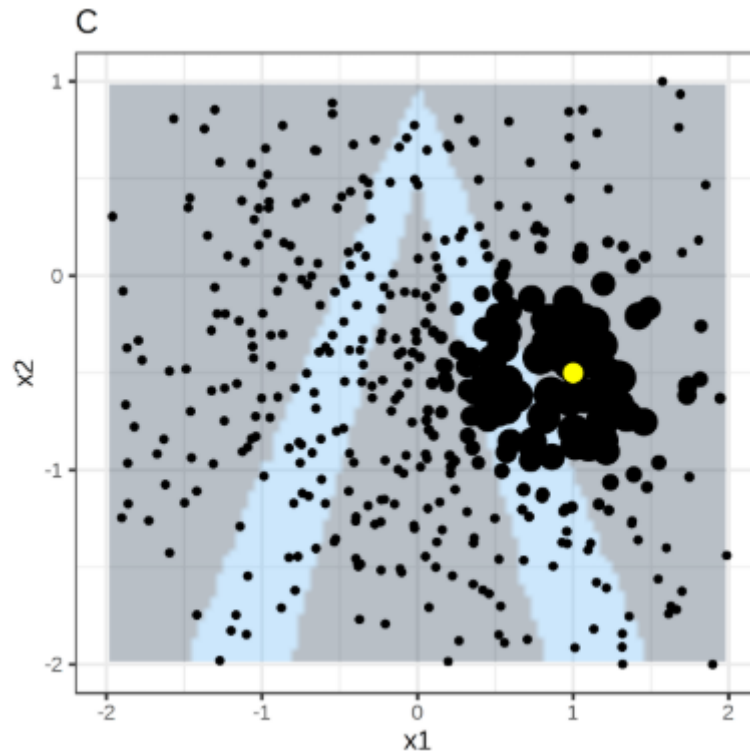


Figure 2-10: LIME Algorithm: Generated sample data for training local model [32]

Increasing the number of features improves the fidelity of the local surrogate but at the cost of explainability.

LIME uses an exponential smoothing kernel to define the weight or proximity of the new inputs to the original target input. The kernel width defines the size of the neighbourhood, i.e., the new inputs must be within the neighbourhood to influence the interpretable model. Determining the kernel width is a design parameter that greatly influences the fidelity of the model. Hence, it has to be chosen with care as changes to the kernel width have a great influence on the explanations [32]. The next problem with LIME is that the explanations are not consistent, i.e., when the same process is repeated, the explanations provided can vary greatly [32].

2.3.2 Anchor Local Interpretable Model-Agnostic Explanations

Anchor Local Interpretable Model-Agnostic Explanations (aLIME) [42] [41] is a method where model agnostic explanations are provided by easy to understand if-then rules, and these rules are called anchors. An Anchor is a rule that sufficiently "anchors" the prediction locally, i.e., changes to variables other than the anchors does not change the prediction. Like LIME, the aLIME approach uses a perturbation-based strategy to generate if-then rules or anchors as local explanations for predicting the behaviour of black-box models. These rules are reusable since they are scoped: anchors include the notion of coverage, stating precisely to which other, possibly unseen, instances they apply [32]. An anchor can be formally defined as follows [32] [41].

$$E_{D_x(z|A)}[1_{f(x)=f(z)}] \geq \tau, A(x) = 1 \quad (2.9)$$

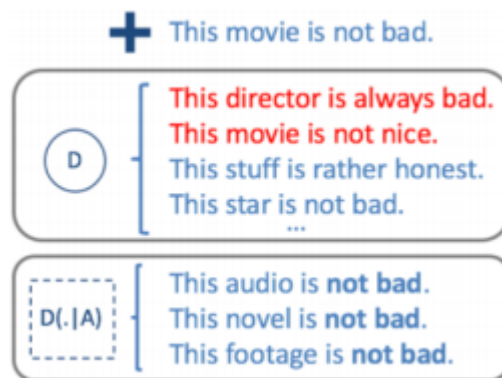


Figure 2-11: Anchors generated for keywords "not" and "bad" [41]

Where x represents the prediction being explained, A represents a set of antecedents, i.e., a rule or anchors, such that $A(x) = 1$ when all the feature predicates of A correspond to x 's feature values, f represents the AI model being explained, which can be queried to get the predictions for x and its perturbations. $D_x(\cdot|A)$

indicates the distribution of neighbours x matching A . $0 < \tau < 1$ specifies a threshold and only rules that achieve a local fidelity greater than τ are considered valid [32].

For example, in Figure 2-11, $x = \text{"This movie is not bad"}$, $f(x) = \text{Positive}$, $A(x) = 1$ where $A = \{\text{"not"}, \text{"bad"}\}$. Let $D(\cdot | A)$ denote the conditional distribution when the rule A applies (e.g. similar texts where “not” and “bad” are present, Figure 2-11 bottom) [41]. Then the anchor A is easy to apply: if words “not” and “bad” are present, then the model will predict positive and if either (or both) words are not present then the model prediction is unknown.

Although anchors mathematical description may seem straightforward, generating particular rules is infeasible as it would require evaluating $1_{f(x)=f(z)}$ for all $z \in D(\cdot | A)$ [32]. Therefore, the authors propose to introduce a probabilistic definition (equation (2.10)) where anchors satisfy the precision constraint with high probability.

$$P(\text{prec}(A) \geq \tau) \geq 1 - \delta \text{ with } \text{prec}(A) = E_{D_x(z|A)}[1_{f(x)=f(z)}] \quad (2.10)$$

If multiple anchors satisfy the criterion, the anchor which describes a larger part of the search space is preferred, i.e., one with larger coverage [41].

The main advantage of the aLIME approach over LIME is that it uses IF-Then rules which are easy to interpret (even for laypersons) [32]. However, the IF-Then rules anchor model presented in [41], use crisp logic and thus will struggle with variables which do not have clear, crisp boundaries, like income, age, etc. Also, the approach in [41] will not be able to handle models generated from a big number of inputs. Furthermore, explaining the prediction with just an anchor IF-Then rule does not give a full picture about the decision as for example in case of classification problems, there

are always pros and cons which humans weigh in their minds and take a decision accordingly [1].

2.3.3 SHAP

SHAP (SHapley Additive exPlanations) [43] ranks importance or relevance each feature has to the prediction or output of the model to be explained. It is based on calculating the contribution of input features to each prediction using Shapely values (Game Theory) [44]. Shapely values tell us how to distribute the prediction among the features fairly. SHAP specifies the explanation as [44]:

$$g(z') = \phi_0 + \sum_{j=1}^M \phi_j z'_j \quad (2.11)$$

where g represents the explanation model, $z' \in \{0,1\}^M$ represents the simplified feature vector, M represents the maximum feature size and $\phi_j \in R$ represents the feature attribution for a feature j , the Shapely values. The simplified features used here could be individual features or groups of features depending on the type of data; for example, for image data, super pixels are used as the simplified features.

To calculate the Shapley values, we simulate that only some features values are playing ("present") and some are not ("absent") creating the simplified feature vector [32]. A linear model is then trained to represent these values, to calculate the ϕ_j (shapely values). In the simplified feature vector, when the feature values are present, then the value of that feature is picked from the input x for which the explanation is being generated. When the feature values are absent, they are replaced by random values from some of the other inputs to the original model.



Figure 2-12: SHAP explanations [43]

One method for calculating the shapely values is by using Kernel SHAP. Kernel SHAP calculates the shapely values in 5 steps listed below [32].

- A Sample simplified feature vector is created $z'_k \in \{0,1\}^M$, $k \in \{1, \dots, K\}$ (1 = feature present in coalition, 0 = feature absent).
- A prediction is generated for each z'_k by first converting z'_k to the original feature space (1's are replaced by the feature values from x (original input for which we are generating the explanation), 0's are replaced by feature values from a random input) and then applying the model $f : f(h_x(z'_k))$
- Compute the weight for each z'_k with the SHAP kernel.
- Fit weighted linear model.
- Return Shapley values ϕ_j , the coefficients from the linear model.

The intuition behind this is: We can learn about individual features by studying their effects in isolation. If a simplified feature vector consists of a single feature, we can learn about that features' isolated main effect on the output prediction. If a simplified feature vector consists of all but one feature, we can learn about that features' total effect (main effect plus feature interactions). The SHAP kernel is defined as follows:

$$\pi_x(z') = \frac{(M-1)}{(M \text{ choose } |z'|) |z'| (M - |z'|)} \quad (2.12)$$

Where M represents the maximum simplified feature vector size and $|z'|$ represents the number of features present in instance z' . Lundberg and Lee show that a linear regression with this kernel weight yields Shapley values [43] [32].

The linear model g is trained by optimizing the following loss function L :

$$L(f, g, \pi_x) = \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi_x(z') \quad (2.13)$$

where Z represents the training data. The estimated coefficients of the model, the ϕ_j are the Shapley values.

The main advantage of SHAP is that SHAP has a solid theoretical foundation in game theory. If the prediction is fairly distributed among the feature values, then we get contrastive explanations that compare the prediction with the average prediction.

The disadvantages of SHAP are that first, it takes a long time to calculate the SHAP values, so it is impractical to use it for many instances. The second disadvantage is that it ignores the correlation between the input features; this could lead it to put emphasis on unlikely data points [32]. The third disadvantage is that when the inputs have many features, it can become very hard to distinguish between the features using the feature importance scores.

2.3.4 Partial Dependence Plot

The partial dependence plot (PDP) shows the effect of one or two features on the output of a machine learning model[45]. A partial dependence plot can show the relationship between an input feature and the output. For example, when applied to a linear regression model, partial dependence plots always show a linear relationship between the selected features and the output [32].

The partial dependence function for regression is defined as [32]:

$$\hat{f}_{x_s}(x_s) = E_{x_c}[\hat{f}(x_s, x_c)] = \int \hat{f}(x_s, x_c) dP(x_c) \quad (2.14)$$

The x_s represents the features that are being plotted in the PDP and x_c represents the rest of the input features. Usually, only one or two features are plotted in the PDP represented by the set S. The feature(s) in S are those features for which we want to know the effect on the prediction. PDP is drawn by marginalizing the output predictions over the distribution of the features in set C so that the plot only shows the relationship between the input features in set S and the output predictions. This allows us to create plots that are only dependent on the input features represented in the set S [32].

The partial function $\hat{f}_{x_s}(x_s)$ is estimated by calculating averages in the training data, also known as the Monte Carlo method [32]:

$$\hat{f}_{x_s}(x_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x_s, x_c^i) \quad (2.15)$$

Where x_c^i represents the features which are marginalised over the output predictions. n is the number of records in the dataset.

In classification problems, the partial dependence plot displays the probability of a certain class. An easy way to deal with multiple classes is to draw one plot per class [32].

One of the drawbacks of the PDP is that we can only draw two features per plot. This means that for inputs with many features, we will have to create a large number of PDP plots. This means that PDP becomes extremely hard to understand for inputs with a large number of features. The second major drawback is that the features for which the plots are being created are assumed not to be correlated to the rest of the features. If the features are correlated, then the plot might contain points that are unlikely[32].

2.3.5 Individual Conditional Expectation

ICE (Individual Conditional Expectation) [46] plots are equivalent to PDP for individual instances. An ICE plot (depicted in Figure 2-13) visualises each individual instance or prediction as a line, compared to one line overall in PDP. A PDP (depicted as the Yellow line in Figure 2-13) is the average of all lines of an ICE plot.

The values for a line on the ICE plot can be computed by keeping all other features the same, creating variants of this instance by replacing the values (generally from a grid) and fetching the predictions for the new input from the black box [32].

PDPs can show the average relationship between a feature and the model predictions. This only works well if the interactions between the features for which the PDP is calculated and the other features are weak. In the case of interactions, the ICE plot will provide much more insight [32].

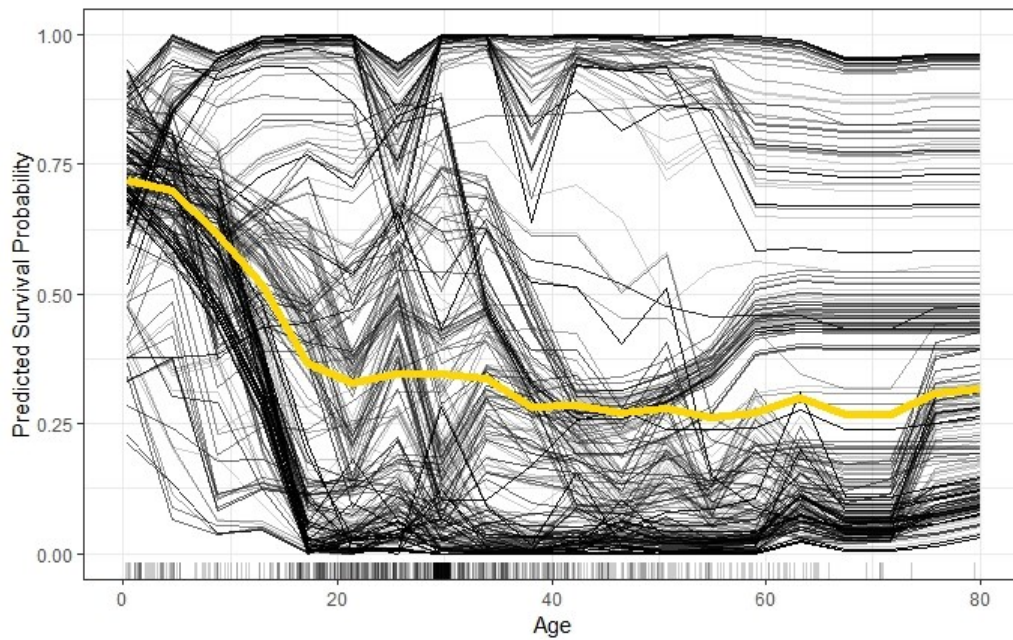


Figure 2-13: ICE plot of survival probability by Age [32]

The advantage of Individual conditional expectation curves is that they are more intuitive to understand than partial dependence plots as one line represents the predictions for one instance if we vary the feature of interest. They can also uncover heterogeneous relationships, unlike PDPs [32].

The disadvantage of ICE plots is that they can only display one feature meaningfully because plotting the third dimension requires the use of a three-dimensional plot and we would not be able to distinguish the features of the plot. Multiple ICE plot for each feature can be drawn, but as the number of features increases, it becomes increasingly difficult to understand.

2.3.6 Interpretable Mimic Learning

Another approach to model induction is Interpretable mimic learning [47]. The idea behind this approach (depicted in Figure 2-14) is to use knowledge distillation to transfer the knowledge of the DNN or another black-box model to a simpler, more interpretable model. This is done by training the student/mimic model on the soft labels

generated from the outputs of the parent/base model. The soft label, in contrast to a hard label from the raw data, is a real value output of the teacher model, whose values are usually in the range $[0,1]$ [47].

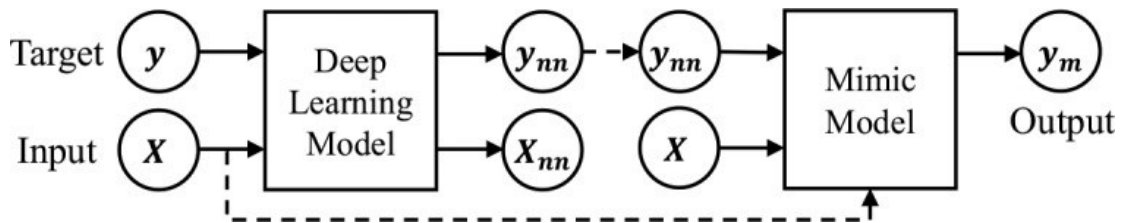


Figure 2-14: Interpretable Mimic Learning

The authors note that a shallow neural network is not as accurate as a Deep network when trained on the same training data. However, the accuracy of the shallow neural network trained on soft labels is similar to or better than the deep model [47]. The authors suggest that this is because some potential noise and error in the training data (input features or labels) may affect the training efficacy of simple models [47]. The teacher model may eliminate some of these errors, thus making learning easier for the student model. Soft labels from the teacher model are usually more informative than the original hard label (i.e. 0/1 in classification tasks), which further improves the student model [47].

The authors choose gradient boosting trees (GBT) as the student model. Gradient boosting machines are a method which trains an ensemble of weak learners to optimize a differentiable loss function by stages[47]. This is done to ensure that the student model retains the accuracy of the original DNN.

We must note here that GBT is an ensemble of decision trees. This means that, although it retains some of the interpretability of the decision trees, it is a much more

complex to interpret. Furthermore, it becomes more complex to interpret as the size of the DNN increases.

2.4 Hybrid Deep Learning and Fuzzy Logic Systems

Semantic representation of the predictions such as the use of natural language seems to be the best way of generation explanations [16]. One way of generating these types of explanations is to use IF-Then rules, which are intuitive to humans and usually require low effort to comprehend and apply [42]. However, IF-Then rules that use crisp logic will struggle with variables that have noisy boundaries, such as income, age, etc. Furthermore, explaining the prediction with just an IF-Then rule does not give a full picture about the decision as an example in case of classification problems, there are always pros and cons which humans weigh in their minds and take a decision accordingly. Also, another major problem with crisp logic is the inability to understand the model behaviour in the neighbourhood of the instance and how the prediction can change if certain features are changed, etc[1].

From the above discussion, offering users if-then rules that include linguistic labels appears to be an approach that can facilitate the explainability of a model. The AI technique that satisfies these conditions is the fuzzy logic system (FLS). However, when there are many features in the input data FLS can become opaque as modelling such data will require many rules. There are, of course, methods to mitigate this problem using methods such as [48, 49] but they still require feature selection to be effective [1].

A good way to mitigate the problems mentioned about is to combine connectionist and symbolic paradigms [14, 50], i.e., combining high performance AI models such as deep learning with fuzzy logic systems. There are several advantages to this approach.

- Since fuzzy logic systems use linguistic IF-Then rules, it makes the system inherently interpretable.
- The system can be trained using both labelled and unlabelled data.
- The system can handle the uncertainties inherent in the data since it uses fuzzy logic.
- Since the system is composed of transparent rules and membership functions, which can be relatively easily changed, if there are any problems.
- There is no need for feature selection for inputs with a large number of features.

Combining multiple machine learning algorithms to solve problems has been an active branch of AI research for several years [50, 51]. Deep learning has been combined with many AI algorithms such as random forests[52], decision trees [53] etc, including with FLSs. In the following sections, we introduce some of the techniques or method that have combined Deep learning with Fuzzy logic in the literature.

2.4.1 Fuzzy Restricted Boltzmann Machine

A Fuzzy Restricted Boltzmann Machine (FRBM) [54] is a neural network where the connection weights and bias of the network are fuzzy parameters (depicted in Figure 2-15). The FRBMs can then be stacked to create a Deep Network. This method provides several advantages over a Restricted Boltzmann Machine (RBM). First, the FRBM is better than RBM in modelling probabilities; specifically, the RBM is treated

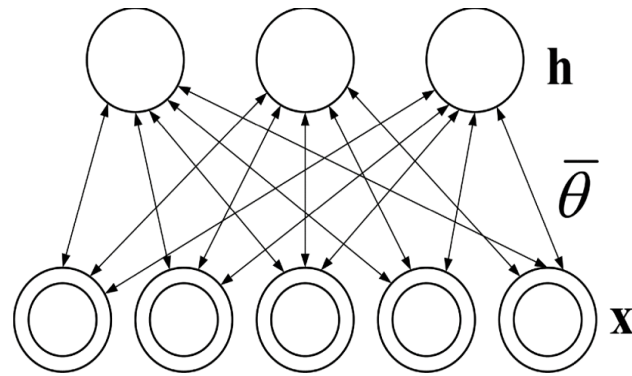


Figure 2-15: Fuzzy Restricted Boltzmann machine [54]

as a special case of FRBM when no uncertainty exists in the FRBM model. Second, the FRBM model is more robust when compared to an RBM as it has the inherent ability to model uncertainty; it will be more robust when encountering noisy data. These advantages spring from the fuzzy extension of the connection weights and bias of the neural network layers [54]. The Interval type-2 versions of the FRBM have been proposed in [55, 56]. These advantages spring from the fuzzy extension of the connection weights and bias of the neural network layers [54], which are shown to outperform the FRBM.

The main drawback of this system and its IT2 versions is that it does not enhance the interpretability of the Deep Neural Network and only uses the uncertainty modelling capabilities of the Fuzzy logic. This means that the weights and biases cannot be directly used to gain insight into the relative importance of the inputs or underlying relationships between the inputs and the predicted outputs [57]. Additional tools such as the methods described in Sections 2.1 or 2.3 have to be used to gain insight into the model and provide explanations for the predictions.

2.4.2 Fuzzy Deep Neural Network

Fuzzy Deep Neural Network (FDNN) [58] is a method where the neural network representation and a fuzzy representation are trained at the same time. As depicted in

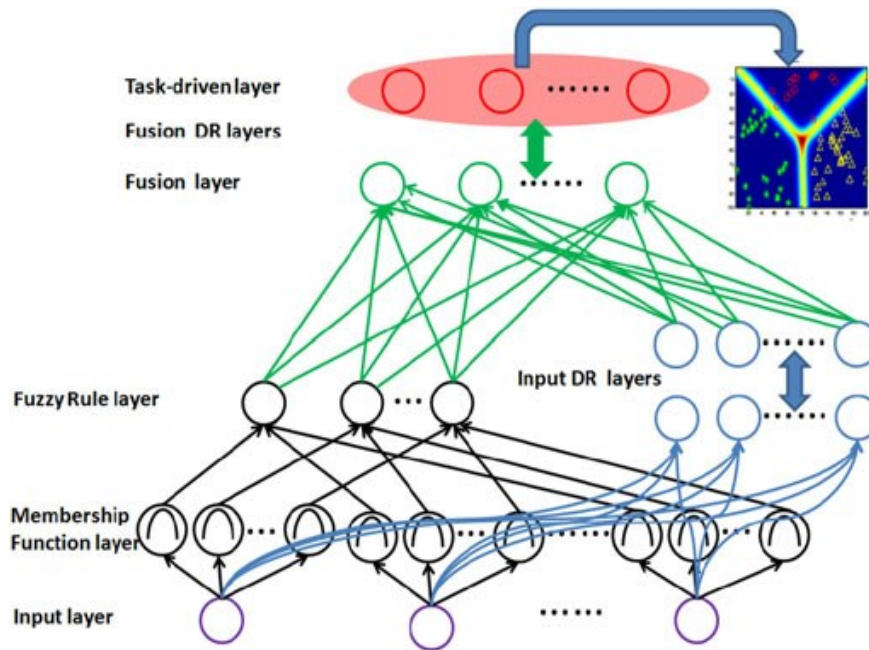


Figure 2-16: Fuzzy Deep Neural Network [58]

Figure 2-16, The outputs of these two representations are combined and become the input for the next set of layers called the fusion layer. According to the authors, there are several advantages to using Fuzzy Deep Neural Network, first, since a fuzzy representation of the data is used the uncertainties in the data. Second, since the nodes in the various layers interact with each other in uncertain ways, the fuzzy representation will be able to account for these uncertainties. Third, since the parameters that represent the relationship between nodes from adjacent layers are fuzzy numbers and the learning process of the fuzzy representation is extended to the wider network, it results in an improvement in the fitness of the joint probability distribution. This, when combined with the inherent advantages of Deep learning, the authors have shown that the FDNN has superior performance when dealing with noisy or uncertain datasets [58].

The main drawback of FDNN is that even though the fuzzy layers are interpretable, the neural network layers are difficult to interpret. So, when the output of the two different layers is combined, the model becomes opaque. Further, since the FDNN uses

fusion layers, the methods described in Section 2.1 cannot be used to provide an explanation for the outputs. So, the methods used in Section 2.3 have to be used to provide explanations for the FDNN. Hence, this method can only be used to enhance the performance of Deep learning and not its interpretability.

2.4.3 Fuzzy Deep Learning

Fuzzy Deep Learning (FDL) (depicted in Figure 2-17) is a model proposed by Seonyeong Park and colleagues in [59]. In their paper they propose a four-layer system where the first layer calculates the membership grades of the inputs using membership functions, the second layer calculates the firing level of the rules using the t-norm operation, the third layer computes linear regression functions by normalizing weights and finally, the fourth layer provides the output by summing the outcomes according to all fuzzy if-then rules [59]. The main difference between this technique and the neuro-fuzzy techniques is the use of the linear regression function in the third layer. The authors show that this new method outperforms other traditional methods in predicting tumour movement during radiotherapy [59].

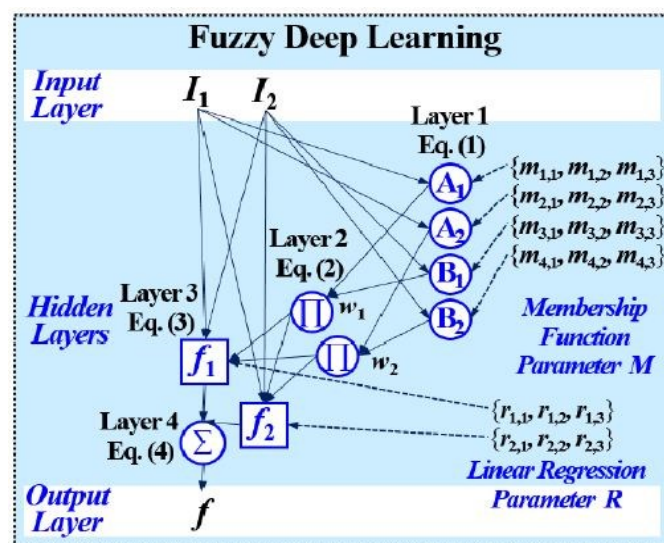


Figure 2-17: Fuzzy Deep Learning [59]

There are multiple drawbacks to this method, such as it can only be used for supervised learning, and there is no mention of how the output is connected to the inputs. So, there are improvements required before this method can be used for XAI applications.

2.4.4 Takagi Sugeno Deep Fuzzy Network

The Takagi Sugeno Deep Fuzzy Network (TSDFN) (depicted in Figure 2-18), was proposed by Shreedhar Kumar Rajurkar and Nishchal Kumar Verma in [60]. The authors explain their concepts using a three-layered TSDFN where the layers are input, hidden and output. They propose that the number of nodes in the hidden layer may vary based on the applications and each node in the hidden layer is a Takagi Sugeno Fuzzy Logic System (TS FLS). Furthermore, the output layer is a single node or multiple nodes depending on the desired output, and these nodes are also TS FLS systems. This system is trained using a backpropagation algorithm.

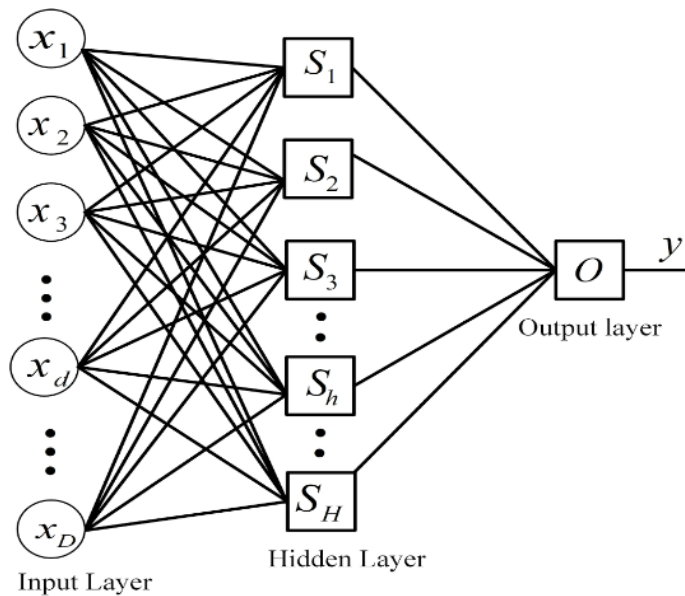


Figure 2-18: Takagi Sugeno Deep Fuzzy Network [60]

The system exhibits some level of interpretability, but since multiple layers are used, we do not know what the output of the hidden layers or input of the output layer represents. Another problem is that of rule explosion, i.e., the number of rules and membership functions increases exponentially as the number of features in the input increases. Which means that it becomes difficult to interpret as the number of input features increases. Hence, there are improvements needed before this system can be used for XAI applications.

2.4.5 Fuzzy Deep Belief Network

The Fuzzy Deep Belief Network (FDBN) (depicted in Figure 2-19), was proposed by Shusen Zhou and colleagues in [62]. The authors propose a system where they first train a Deep Belief Network (DBN) using greedy layer-wise pre-training [61]. Then two membership functions are created based on the mapping results of the trained

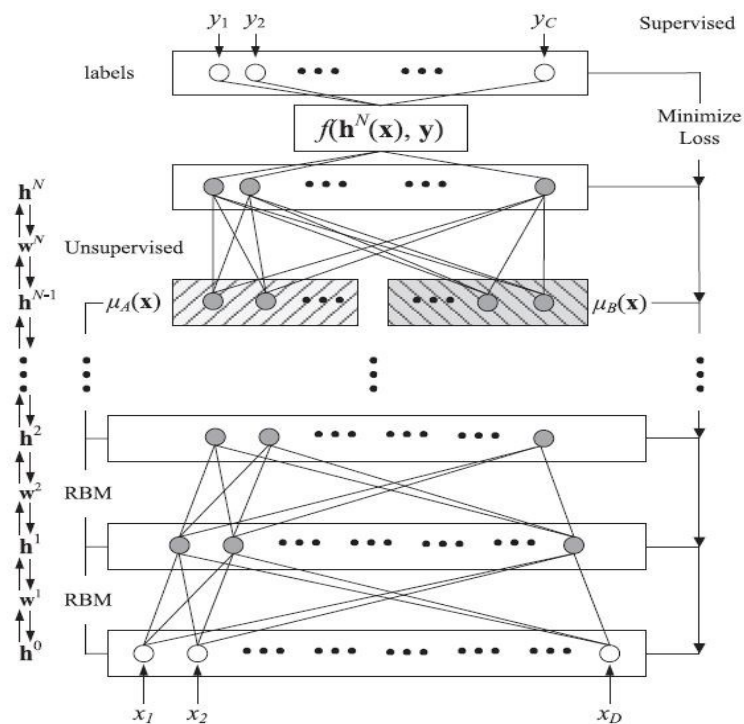


Figure 2-19: Fuzzy Deep Belief Network [61]

DBN. Once the membership functions are trained, they are used to activate the (n-1) layer of the DBN.

The authors claim that this method provides comparable performance to other AI algorithms [62]. The biggest drawback of this method is that it does not introduce any interpretability to the Deep Belief Network it uses fuzzy inputs to improve the accuracy of the DBN. Hence, this system cannot be used for XAI applications.

2.4.6 Active Fuzzy Deep Belief Network

Active learning is a machine learning technique that selects the most informative samples for labelling and uses them as training data [63]. Active FDBN combines Active learning with FDBN systems [62]. Where first the FDBN is trained using a labelled dataset and all the unlabelled data. Once the first FDBN is trained, the unlabelled data set is analysed, and some of the unlabelled data is converted into labelled data based on a set of criteria [62]. After that, the FDBN is retrained using the newly labelled and the unlabelled data. The authors have shown that using this method improves the performance of the FDBN. This method has the same drawback as the FDBN method, i.e. it is not interpretable hence cannot be used for XAI applications.

2.4.7 Pythagorean Fuzzy Deep Boltzmann Machine

The Pythagorean Fuzzy Deep Boltzmann Machine (PFDBM), was proposed by Yu-Jun Zheng and his colleagues [64]. In this system, the weights in a Deep Boltzmann Machine are replaced with Pythagorean Fuzzy Numbers [65] represented by the weights in Figure 2-20.

The authors claim that this algorithm provides competitive performance when compared to other algorithms in the field of passenger profiling [1]. They put forth

three reasons for using fuzzy logic in their model. First, fuzzified neural networks can handle inputs with fuzzy (labelled) or incomplete features [66], which are inevitable in passenger profilers [64]. Second, fuzzy parameters can improve the representation ability of DBM by supporting fuzzy probability distribution [67] over cross-layer units, as the principle of incompatibility asserts that high precision is incompatible with high complexity in dealing with complex systems, such as passenger profilers. Thirdly, the parameter learning of fuzzy DBM has a larger space than its crisp counterpart and thus will be more helpful in utilizing the merits of deep learning [64].

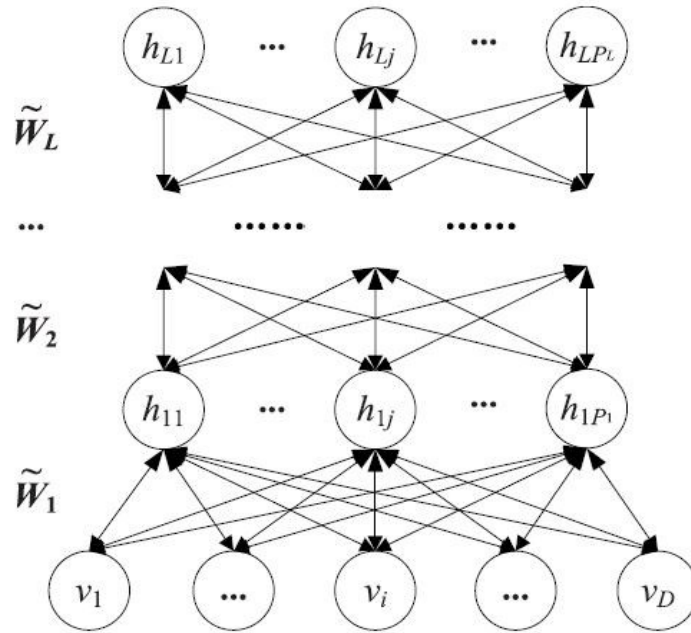


Figure 2-20: Pythagorean Fuzzy Deep Boltzmann Machine [65]

Mathematically the PFDBM can be represented as follows, given $\theta = [\mathbf{W}_1, \dots, \mathbf{W}_L]$ denotes the fuzzy parameters of the PFDBM shown in Figure 2-20, the fuzzy energy state $\{\mathbf{v}, \mathbf{h}_1, \dots, \mathbf{h}_L\}$ of the model is defined as follows.

$$\tilde{E}(\mathbf{v}, \mathbf{h}_1, \dots, \mathbf{h}_L, \theta) = -\mathbf{v}^T \mathbf{W}_1 \mathbf{h}_1 - \sum_{l=2}^L \mathbf{h}_{l-1}^T \mathbf{W}_l \mathbf{h}_l \quad (2.16)$$

This method has a major drawback in that it does not utilize the interpretability of Fuzzy models and only tries to solve the uncertainty and incompleteness of the training data. This means that the weights and biases cannot be directly used to gain insight into the relative importance of the inputs or underlying relationships between the inputs and the predicted outputs [57]. Moreover, additional tools such as the methods described in Sections 2.1 or 2.3 have to be used to gain insight into the model and provide explanations for the predictions. Hence, this method cannot be used as-is for XAI applications, and it will need further research to input interpretability into the algorithm [1].

2.5 Summary

This chapter defines and explains the need for Explainable AI, and it also explains the three main methods that are being explored to achieve Explainable AI, Deep Explanations, Interpretable Models and Model induction.

It explains that the methods used to achieve Deep Explanations, provide details about all the input features that affect the output and that there is no structure to these details. This might be useful in image recognition, but in other problems, it might be hard to distinguish the most important features.

It explores the various existing interpretable models such as decision trees, fuzzy logic etc, and explains how they have lower performance when compared to their opaque counterparts and that they can also become opaque for inputs with a large number of features.

It explores the various model induction methods in the literature and explains that these methods are not suitable for use in regulated applications as they are generally achieved using surrogate models.

It suggests an alternative to the above methods, combining the predictive power and embedded feature selection of the Deep Learning models with the explainability of Fuzzy Logic Systems. It explores the various methods that have been used in the literature to achieve this. It highlights that these methods have primarily focused on increasing the performance of Deep learning by taking advantage of the uncertainty modelling capability of fuzzy logic while not enhancing the explainability of the system.

The next chapter will describe the Fuzzy Logic Systems in detail.

Chapter 3. An Overview on Fuzzy Logic

Fuzzy Logic (FL) was first introduced by Lotfi Zadeh in his 1965 seminal paper ‘Fuzzy Sets’ [68]. Zadeh describes Fuzzy sets as the non-binary classification of elements to classes; i.e., instead of classifying an element of a set as either belonging to a class or not, there is instead a degree of membership to that class [69].

An example of non-binary classification might be “height” if there are three classes that define the term height, *short*, *average* and *tall*. One way to classify people as tall could be to say that anyone over 1.9 meters is tall. However, what about someone who is 1.89 meters? Should this person be classified as average? This classification would also change based on the context such as gender, age, location etc. In these situations, we use a non-binary classification or fuzzy sets to determine the degree of membership to class rather than use true or false to determine the class.

Zadeh goes on to state that such imprecisely defined classes exist throughout the real world and play an essential part in human reasoning and decision-making particularly when it comes to pattern recognition, communication and abstraction [69].

Fuzzy sets and systems are now widely used in many industries and fields to solve practical problems and are subjects of intense research by academics all over the world. Furthermore, Fuzzy rule-based systems, which are derived from fuzzy sets, have been demonstrated as a powerful design methodology [38].

A rule-based fuzzy logic system (FLS) processes its inputs nonlinearly, and an essential facet of FLSs is modelling imprecise and uncertain data and representing it with a set of if-then rules to describe a given behaviour in human-readable form. A

Fuzzy rule has the structure “IF p THEN q ”, in which p is called the rule’s *antecedent* and q is called the rule’s *consequent*.

An example of a fuzzy rule is “IF service is good and food is delicious THEN tip is high”. Here the terms “good”, “delicious” and “high” are called linguistic labels, and these values can be hard to define as everyone has different ideas about what a linguistic label constitutes. So, we use fuzzy sets and membership functions that describe them to define these linguistic labels. Two main types of fuzzy sets are used in this thesis, type-1, and type-2. Type-1 fuzzy sets are described by membership functions that are totally certain, whereas type-2 fuzzy sets are described by membership functions that are themselves fuzzy. The latter can be used to quantify the different kinds of uncertainties that can occur [38].

A FLS that is described entirely in terms of type-1 fuzzy sets is called a type-1 FLS, whereas a FLS that is described using at least one type-2 fuzzy set is called a type-2 FLS. Type-1 FLSs are unable to directly handle rule uncertainties because they use type-1 fuzzy sets that are certain. Type-2 FLSs, on the other hand, can be used in circumstances where it is difficult to determine the exact membership functions of a fuzzy set [38].

3.1 Uncertainty

Fuzzy logic has been designed to handle uncertainty in many forms. In general, uncertainty comes in many guises and is independent of the kind of fuzzy logic, or any kind of methodology, one uses to handle it [70] [38].

The following are the sources of uncertainty that can occur in FLS [38].

- Uncertainty about the meaning of the words used in the rules.
- Uncertainty about the consequents used in the rules.
- Uncertainty about the measurements that activate the FLS.
- Uncertainty about the data, eg., missing or unreliable data, that are used to tune the parameters of the FLS.

Uncertainty about the meaning of words might arise because words mean different things to different people [71], which means that FL must somehow use this uncertainty when it computes with words [38]. Type-1 FLS handles uncertainties about the meaning of words by modelling the words as type-1 membership functions. Once type-1 membership functions are chosen, all uncertainty associated with it disappears [38]. On the other hand, Type-2 FLS handles uncertainties about the meaning of words by modelling the uncertainties. Although this is also totally precise, it includes a footprint of uncertainty that provides new degrees of freedom that allows type-2 FLS to handle uncertainty in new ways [38].

Uncertainty about the consequents arises because consequents are sometimes obtained from experts, by means of knowledge mining, or are extracted directly from data. Because experts do not all agree, a survey of experts usually leads to a histogram of possibilities for the consequent of rules. This type of uncertainty can be handled by a type-2 FLS [38].

Uncertainty in measurement can occur due to noise, as measurements are usually corrupted by noise, they can also occur due to the limitation of the measuring (for example, sensor resolution) system [38] [72]. Uncertainty in measurement can be

handled within the framework of a FLS by modelling them as fuzzy sets (either type-1 or type-2) [38].

Finally, a FLS contains many parameters whose values must be set before the FLS is operational. One of the ways to do this is to make use of a set of data or training set. This set usually contains input-output pairs, and if these pair are generated from measurements, then they contain the same uncertainty as the measurements that trigger an FLS [38]. This means that the FLS must be trained using unreliable data. This type of uncertainty can be handled by using type-2 FLS [38].

3.2 Type-1 Fuzzy Logic Systems

A fuzzy logic system (FLS) can be defined as a nonlinear mapping of an input feature vector into a scalar output. In a fuzzy logic system (depicted in Figure 3-1) the crisp inputs in the data are first fuzzified into an input fuzzy set; singleton fuzzification is commonly used as it simplifies the computation. Once the inputs are fuzzified, the inference engine then activates the rule base using the input fuzzy sets and produces the output fuzzy sets. In the final step, the output fuzzy sets produced in the previous steps are converted into a crisp number. Crisp numbers are real numbers with no

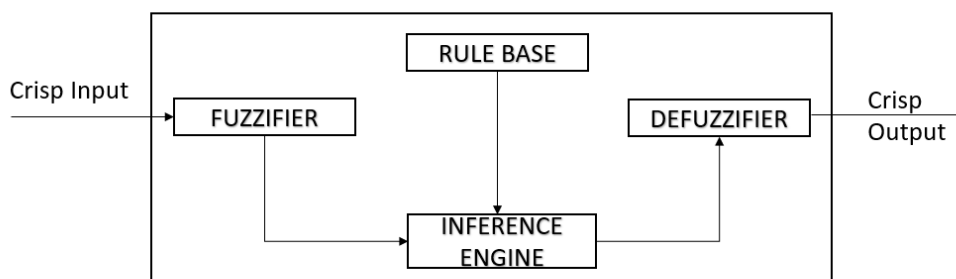


Figure 3-1: Type-1 Fuzzy Logic System [38]

uncertainty associated with them. These numbers are essential for engineering applications, as an example in control systems. For example, the speed of a motor in Rotations per minute, the temperature in Celsius etc, are all examples of crisp number and they are needed as either inputs or outputs to the fuzzy logic system [69] [38].

3.2.1 Linguistic Variables

Zadeh describes linguistic variables as “variables whose values are not numbers but words or sentences in natural or artificial language. The motivation for the use of words or sentences rather than numbers is that linguistic characterizations are, in general, less specific than numerical ones.” [68].

This means that numerical values can be classified under certain linguistic variables and still retain its contextual meaning. Given we use classification rather than true or false in fuzzy logic, a numerical value may be classified into two or more linguistic labels, but with different degrees of membership [69].

In fuzzy logic systems, a linguistic variable is fully characterised by a quintuple (u, X, U, g, m) , where u is the name of the variable, X is the set of linguistic terms of the linguistic variable u . g is the syntactic rule for generating the linguistic terms, and m is the rule that assigns the linguistic term $x \in X$ its meaning, $m(x)$, is the fuzzy set on U , that is, $m: X \rightarrow F(U)$, where $F(U)$ denotes the set of fuzzy sets of U . u is generally referred to as the linguistic variable [38].

We can use an example to illustrate: Let pressure (p) be interpreted as a linguistic variable. One might interpret this linguistic variable as the following terms $p(\text{pressure}) = [\text{weak, low, okay, strong, high}]$ each term in the variable $p(\text{temperature})$ is characterised by a set in the universe of discourse $U = [100 \text{ psi, } 2300 \text{ psi}]$. We might

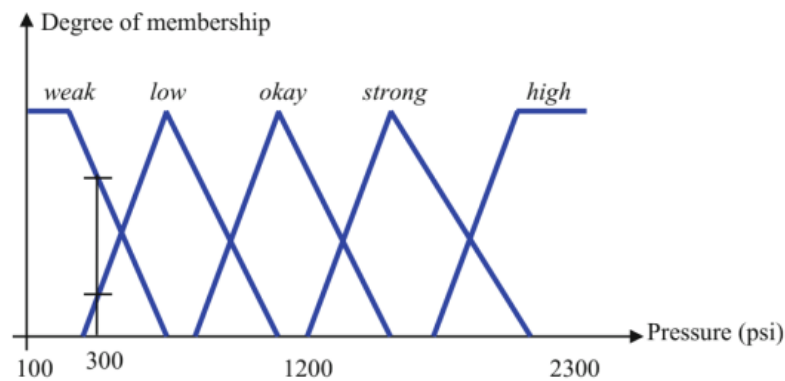


Figure 3-2: Membership Functions for Pressure [38]

interpret weak pressure as below 300 psi, low pressure as between 300 psi and 1000 psi, okay pressure as pressure between 700 and 1500 psi, strong pressure as between 1300 and 2000 psi and high pressure as above 1700 psi.

These terms can be characterised as fuzzy sets whose membership functions are shown in Figure 3-2. Measured pressure (p) values lie along the x-axis. In Figure 3-2, a vertical line from any measured value intersects at most, two linguistic labels or membership functions (see Membership Functions). For example, when $p = 300$ psi pressure can be described by the linguistic labels weak and low, but to different degrees of membership [38].

3.2.2 Membership Functions

Membership functions are functions that quantify the degree of membership of a numerical value to a linguistic term. Membership functions have the mathematical notation $\mu_F(x)$ [38].

The most common geometric shapes used for defining membership functions are piecewise linear functions such as triangular or trapezoidal, Gaussian, bell-shaped, some of these are shown in Figure 3-3. Membership functions can be defined using a

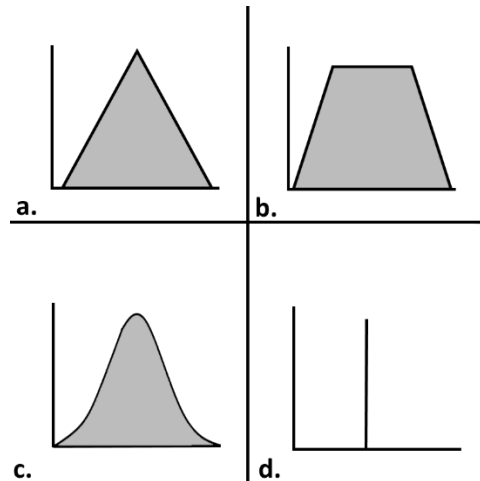


Figure 3-3: Types of Membership Function a) Triangular b) Trapezoidal c) Gaussian d) Singleton

[69]

variety of methods. Membership functions are sometimes chosen by the user based on their experience. The membership functions chosen this way vary quite drastically depending on the user's experiences, perspectives, cultures, etc. Other method include polling a set of people, clustering using methods such as fuzzy c means clustering [73], designed using optimisation procedures [74], etc.

Triangular Membership Functions are defined using the following formula [38].

$$\mu_A(x) = \mu_A(x; a, b, c) = \begin{cases} (x-a)/(b-a) & \text{if } a \leq x < b \\ (c-x)/(c-b) & \text{if } b \leq x \leq c \\ 0 & \text{if } x > c \text{ or } x < a \end{cases} \quad (3.1)$$

Trapezoidal Membership functions are defined using the following formula [38].

$$\mu_A(x) = \mu_A(x; a, b, c, d) = \begin{cases} (x-a)/(b-a) & \text{if } a \leq x < b \\ 1 & \text{if } b \leq x \leq c \\ (d-x)/(d-c) & \text{if } c < x \leq d \\ 0 & \text{if } x > d \text{ or } x < a \end{cases} \quad (3.2)$$

Gaussian Membership functions are defined using the following formula [38].

$$\mu_A(x) = \mu_A(x; m, \sigma) = \exp \left[- \left(\frac{(x-m)}{2\sigma} \right)^2 \right] \quad (3.3)$$

Greater resolution is achieved by using more linguistic terms or membership functions. Membership functions are designed to overlap with each other. By doing this, we are able to distribute our decisions over more than one input class, which helps to make FL systems robust [69]. Membership functions are generally scaled between zero and unity, to ensure that the variables are normalised [69]. The choice of membership functions generally depends on the user's preferences. However, we can use the partition theory put forward in [75] to determine the type of MFs to use, generally larger number of partitions indicates more degrees of freedom which might mean better performance.

3.2.3 Fuzzy Set Theoretic Operations

Fuzzy set theoretical operations, union, intersection and complement, are defined in terms of their membership functions [38] (depicted in Figure 3-4).

Let A and B be two subsets of X. The *union* of A and B, denoted by $A \cup B$, contains all the elements in either A or B, i.e. [38].

$$\mu_{A \cup B}(x) = \begin{cases} 1 & \text{if } x \in A \text{ or } x \in B \\ 0 & \text{if } x \notin A \text{ and } x \notin B \end{cases} \quad (3.4)$$

The *intersection* of A and B denoted $A \cap B$, contains all the elements that are simultaneously in A and B, i.e. [38].

$$\mu_{A \cap B}(x) = \begin{cases} 1 & \text{if } x \in A \text{ and } x \in B \\ 0 & \text{if } x \notin A \text{ or } x \notin B \end{cases} \quad (3.5)$$

Let \bar{A} denote the complement of A; it contains all the elements, not in A, i.e. [38],

$$\mu_{\bar{A}}(x) = \begin{cases} 1 & \text{if } x \notin A \\ 0 & \text{if } x \in A \end{cases} \quad (3.6)$$

From these facts, it is easy to show that [38]:

$$A \cup B \Rightarrow \mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)] \quad (3.7)$$

$$A \cap B \Rightarrow \mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)] \quad (3.8)$$

$$\bar{A} \Rightarrow \mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (3.9)$$

In fuzzy logic, union, intersection and complement are defined in terms of their membership functions. Let fuzzy sets A and B be described by their membership functions $\mu_A(x)$ and $\mu_B(x)$. One definition of fuzzy union leads to the membership function [38]:

$$\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)] \quad (3.10)$$

Moreover, one definition of the fuzzy intersection leads to the membership function [38]:

$$\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)] \quad (3.11)$$

Fuzzy intersection can also be defined as the product of the two membership function as [38]:

$$\mu_{A \cap B}(x) = \mu_A(x) \mu_B(x) \quad (3.12)$$

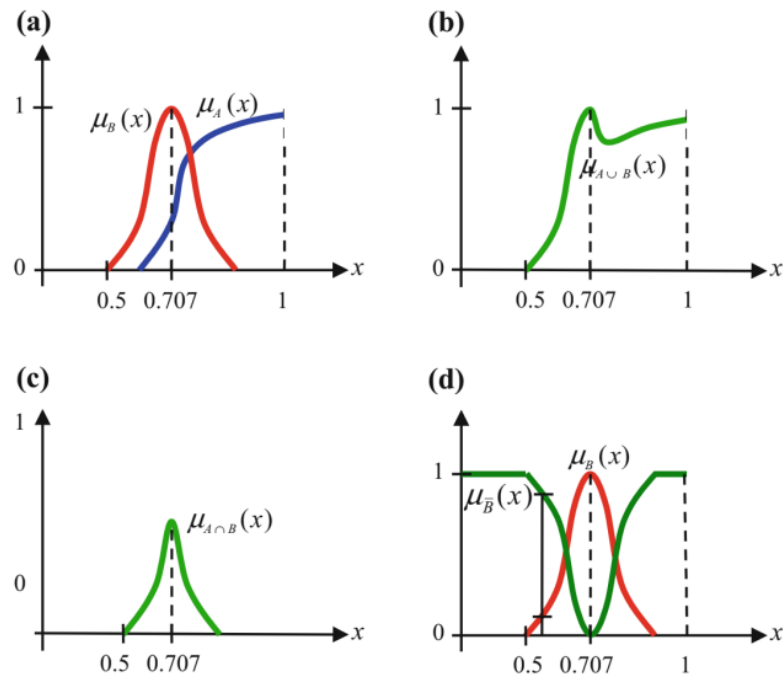


Figure 3-4: Fuzzy set theoretical operations a) Fuzzy sets A and B b) A union B c) A intersect B d) B complement [38]

Additionally, the membership function of the fuzzy compliment is [38]:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (3.13)$$

3.2.4 Rules

Fuzzy logic systems are rule-based systems. There are two main types of rule structures that are used in Fuzzy logic, one structure is attributed to Zadeh, and the other is attributed to Takagi and Sugeno [38]. In either case, the rules are expressed as a collection of IF-THEN statements. The IF-part of the rule is its *antecedent*, and the THEN part of a rule is its *consequent*.

Suppose a fuzzy logic system has p inputs $x_1 \in X_1, \dots, x_p \in X_p$ and one output $y \in Y$

. Let us suppose it has M rules, where the l^{th} Zadeh rule has the form [38]:

$$R^l : \text{IF } x_1 \text{ is } F_1^l \text{ and } \dots \text{ and } x_p \text{ is } F_p^l \text{ THEN } y \text{ is } G^l \quad l = 1, \dots, M \quad (3.14)$$

This rule represents a fuzzy relation between the input space X_1, \dots, X_p and the output space, Y of the fuzzy logic system [38].

It is possible to cast “nonobvious” rules into complete rules [38]. Six such rules are summarised in the below section

3.2.4.1 Incomplete IF rules

Suppose we have created a rule base where there are p inputs, where some of the rules have only a subset of m antecedents, e.g. [38].

$$R : IF x_1 is F_1 and , \dots, and x_m is F_m, THEN y is G \quad (3.15)$$

Such rules are called incomplete IF rules and apply regardless of x_{m+1}, \dots, x_p . They can be put into the format of the complete IF rule by treating the unnamed antecedents (e.g., x_{m+1}, \dots, x_p) as elements of the fuzzy set IN-COMPLETE (IN for short) where, by definition $\mu_{IN}(x)=1$ for all $x \in X$, i.e. [38].

$$\begin{aligned} & (IF x_1 is F_1 and , \dots, and x_m is F_m, THEN y is G) \\ \Leftrightarrow & (IF x_1 is F_1 and , \dots, and x_m is F_m and x_{m+1} is IN \dots and x_p is IN , THEN y is G) \end{aligned} \quad (3.16)$$

3.2.4.2 Mixed Rules

Not all rules use the “and” connective; some use the “or” connective, and some use a mixture of both. The latter rules are called *mixed rules*. These rules can be decomposed into a collection of equivalent rules, using standard techniques from crisp logic. Suppose, for example; we have the rule [38]:

$$IF(x_1 is F_1 and , \dots, and x_m is F_m) or (x_{m+1} is F_{m+1} \dots and x_p is F_p), THEN y is G \quad (3.17)$$

This rule can be expressed as the following two rules [38]:

$$\begin{aligned} R^1 &: \text{IF } x_1 \text{ is } F_1 \text{ and } \dots \text{ and } x_m \text{ is } F_m, \text{ THEN } y \text{ is } G \\ R^2 &: \text{IF } x_{m+1} \text{ is } F_{m+1} \text{ and } \dots \text{ and } x_p \text{ is } F_p, \text{ THEN } y \text{ is } G \end{aligned} \quad (3.18)$$

Observe that both rules are Incomplete IF rules.

3.2.4.3 Fuzzy Statement Rules

Some rules do not appear to have antecedents; they are statements involving fuzzy sets. Hence, they are called *fuzzy statement rules*. For example, *y is G* is such a rule. Clearly, this is an extreme case of an incomplete IF rule, and can therefore be formulated as [38]:

$$\text{IF } x_1 \text{ is } IN \text{ and } \dots \text{ and } x_p \text{ is } IN, \text{ THEN } y \text{ is } G \quad (3.19)$$

3.2.4.4 Comparative Rules

Some rules are comparative, e.g. [38]

The Smaller the x the bigger the y.

Such rules must first be formatted into IF-THEN rules; the preceding rule can be expressed as follows [38]:

IF x is S, THEN y is B.

Where S is a fuzzy set representing *smaller*, and B is a fuzzy set representing *bigger* [38].

3.2.4.5 *Unless Rules*

Rules are sometimes stated using the connective “unless”; such rules are called *unless rules* and can be put into the required format by using logical operators. For example, the rule [38]:

$$y \text{ is } G \text{ unless } x_1 \text{ is } F_1 \text{ and } \dots \text{ and } x_p \text{ is } F_p \quad (3.20)$$

can be expressed as [38]:

$$\text{IF not } (x_1 \text{ is } F_1 \text{ and } \dots \text{ and } x_p \text{ is } F_p), \text{ THEN } y \text{ is } G \quad (3.21)$$

3.2.4.6 *Quantifier Rules*

Rules sometimes include the quantifiers “some” or “all”; such rules are called quantifier rules. Because of the duality between propositional logic and set theory, rules with the quantifier “some” means that we have to apply the union operator to the antecedents or consequents to which the “some” applies, whereas rules with the quantifier “all” mean we have to apply the intersection operator to the antecedents or consequents to which the “all” applies [59].

3.2.5 *Fuzzifier*

The fuzzifier maps a crisp point $x = (x_1, \dots, x_n) \in X$ into a fuzzy set A_x in U . There are two types of fuzzifiers: Singleton and Non-singleton [38].

A Singleton fuzzifier is one in which $\mu_{X_i}(x'_i) = 1$ and $\mu_{X_i}(x_i) = 0$ for $x_i \in X_i$ and $x_i \neq x'_i$ [38]. Singleton fuzzifier is the most commonly used fuzzifier.

A non-singleton fuzzification is one in which measurement $x_i = x'_i$ is mapped into a fuzzy number [38]. For which $\mu_{X_i}(x'_i) = 1$ and $\mu_{X_i}(x_i)$ decreases from unity as x_i

moves away from x'_i [38]. Non-singleton fuzzifiers are used in instances where the inputs are noisy or uncertain.

Conceptually, the non-singleton fuzzifier implies that the given input value x'_i is the most likely value to be the correct one from all the values in its immediate neighbourhood; however, because the uncertainty in the input due to noise, neighbouring points are also likely to be the correct, but to a lesser degree [38]. Figure 3-5 illustrates singleton and non-singleton fuzzification.

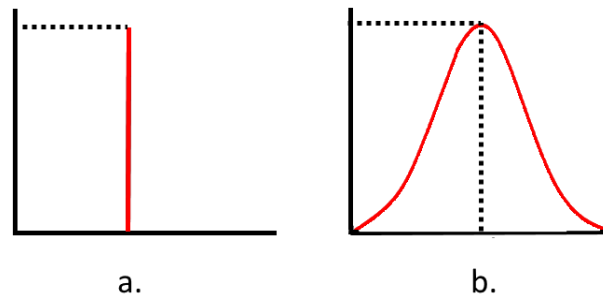


Figure 3-5: a) Singleton Fuzzification b) Non-singleton Fuzzification [76]

3.2.6 Fuzzy Inference Engine

In the fuzzy inference engine (which is labelled inference engine in Figure 3-1), fuzzy logic principles are used to interpret fuzzy IF-THEN rules into a mapping between the fuzzy input sets in $X_1 \times \dots \times X_p$ to fuzzy output sets in Y . Each rule is interpreted as a fuzzy implication (Mamdani implication [77] is used here). We treat the fuzzy inference engine as a system, one that maps fuzzy set into fuzzy sets [38].

When Zadeh rules and Mamdani fuzzy system is used, then the fuzzy inference can be expressed as follows [38]:

$$\mu_{A' \rightarrow G'}(x, y) = [T_{i=1}^p \mu_{F'_i}(x_i)] * \mu_{G'}(y) \quad (3.22)$$

In (3.22), $T_{i=1}^p \mu_{F_i^l}(x_i)$ is called the firing strength of the rule. There are three widely used connectives to calculate firing strength. If all connectives in a rule are “And” then the minimum (3.23) or the product membership degrees can be used (3.24) (t-norm) [38]:

$$\mu_{A \rightarrow B}(x) \equiv \min[\mu_A(x), \mu_B(x)] \quad (3.23)$$

$$\mu_{A \rightarrow B}(x) = \mu_A(x)\mu_B(x) \quad (3.24)$$

If all the rule connectives are “Or” then the maximum membership degree can be used (3.25) (t-conorm) [38]:

$$\mu_{A \rightarrow B}(x) \equiv \max[\mu_A(x), \mu_B(x)] \quad (3.25)$$

3.2.7 Defuzzifier

Defuzzification produces a crisp output for FLS from the fuzzy sets that are the outputs of the inference engine [38]. One criterion we use for the choice of a defuzzifier is its computational simplicity. Since we use population-based optimization to optimise the FLS, the FLS is called frequently during the optimisation process. This means that complex defuzzification will lead to much more computational resources being used. Some defuzzification methods are as follows:

3.2.7.1 Centroid Defuzzifier

The centroid defuzzifier combines the output fuzzy sets using union (i.e. a t-conorm, e.g. maximum) and then find the centroid of this set [38]. i.e.,

$$y_c(x') = \frac{\sum_{i=1}^N y_i \mu_B(y_i | x')}{\sum_{i=1}^N \mu_B(y_i | x')} \quad (3.26)$$

Unfortunately, the centroid defuzzification is usually difficult and time-consuming to compute as we have to first compute the union of the fuzzy sets.

3.2.7.2 *Height Defuzzifier*

The height defuzzifier [78], also called the centre average defuzzifier, replaces each rule output fuzzy set with a singleton having maximum membership in the rule's consequent membership set, then calculating the centroid of the type-1 set comprised of these singletons. The output of a height defuzzifier is given as [38]:

$$y_h(x') = \frac{\sum_{i=1}^N \bar{y}^i \mu_{B^i}(\bar{y}^i | x')}{\sum_{i=1}^N \mu_{B^i}(\bar{y}^i | x')} \quad (3.27)$$

This is very easy to use because the centres of gravity of commonly used membership functions are known ahead of time. For example, regardless of whether minimum or product inference is used, the centre of gravity of B^l for [69]:

- A symmetric triangular consequent membership function is at the apex of the triangle.
- A Gaussian consequent membership function is at the centre value of the Gaussian function.
- A symmetric trapezoidal membership function is at the midpoint of its support.

3.2.7.3 *Centre-Of-Sets Defuzzifier*

In centre-of-sets defuzzification [79], each rule consequent is replaced set by a singleton located at its centroid, whose amplitude equals the firing level, and then the centroid of the type-1 set comprised of these singletons. The expression of the output of the Centre-of-sets defuzzifier is given as [38]:

$$y_{cos}(x') = \frac{\sum_{l=1}^M COG(G^l) f^l(x')}{\sum_{l=1}^M f^l(x')} = \frac{\sum_{l=1}^M c^l f^l(x')}{\sum_{l=1}^M f^l(x')} \quad (3.28)$$

3.3 Type-2 Fuzzy Logic Systems

Type-1 fuzzy logic systems have limited capabilities in modelling and minimizing uncertainties in the data [69]. As discussed, uncertainty comes in many guises and is independent of the kind of fuzzy system or methodology one uses to handle it. Two important aspects of uncertainties are linguistic and random. The former is associated with words, and the fact that *words can mean different things to different people*, and the latter is associated with unpredictability. Probability theory is used to handle random uncertainty, and fuzzy systems are used to handle linguistic uncertainty, and sometimes FLSs can also be used to handle both kinds of uncertainty, because a fuzzy system may use noisy measurements or operate under random disturbances [69].

Adding uncertainty to the type-1 membership functions means that the membership grade is no longer a crisp number; it is its own set in the range [0, 1]. Calculating all $x \in X$ creates a three-dimensional membership function, a type-2 membership function that characterises a type-2 fuzzy set.

3.3.1 General Type-2 Fuzzy Sets

Consider the transition from ordinary sets to fuzzy sets. When we cannot determine the membership of an element in a set as 0 or 1, we use fuzzy sets of type 1. Similarly, When the circumstances are so fuzzy, we have trouble determining the membership grade even as a crisp number [0,1] we use fuzzy sets of type-2, a concept that was first introduced by Zadeh in 1975 [38].

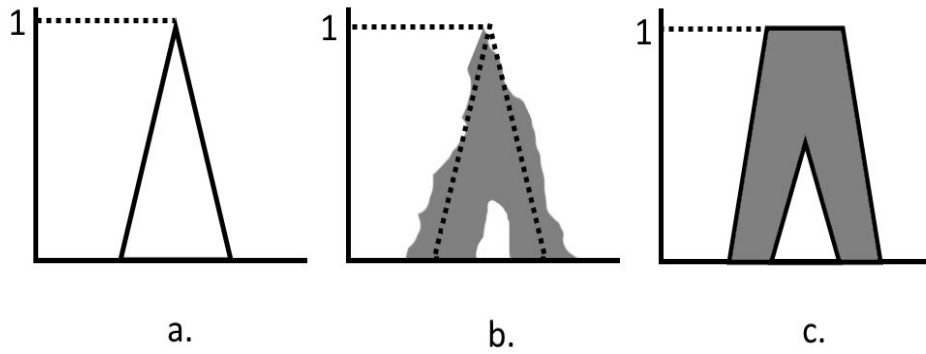


Figure 3-6: a) Type-1 Membership Function b) Blurred Type-1 Membership Function c) Footprint of Uncertainty [38]

A type-2 set can also be described as a type-1 fuzzy set where the degree of membership is fuzzy. Figure 3-6 *a.* shows a type-1 membership function, where the membership is blurred by shifting the triangle on the x-axis, but not necessarily by the same amount, this would generate Figure 3-6 *b.* This means that the degree of membership is no longer a single value but is represented by a fuzzy set; whose degree of membership takes on values wherever the vertical line intersects the blurs.

Calculating all $x \in X$ creates a three-dimensional membership function, a type-2 membership function that characterises a type-2 fuzzy set [38].

A type-2 fuzzy set denoted \tilde{F} , is characterised by a type-2 membership function $\mu_{\tilde{F}}(x, u)$, where $x \in X$ and $u \in J_x \subseteq [0, 1]$, i.e., [80]

$$\tilde{F} = \{(x, u), \mu_{\tilde{F}}(x, u) \mid \forall x \in X, \forall u \in J_x \subseteq [0, 1]\} \quad (3.29)$$

In which $\mu_{\tilde{F}}(x, u) \in [0, 1]$. \tilde{F} can also be expressed as [38, 80]

$$\tilde{F} = \int_{x \in X} \int_{u \in J_x} \mu_{\tilde{F}}(x, u) / (x, u), \quad J_x \subseteq [0, 1] \quad (3.30)$$

Where \coprod denotes union over all admissible x and u [38]. An example of a general type-2 fuzzy set is depicted in Figure 3-7 (a) and (b). J_x is called the primary membership of x in \tilde{F} . At each value of x say $x = x'$, the two-dimensional (2-D) plane, whose axes are u and $\mu_{\tilde{F}}(x', u)$, is called a vertical slice of \tilde{F} . It is $\mu_{\tilde{F}}(x = x', u)$, for $x' \in X$ and $\forall u \in J_{x'} \subseteq [0, 1]$, i.e., [38, 80]

$$\mu_{\tilde{F}}(x = x', u) \equiv \mu_{\tilde{F}}(x') = \int_{u \in J_{x'}} f_{x'}(u) / u \quad J_{x'} \subseteq [0, 1] \quad (3.31)$$

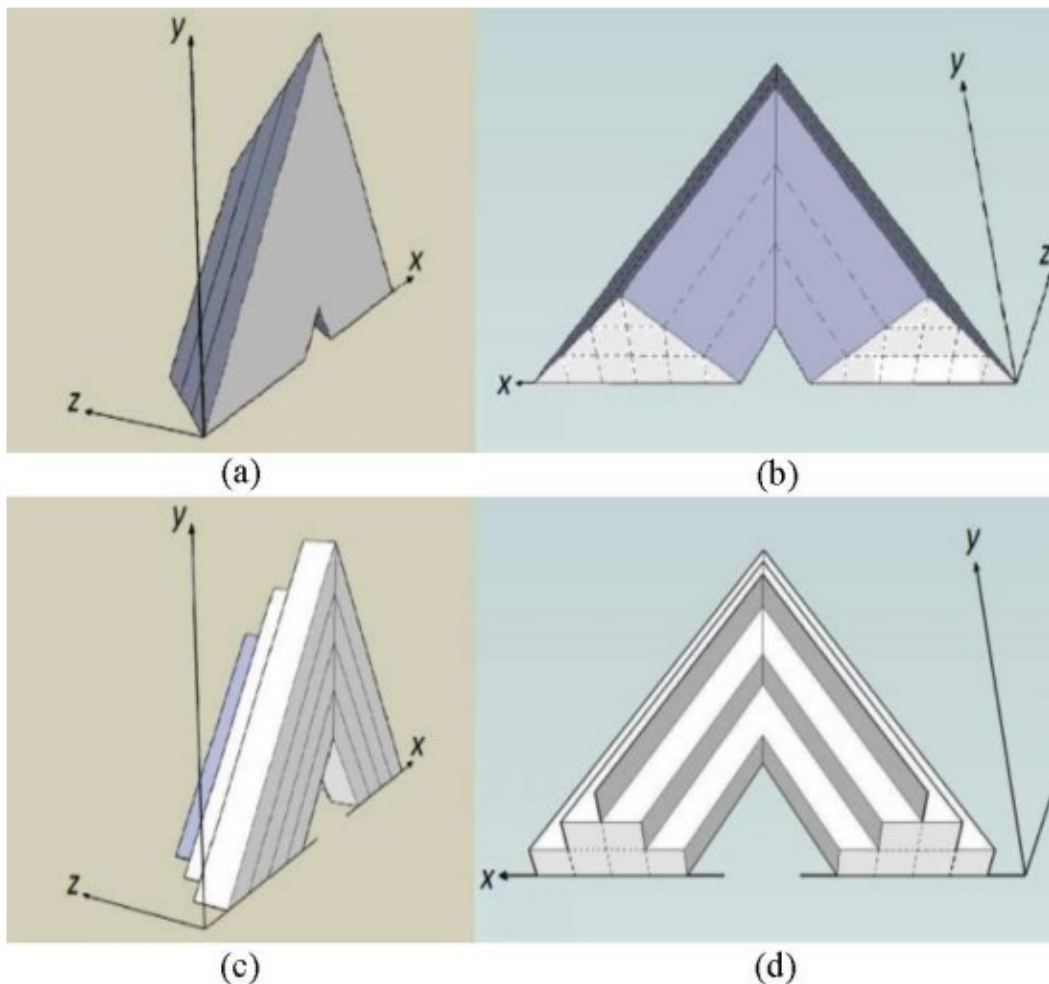


Figure 3-7: (a) Side view of a general type-2 fuzzy set, with three zLevels on the third dimension (b) Tilted rear/below view of the same set, indicating the position of the three zSlices. (c) Side view of the same zSlices version in (a), with $I=3$. (d) Tilted rear/below view of the same set, showing the zSlices [80].

Where $0 \leq f_x(u) \leq 1$, and $\mu_{\tilde{F}}(x')$ is referred to as a secondary MF[80].

3.3.2 z-slices Based General Type-2 Fuzzy Sets

Recently there have been several methods that have been proposed to limit the complexity of general type-2 fuzzy logic such as an alpha plane based representation presented in [81] or geometric representation presented in [82]. One of these methods, a zSlices based representation [80], is presented in this section.

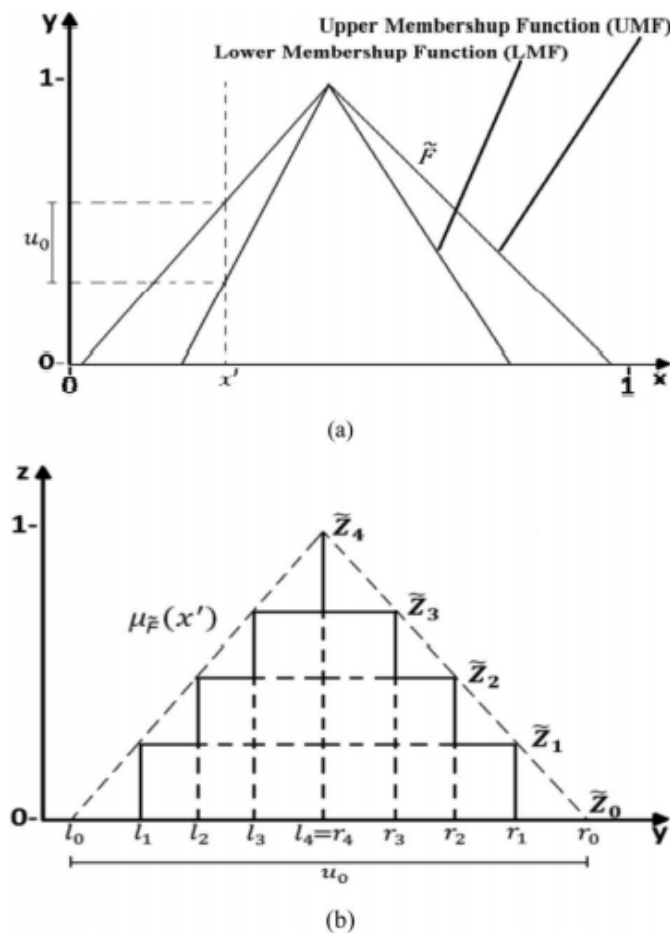


Figure 3-8: (a) Front view of a general type-2 set \tilde{F} . (b) Third dimension at x' of a zSlices-based type-2 fuzzy set with $I=4$ [80]

A zSlice is formed by slicing a general type-2 fuzzy set in the third dimension (z) at level z_i . This slicing action will result in an interval set in the third dimension with

height z_i . As such, a zSlice \tilde{Z}_i is a fuzzy set with its membership grade $\mu_{\tilde{Z}_i(x,u)}$ in the third dimension equal to z_i , where $0 \leq z_i \leq 1$. Thus, the zSlice \tilde{Z}_i can be written as follows [80]:

$$\tilde{Z}_i = \int_{x \in X} \int_{u_i \in [l_i, r_i]} z_i / (x, u_i) \quad (3.32)$$

Where, at each x value (depicted in Figure 3-8 (a)), zSlicing creates an interval set with height z_i and domain J_{ix} , which ranges from l_i to r_i , as shown in Figure 3-8 (b), $1 \leq i \leq I$, where I is the number of zSlices (excluding \tilde{Z}_0) and $z_i = i / I$ [80].

A general type-2 fuzzy set \tilde{F} can be seen as equivalent to an infinite collection of zSlices. In a discrete universe of discourse \tilde{F} can be represented as follows [80]:

$$\tilde{F} = \sum_{i=0}^I \tilde{Z}_i \quad (3.33)$$

The summation sign in (3.33) does not represent the arithmetic addition but denotes the set theoretical operation union. If the maximum operation is used to represent the union operation, whenever a u value is attached to more than one z_i values, the maximum z_i is chosen and attached to the given u value. Hence, the $\mu_{\tilde{F}}(x')$ at x' of the zSlices-based general type-2 fuzzy set \tilde{F} shown in Figure 3-8 (b) can be expressed as follows [80]:

$$\begin{aligned} \mu_{\tilde{F}}(x') &= \sum_{i=0}^I \sum_{u_i \in [l_i, r_i]} z_i / u_i \\ &= \sum_{u_i \in J_x^i} \max(z_i) / u, J_{x'} \subseteq [0, 1] \end{aligned} \quad (3.34)$$

Where $0 \leq i \leq I$. It is worth noting that at x' , $\mu_{\tilde{F}}(x')$ is a type-1 fuzzy set.

3.3.3 Interval Type-2 Fuzzy Sets

There are many possible choices for secondary membership functions. When $f_x(u) = 1 \ \forall u \in J_x \subseteq [0,1]$ in equation (3.31), then the secondary membership functions are interval sets, and, if it is true for $\forall x \in X$, we have an interval type-2 membership function which characterizes an interval type-2 fuzzy set (IT2FS) [83]. An IT2 FS is said to be maximally uncertain because all its secondary membership grades are the same value. A general Type-2 fuzzy set is said to be less uncertain than an IT2 FS because its secondary grades are not all the same.

Since all the secondary memberships of an interval type-2 fuzzy set are unity, an interval type-2 fuzzy set is represented by its domain interval, which can be represented by its left and right end-points $[l,r]$ [83]. The two endpoints are associated with two type-1 membership functions that are referred to as upper and lower membership functions which are bound by a footprint of uncertainty FOU [83]. The upper membership function and lower membership function are represented as $\bar{\mu}_{\tilde{F}}(x)$ and $\underline{\mu}_{\tilde{F}}(x)$ respectively.

It has been argued that using type-2 fuzzy sets to represent the inputs and outputs of FLS has many advantages when compared to the type-1 fuzzy sets; some of these advantages are as follows [84]:

- As the type-2 fuzzy sets membership functions are fuzzy and contain a footprint of uncertainty, then they can model and handle the linguistic and numerical uncertainties associated with the inputs and outputs of the FLS. Therefore, FLSs

that are based on type-2 fuzzy sets will have the potential to produce a better performance than the type-1 FLCs when dealing with uncertainties [83].

- Using type-2 fuzzy sets to represent the FLS inputs and outputs will result in the reduction of the FLS rule base when compared to using type-1 fuzzy sets, as the uncertainty represented in the footprint of uncertainty in type-2 fuzzy sets lets us cover the same range as type-1 fuzzy sets with a smaller number of labels and the rule reduction will be greater when the number of the FLS inputs increases [38].
- Each input and output will be represented by a large number of type-1 fuzzy sets, which are embedded in the type-2 fuzzy sets [38]. The use of such a large number of type-1 fuzzy sets to describe the input and output variables allows for a detailed description of the analytical control surface as the addition of the extra levels of classification give a much smoother control surface and response. In addition, according to Karnik and Mendel [85], the type-2 FLS can be thought of as a collection of many different embedded type-1 FLSs.
- It has been shown in [86] that the extra degrees of freedom provided by the footprint of uncertainty enables a type-2 FLS to produce outputs that cannot be achieved by type-1 FLSs with the same number of membership functions. It has been shown that a type-2 fuzzy set may give rise to an equivalent type-1 membership grade that is negative or larger than unity. Thus, a type-2 FLS can model more complex input-output relationships than its type-1 counterpart and, thus, can give better control response.

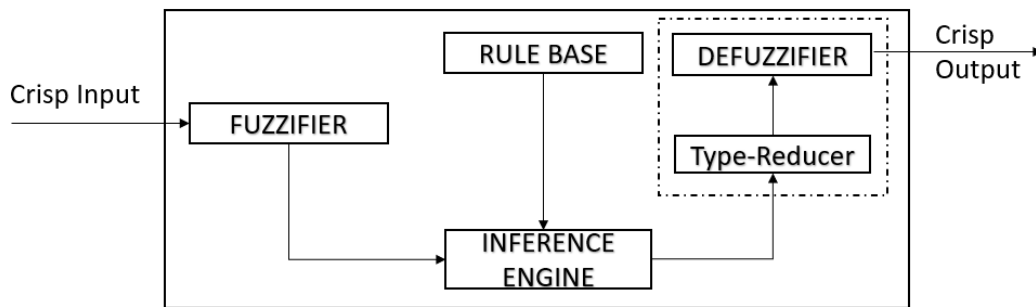


Figure 3-9: Type -2 Fuzzy Logic System

3.3.4 Interval Type-2 Fuzzy Logic Systems

A Typical IT2FLS is depicted in Figure 3-9; it contains five components: fuzzifier, rule base, inference engine, type-Reducer and a defuzzifier. The main difference between an IT2FLS and a type-1 FLS is that the output fuzzy sets of the inference engine have to be type-Reducer to output a crisp value. Furthermore, it employs IT2 fuzzy sets in the input and output of the FLS [38].

The IT2FLS works in the following way: the crisp inputs in the data are first fuzzified into an input interval type-2 fuzzy set; singleton fuzzification is commonly used in IT2FLS as it simplifies the computation. Once the inputs are fuzzified, the inference engine then activates the rule base using the input interval type-2 fuzzy sets and produces the output interval type-2 fuzzy sets. There is no difference between the rule base of a T1FLS and a type-2 FLS except that the fuzzy sets are interval type-2 fuzzy sets instead of type-1 fuzzy sets. In the final step, the output interval type-2 sets produced in the previous steps are converted into a crisp number. There are two methods for doing this; the first method is the conventional two-step process where the output type-2 sets are converted into type-reduced type-1 fuzzy sets followed by defuzzification of the type reduced sets. The second method is the direct defuzzification

process which was introduced because of the computational complexity of the first method [38].

3.3.5 Type-Reduction + Defuzzification

In engineering applications, it is desirable to get the output of a fuzzy logic system as a crisp number. In type-1 FLS this is achieved by defuzzification of the output type-1 fuzzy sets. For Interval type-2 fuzzy logic system, there are two main methods for achieving this. (a) Map the IT2 fuzzy set directly to a number also called direct defuzzification. (b) Map the IT2 fuzzy sets into type-1 fuzzy sets and then map them into a number, Type-Reduction + Defuzzification [38]. In this section, we describe one the method that can be used for Type-Reduction + Defuzzification.

Type-reduction + defuzzification must reduce to defuzzification when the uncertainties of a T2 FS disappear. Consequently, the type-reduction methods are extensions of type-1 defuzzification methods, and, once a type-reduction method is applied to IT2 FS, then it is relatively easy to apply the same to general type-2 fuzzy sets. The starting point for type-reduction is for IT2 FSs, for which each weighted average is a special kind of interval weighted average [38].

Given

$$x_i \in X_i \equiv [\underline{x}_i, \bar{x}_i], \quad i = 1, 2, \dots, N \quad (3.35)$$

$$w_i \in W_i \equiv [\underline{w}_i, \bar{w}_i], \quad i = 1, 2, \dots, N \quad (3.36)$$

Where $\underline{x}_i \leq \bar{x}_i$ and $\underline{w}_i \leq \bar{w}_i$, and X_i and W_i represent sets of intervals. We can compute the interval weighted average as follows.

$$Y_{IWA} = \frac{\sum_{i=1}^n X_i W_i}{\sum_{i=1}^n W_i} = [y_l, y_r] \quad (3.37)$$

Where Y_{IWA} is completely defined by its two endpoints, y_l and y_r . Because $X_i (i=1, \dots, n)$ appear only in the numerator, the smallest and largest value of each X_i is used to find y_l (y_r), i.e., [38]:

$$y_l = \min_{\forall w_i \in [c_i, d_i]} \frac{\sum_{i=1}^n a_i w_i}{\sum_{i=1}^n w_i} \quad (3.38)$$

$$y_r = \max_{\forall w_i \in [c_i, d_i]} \frac{\sum_{i=1}^n b_i w_i}{\sum_{i=1}^n w_i} \quad (3.39)$$

Where the notations under min and max in (3.38) and (3.39) mean that i ranges from 1 to n and each w_i ranges from c_i and d_i [38].

It is well known that the equations in (3.38) and (3.39) can be rewritten as follows [38].

$$y_l = \frac{\sum_{i=1}^L a_i d_i + \sum_{i=L+1}^n a_i c_i}{\sum_{i=1}^L d_i + \sum_{i=L+1}^n c_i} \quad (3.40)$$

$$y_r = \frac{\sum_{i=1}^R b_i c_i + \sum_{i=R+1}^n b_i d_i}{\sum_{i=1}^R c_i + \sum_{i=R+1}^n d_i} \quad (3.41)$$

Where L and R are switch points. There is no closed form solution for L and R , hence to calculate y_l and y_r , iterative algorithms such as Karnik Mendel(KM) algorithm are used.

3.3.5.1 Karnik-Mendel Algorithm

The KM Algorithm for computing y_l is as follows[38]:

Step 1: Initialize w_i by setting $w_i = (c_i + d_i)/2, i=1,2,\dots,n$

$$y' = y(w_1, \dots, w_n) = \frac{\sum_{i=1}^n a_i w_i}{\sum_{i=1}^n w_i} \quad (3.42)$$

Step 2: Find $k \in \{1, 2, \dots, n-1\}$ such that $a_k \leq y' \leq a_{k+1}$

Step 3: Set $w_i = d_i$ when $i \leq k$, and $w_i = c_i$ when $i \geq k+1$, and then compute

$$y_l(k) = \frac{\sum_{i=1}^k a_i d_i + \sum_{i=k+1}^n a_i c_i}{\sum_{i=1}^k d_i + \sum_{i=k+1}^n c_i} \quad (3.43)$$

Step 4: Check if $y_l(k) = y'$. If yes, stop and set $y_l(k) = y_l(L)$ and call k L . If no, go to Step 5.

Step 5: Set $y' = y_l(k)$ and go to Step 2.

The KM Algorithm for calculating y_r is as follows [38]:

Step 1: Initialize w_i by setting $w_i = (c_i + d_i)/2, i=1,2,\dots,n$

$$y' = y(w_1, \dots, w_n) = \frac{\sum_{i=1}^n b_i w_i}{\sum_{i=1}^n w_i} \quad (3.44)$$

Step 2: Find $k \in \{1, 2, \dots, n-1\}$ such that $b_k \leq y' \leq b_{k+1}$

Step 3: Set $w_i = c_i$ when $i \leq k$, and $w_i = d_i$ when $i \geq k+1$, and then compute

$$y_r(k) = \frac{\sum_{i=1}^k b_i c_i + \sum_{i=k+1}^n b_i d_i}{\sum_{i=1}^k c_i + \sum_{i=k+1}^n d_i} \quad (3.45)$$

Step 4: Check if $y_r(k) = y'$. If yes, stop and set $y_r(k) = y_r(R)$ and call k R . If no, go to Step 5.

Step 5: Set $y' = y_r(k)$ and go to Step 2.

3.3.5.2 Centre of Sets Type Reduction

The Centre-of-Sets type reduction [39] is described in this section. Regardless of which type-reduction method is used, the type-reduced set is also an interval set and has the following structure [38]:

$$Y_{TR} = [y_l, y_r] \quad (3.46)$$

Recall that the COS defuzzifier for a type-1 fuzzy system replaces each rule consequent T1 FS by a singleton located at its centroid, with an amplitude equal to the firing level.

Similarly, in the COS type-reducer each rule consequent IT2 FS, \tilde{G}^i , by the support of its centroid, $[c_l(\tilde{G}^i), c_r(\tilde{G}^i)]$, and assigns a secondary MF of $1/[\underline{f}^i(x'), \bar{f}^i(x')]$ to it, where $[\underline{f}^i(x'), \bar{f}^i(x')]$ are the firing intervals for the i^{th} rule [38]. Hence, from (3.40), (3.41) and (3.28) COS type-reduction is as follows [38]:

$$y_l^{COS}(x') = \frac{\sum_{i=1}^L c_l(\tilde{G}^i) \bar{f}^i(x') + \sum_{i=L+1}^M c_l(\tilde{G}^i) \underline{f}^i(x')}{\sum_{i=1}^L \bar{f}^i(x') + \sum_{i=L+1}^M \underline{f}^i(x')} \quad (3.47)$$

$$y_r^{COS}(x') = \frac{\sum_{i=1}^R c_r(\tilde{G}^i) \bar{f}^i(x') + \sum_{i=R+1}^M c_r(\tilde{G}^i) \underline{f}^i(x')}{\sum_{i=1}^R \bar{f}^i(x') + \sum_{i=R+1}^M \underline{f}^i(x')} \quad (3.48)$$

Where $c_l(\tilde{G}^i)$ and $c_r(\tilde{G}^i)$ are the left and right endpoints of the centroid of the IT2 consequent, \bar{f}^i and \underline{f}^i are the upper and lower firing levels of the i^{th} rule [38], and the switch points L and R can be calculated using the KM algorithm (described in section 3.3.5.1).

Please note that to compute the $y_l^{COS}(x')$ and $y_r^{COS}(x')$ one must first compute the centroid of each rule's IT2 consequent set, $[c_l(\tilde{G}^i), c_r(\tilde{G}^i)]$. Which can be done by using the below equations [38]:

$$c_l(\tilde{G}) = \frac{\sum_{i=1}^L x_i \bar{\mu}_{\tilde{G}}(x_i) + \sum_{i=L+1}^n x_i \underline{\mu}_{\tilde{G}}(x_i)}{\sum_{i=1}^L \bar{\mu}_{\tilde{G}}(x_i) + \sum_{i=L+1}^n \underline{\mu}_{\tilde{G}}(x_i)} \quad (3.49)$$

$$c_r(\tilde{G}) = \frac{\sum_{i=1}^R x_i \underline{\mu}_{\tilde{G}}(x_i) + \sum_{i=R+1}^n x_i \bar{\mu}_{\tilde{G}}(x_i)}{\sum_{i=1}^R \underline{\mu}_{\tilde{G}}(x_i) + \sum_{i=R+1}^n \bar{\mu}_{\tilde{G}}(x_i)} \quad (3.50)$$

Where, $\underline{\mu}_{\tilde{G}}(x_i)$ and $\bar{\mu}_{\tilde{G}}(x_i)$ are the upper and lower membership function of the IT2 consequent set at x_i , and, the switch points L and R can be calculated using the KM algorithm (described in section 3.3.5.1). The centroids only need to be calculated once as these centroids do not depend on the input to the fuzzy system.

Next, the intervals calculated using (3.47) and (3.48) are averaged to get the crisp output.

$$Y_{COS} = \frac{y_l(x') + y_r(x')}{2} \quad (3.51)$$

3.3.6 Direct Defuzzification

The main motivation for direct defuzzification is to avoid the iterative computation needed to perform the type-reduction. Another motivation is to obtain a formula for crisp output of the IT2FLS that can be used in some sort of mathematical analysis [38].

Nie Tan method [87], is formulated using the vertical slice representation of an IT2 FLS from (3.31). For an IT2 FLS, each vertical slice is an embedded type-1 fuzzy set that can be easily type reduced. The intuition behind the Nie Tan method is that the computational overhead of type reduction can be reduced by first type reducing individual vertical slices, before defuzzifying the resulting type-1 fuzzy set to obtain the centroid of the IT2 FLS [87].

Consider an IT2 FLS that is represented by a collection of vertical slices [87], i.e.,

$$\tilde{F}^j = \int_{u \in J_x} 1 / \mu(x_j) / x_j \quad (3.52)$$

where j denotes the j^{th} vertical slice in equation (3.30).

Suppose the continuous vertical slice is discretized into n_j points, then the centroid of each vertical slice can be computed as follows [87]:

$$\mu(x_j) = \frac{1}{n_j} \sum_{i=1}^{n_j} \mu^i(x_j) \quad (3.53)$$

For IT2 FS, the average of a vertical slice that comprises n_j discrete points is the mean of the upper and lower membership functions [87]:

$$\mu(x_j) = \frac{1}{2} (\underline{\mu}(x_j) + \bar{\mu}(x_j)) \quad (3.54)$$

Similarly, the combined fired-output set \tilde{B} can be defuzzified by computing the centre of gravity of the average of its lower and upper MFs [38], i.e.,

$$y_{NT}(x') = COG \left\{ \frac{1}{2} [\underline{\mu}_{\tilde{B}}(y|x') + \bar{\mu}_{\tilde{B}}(y|x')] \right\} = \frac{\sum_{i=1}^N y_i [\underline{\mu}_{\tilde{B}}(y|x') + \bar{\mu}_{\tilde{B}}(y|x')]}{\sum_{i=1}^N [\underline{\mu}_{\tilde{B}}(y|x') + \bar{\mu}_{\tilde{B}}(y|x')]} \quad (3.55)$$

The procedure for computing the crisp output of the IT2FLS using the Nie Tan method is as follows [87].

1. Interval type-2 fuzzy consequent sets are reduced to the crisp output using equation (3.55).
2. The firing set for each rule is then obtained using the meet operation as per normal.
3. An IT2 FS representing the output of the fuzzy inference engine is constructed using the extended sub start composition. Then the crisp output is calculated using the equation (3.55).

Mendel and Liu [88] prove that $y_{NT}(x')$ (3.55) is a first-order Taylor series approximation of the actual defuzzified value of \tilde{B} when using the centroid type - reduction + defuzzification [38]. This means that the Nie Tan direct defuzzification method gives a close approximation of the centroid type-reduction + defuzzification. In addition, we can see that this direct defuzzification depends on only the upper and lower firing levels and so we do not have to perform complex iterative computations. In addition, when all IT2 MF uncertainties disappear in (3.55), it reduces to the correct type-1 defuzzification formula. Hence, in this thesis, we use Nie-tan defuzzification for all the regression datasets.

3.4 Fuzzy Rule-Based Classification Systems

Classification problems involve assigning an input vector x_1, \dots, x_n to a class C_h from a predefined set of classes $C = C_1, \dots, C_M$. In fuzzy rule-based classification systems (FRBCS) these inputs are mapped to the output using rules. The fuzzy rule in the FRBCS can be written as follows [89].

$$\text{Rule } R^j : \text{If } x_1 \text{ is } A_1^j \text{ and } \dots \text{ and } x_n \text{ is } A_n^j \text{ then } y \text{ is Class } C_h \text{ with } CF_h \quad (3.56)$$

Where x_1, \dots, x_n represents the input vector, A_n^j represents the linguistic label for the antecedent pattern i and CF_h is the rule weight.

3.4.1 Scaled Support and Scaled Confidence

In FRBCS, where the rules are generated from data and not from expert knowledge, many of the rules of the FRBCS system will have the same antecedents but the conflicting consequents [89], to resolve this conflict, two measures are generally used.

- Confidence: It is the conditional probability that, if the antecedents are true, then the consequent is true [90].
- Support: It is a measure of how general a given rule is and can be considered to be the probability of occurrence of records with given antecedents and consequent in a particular data set [90].

The confidence and support measure can cause problems in skewed datasets, i.e., in datasets, where the number of training records for one class vastly outnumber the number of training records for the other classes. To overcome these problems, Scaled Confidence and Scaled Support [91] are used in this thesis.

To calculate the scaled confidence and support, we first need to calculate the scaled firing strength of the rules. The scaled firing strength is calculated by dividing the firing strength of a rule by the sum of firing strength of all the rules that have different antecedents but the same consequent. The upper and lower scaled firing strengths are calculated using the following formulas.

$$\overline{fs}^{jt} = \frac{\overline{f}^{jt}}{\sum_{j \in Class_h} \overline{f}^{jt}} \quad (3.57)$$

$$\underline{fs}^{jt} = \frac{\underline{f}^{jt}}{\sum_{j \in Class_h} \underline{f}^{jt}} \quad (3.58)$$

Where \overline{f}^{jt} and \underline{f}^{jt} represent the upper and lower firing levels of the rules.

$\sum_{j \in Class_h} \overline{f}^{jt}$ and $\sum_{j \in Class_h} \underline{f}^{jt}$ represent the sum of firing strength of all the rules that have

$Class_h$ as their consequent.

The scaled confidence ($A_q \rightarrow C_q$) for rules where the class C_q is the consequent class for the antecedents \tilde{A}_q (where there are m conflicting rules with the same antecedents and conflicting consequents) could be written as follows [89]:

$$\bar{c}(\tilde{A}_q \rightarrow C_q) = \frac{\sum_{x_s \in Class C_h} \overline{fs}^{jt}(x_s)}{\sum_{j=1}^m \overline{fs}^{jt}(x_s)} \quad (3.59)$$

$$\underline{c}(\tilde{A}_q \rightarrow C_q) = \frac{\sum_{x_s \in Class C_h} \underline{fs}^{jt}(x_s)}{\sum_{j=1}^m \underline{fs}^{jt}(x_s)} \quad (3.60)$$

Here The scaled confidence can be viewed as measuring the validity of $(\tilde{A}_q \rightarrow C_q)$ [89].

The scaled support for the same can be written as follows [89].

$$\bar{s}(\tilde{A}_q \rightarrow C_q) = \frac{\sum_{x_s \in \text{Class } C_h} \overline{f_s^{j_t}}(x_s)}{m} \quad (3.61)$$

$$\underline{s}(\tilde{A}_q \rightarrow C_q) = \frac{\sum_{x_s \in \text{Class } C_h} \underline{f_s^{j_t}}(x_s)}{m} \quad (3.62)$$

Here the support can be viewed as measuring the coverage of training patterns by $(\tilde{A}_q \rightarrow C_q)$ [89].

These two values can then be used to calculate the scaled dominance or the weights of the rules using the below formulas [89].

$$\bar{d}(A_q \rightarrow C_q) = \bar{c}(\tilde{A}_q \rightarrow C_q) \cdot \bar{s}(\tilde{A}_q \rightarrow C_q) \quad (3.63)$$

$$\underline{d}(A_q \rightarrow C_q) = \underline{c}(\tilde{A}_q \rightarrow C_q) \cdot \underline{s}(\tilde{A}_q \rightarrow C_q) \quad (3.64)$$

3.4.2 Calculate Output Class

To determine the output of the FRBCS, the relative importance of each of the possible output classes has to be calculated, and the class with the highest importance score is the output class for an input vector.

To calculate the importance of each output consequent or class. The upper and lower importance score for each of the output classes is calculated using the below formulae [89].

$$\bar{Z}class_h(x^p) = \frac{\sum_{j \in h} \overline{f^j(x^p)} * \bar{d}(A_q \rightarrow C_q)}{\text{Max}_{j \in h} (\overline{f^j(x^p)} * \bar{d}(A_q \rightarrow C_q))} \quad (3.65)$$

$$\underline{Z}class_h(x^p) = \frac{\sum_{j \in h} \underline{f^j(x^p)} * \underline{d}(A_q \rightarrow C_q)}{\text{Max}_{j \in h} (\underline{f^j(x^p)} * \underline{d}(A_q \rightarrow C_q))} \quad (3.66)$$

Where $\overline{f^j(x^p)}$ and $\underline{f^j(x^p)}$ represent the upper and lower firing levels for the rule j for input (x^p) , $\text{Max}_{j \in h} (\overline{f^j(x^p)} * \bar{d}(A_q \rightarrow C_q))$ and $\text{Max}_{j \in h} (\underline{f^j(x^p)} * \underline{d}(A_q \rightarrow C_q))$ represent the maximum product of the upper or lower firing level and the upper or lower weight for the rules.

Finally, the upper and lower importance scores of the output classes are averaged using (3.67) to get the class importance score.

$$Zclass_h(x^p) = \frac{\bar{Z}class_h(x^p) + \underline{Z}class_h(x^p)}{2} \quad (3.67)$$

The class with the highest $Zclass_h$ will be the class predicted for the input vector (x^p) .

3.4.3 Similarity Metric

In case the incoming input vector (x^p) does not trigger any of the existing rules, we need to decide the output class for the input. The first step in this process is to build all possible from the given input vector, using the matched fuzzy sets [92].

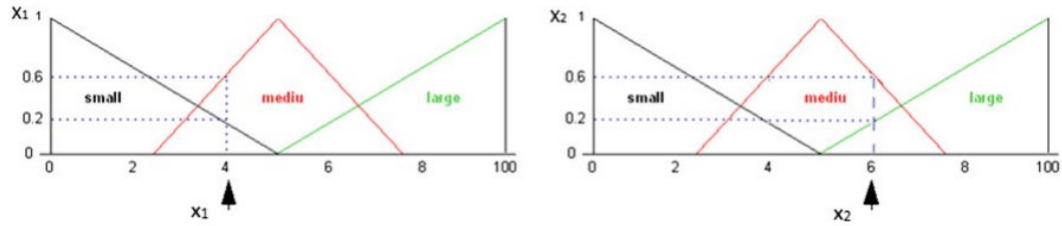


Figure 3-10: An Example to Illustrate Similarity Metric

Suppose we have a classification problem with two inputs x_1 and x_2 , and that the input vector (x^p) will match the fuzzy sets as shown in Figure 3-10. Then from Figure 3-10, we can generate four different rules: $R_1 = \{Small, Medium\}$, $R_2 = \{Small, Large\}$, $R_3 = \{Medium, Medium\}$ and $R_4 = \{Medium, Large\}$. Each rule will have an associated firing strength but no consequent.

To find the consequent of these rules, we need to find the rule that is closest to each of the generated rules. In order to do this, we calculate a similarity measure between each of the generated rules and each of the X rules stored in the rule base. Let the linguistic labels that fit the input vector (x^p) be written as $v_{inputr} = (v_{input1r}, v_{input2r}, \dots, v_{inputnr})$ where r is the index of the r^{th} generated rule. Let the linguistic label corresponding to any given rule in the rule base be $v_j = (v_{j1}, v_{j2}, \dots, v_{jn})$. Then the similarity of the rule can be calculated using the below [89]:

$$\begin{aligned}
 Similarity_{inputr \leftrightarrow j} &= \left(1 - \left| \frac{v_{input1r} - v_{j1}}{V_1} \right| \right) \\
 &* \left(1 - \left| \frac{v_{input2r} - v_{j2}}{V_2} \right| \right) * \dots * \left(1 - \left| \frac{v_{inputnr} - v_{jn}}{V_n} \right| \right)
 \end{aligned} \tag{3.68}$$

Where V_1, \dots, V_n represents the number of linguistic labels representing each variable. Each rule in the rule base will have a similarity value associated with each of the r generated rules. For each of the generated rule, this similarity value is used to choose the consequent or output class [89].

To predict the output class. The upper and lower importance score for each of the output classes is calculated using the below formulae [89].

$$\bar{Z}class_h(x^p) = \frac{\sum_{j \in h} \overline{f^j}(x^p) * \bar{d}(A_q \rightarrow C_q)}{\text{Max}_{j \in h} (\overline{f^j}(x^p) * \bar{d}(A_q \rightarrow C_q))} \quad (3.69)$$

$$\underline{Z}class_h(x^p) = \frac{\sum_{j \in h} \underline{f^j}(x^p) * \underline{d}(A_q \rightarrow C_q)}{\text{Max}_{j \in h} (\underline{f^j}(x^p) * \underline{d}(A_q \rightarrow C_q))} \quad (3.70)$$

Where $\overline{f^j}(x^p)$ and $\underline{f^j}(x^p)$ represent the upper and lower firing levels of the most similar rule in the rule base and $\bar{d}(A_q \rightarrow C_q)$ and $\underline{d}(A_q \rightarrow C_q)$ represent the upper and lower scaled dominance of the most similar rule in the rule base.

Finally, the upper and lower importance score is used to calculate the total importance score of the class.

$$Zclass_{hr} = \frac{\bar{Z}class_h(x^p) + \underline{Z}class_h(x^p)}{2} \quad (3.71)$$

The class with the highest $Zclass_{hr}$ will be the class associated with the input vector (x^p) .

3.5 Design Methods for Fuzzy Logic Systems

This section describes two design methods that can be used to create fuzzy logic systems using examples.

3.5.1 Wang Mendel Method

Wang and Mendel developed a well-known method for developing fuzzy systems from examples, combining both expert knowledge and numerical data examples [93]. They proposed a four-step generalised method for constructing fuzzy systems by generating rules from examples. This method leads to a fuzzy system with the number of rules smaller than the number of training data pairs, and it is widely used as it is simple to implement [38].

Given a set of desired input-output data pairs [38]:

$$(x_1^{(1)}, x_2^{(1)} : y^{(1)}), (x_1^{(2)}, x_2^{(2)} : y^{(2)}), \dots, (x_1^{(n)}, x_2^{(n)} : y^{(n)}) \quad (3.72)$$

Where x_1 and x_2 are inputs and y is an output. This simple two input-one output example is used to explain the intricacies of the Wang Mendel method; extending this to a multi input-multi output system is straightforward. The Wang and Mendel approach consists of the following five steps [93]:

Step 1 – Divide the input and output spaces into fuzzy regions.

Let the domain intervals $x_1, x_2 : y_1$ be represented by $[x_1^-, x_1^+]$, $[x_2^-, x_2^+]$, and $[y^-, y^+]$ respectively, where “domain interval” of a variable means that the value of the variable most probably be present in this interval (the value of the variable can lie outside the interval). Divide each domain interval into $2N + 1$ regions (N can be different for

different variables, and the lengths of these regions can be equal or unequal), denoted by SN (Small N), ..., S1 (Small 1), CE (Centre), B1 (Big 1), ..., BN (Big N), and assign each region a fuzzy membership function [93].

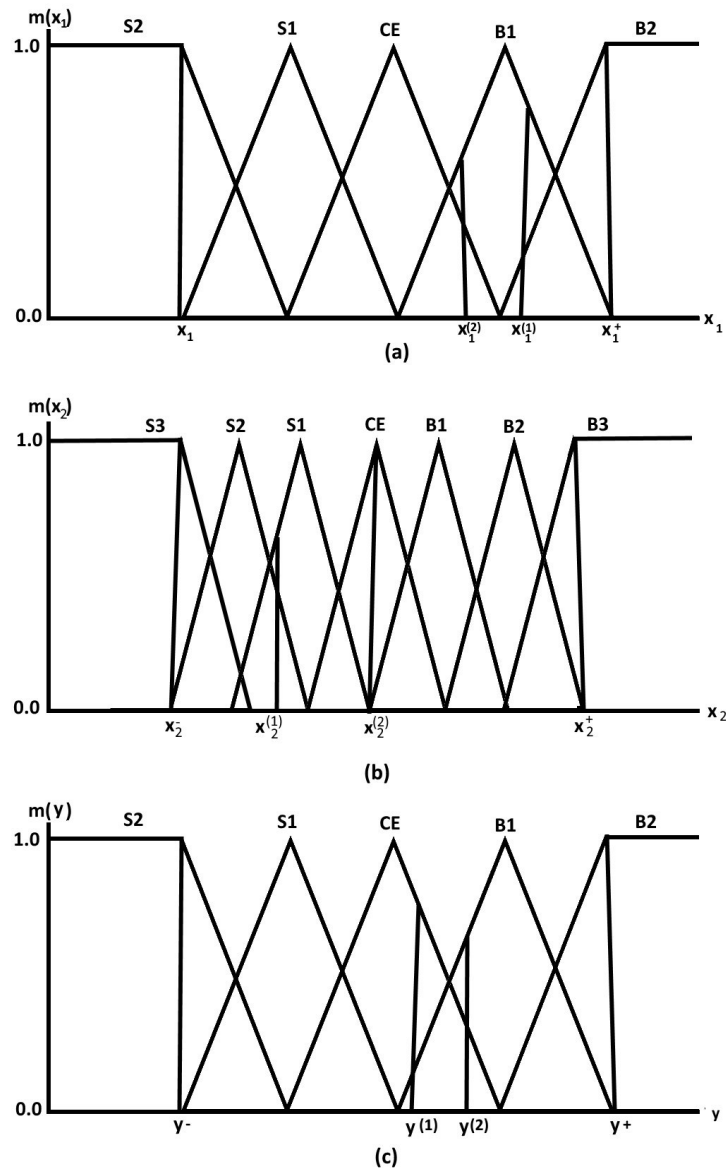


Figure 3-11: Division of Domain Intervals [93]

Figure 3-11 shows an example where the domain interval for the variable x_1 is divided into five regions ($N = 2$), the domain region for the variable x_2 is divided into seven regions ($N = 3$), and the domain interval of the output y is divided into five

regions ($N = 2$). The shape of each membership function in this example is triangular; with one vertex at the centre of the region that has a degree of membership 1; the other two vertices lie at the centres of the two neighbouring regions, respectively, and have a degree of membership zero. Of course, other divisions and other shapes of membership functions are possible [93].

Step 2 – Generate fuzzy rules from given data pairs.

First, determine the degree of membership for the given input $x_1^{(1)}, x_2^{(1)} : y^{(1)}$ in different regions. For example, $x_1^{(1)}$ in Figure 3-11, has degree 0.8 in B1, degree 0.2 in B2 and zero degrees in all other regions. Similarly, $x_2^{(1)}$ in Figure 3-11, has degree 1 in CE, and zero degrees in all other regions [93].

Second, assign a given $x_1^{(1)}, x_2^{(1)}$ and $y^{(1)}$ to the region with the maximum degree of membership. For example, $x_1^{(1)}$ in Figure 3-11, is assigned to be B1, and $x_2^{(1)}$ is assigned to be CE [93].

Finally, we obtain one rule from the one pair of desired input-output data, e.g. [93].

$$(x_1^{(1)}, x_2^{(1)}; y^{(1)}) \Rightarrow [x_1^{(1)}(0.8 \text{ in } B1, \max), x_2^{(1)}(0.7 \text{ in } S1, \max); y^{(1)}(0.9 \text{ in } CE, \max)] \quad (3.73)$$

IF x_1 is B1 and x_2 is S1, THEN y is CE [93].

$$(x_1^{(2)}, x_2^{(2)}; y^{(2)}) \Rightarrow [x_1^{(2)}(0.6 \text{ in } B1, \max), x_2^{(2)}(1 \text{ in } CE, \max); y^{(2)}(0.7 \text{ in } B1, \max)] \quad (3.74)$$

IF x_1 is B1 and x_2 is CE, THEN y is CE [93].

The rules generated in this way are “and” rules, i.e., rules in which conditions of the IF part must be met simultaneously for the result of the THEN part to occur [93].

Step 3 – Assign a degree to each rule

After generating the rules using the method in Step 2, it is highly likely that there will be some conflicting rules, i.e., rules that have the same antecedents but a different consequent. One way to resolve this conflict is to assign a degree to each rule generated from data pairs and pick the rule with the maximum degree from this group. In this way, not only is the conflict resolved, but the number of rules is also greatly reduced [93].

We use the following product strategy to assign a degree to each rule: for the rule: “IF x_1 is A and x_2 is B, THEN y is C” the degree of this rule, denoted by $D(\text{Rule})$, is defined as [93]:

$$D(\text{Rule}) = m_A(x_1)m_B(x_2)m_C(y) \quad (3.75)$$

As examples, Rule 1 has degree [93]

$$D(\text{Rule1}) = m_{B1}(x_1)m_{S1}(x_2)m_{CE}(y) \quad (3.76)$$

$$= 0.8 \times 0.7 \times 0.9 = 0.504$$

(see Figure 3-11) and Rule 2 has degree [93]

$$D(\text{Rule2}) = m_{B1}(x_1)m_{CE}(x_2)m_{B1}(y) \quad (3.77)$$

$$= 0.6 \times 1.0 \times 0.7 = 0.42$$

In practice, we often have some prior information about the data pair. For example, if we let an expert check given data pairs, the expert may suggest that some are very useful and crucial, but others are very unlikely and may be caused just by measurement errors. Therefore, we can assign a degree to each data pair that represents our belief of its usefulness. In this sense, the data pairs constitute a fuzzy set, i.e. the fuzzy set is defined as the useful measurements; a data pair belongs to this set to a degree assigned by a human expert [93].

Suppose the data pair $(x_1^{(1)}, x_2^{(1)} : y^{(1)})$ has degree $m^{(1)}$, then we redefine the degree of Rule 1 as [93].

$$D(\text{Rule1}) = m_{B1}(x_1) m_{S1}(x_2) m_{CE}(y) m^{(1)} \quad (3.78)$$

i.e., the degree of a rule is defined as the product of the degrees of its components and the degree of the data pair that generates this rule. This is important in practical applications because real numerical data have different reliabilities, e.g., some real data can be very bad (“wild data”). For good data, we assign higher degrees, and for bad data, we assign lower degrees. In this way, human experience about the data is used in a common base as other information. If one emphasises objectivity and does not want a human to judge the numerical data, our strategy still works by setting all the degrees of the data pairs equal to unity [93].

Step 4 – Create a combined fuzzy rule base

The form of a fuzzy rule base is illustrated in Figure 3-12. We will fill the boxes of the base with fuzzy rules according to the following strategy: a combined fuzzy rule base is assigned rules from either those generated from the numerical or linguistic rules

B3					
B2					
B1					
x_2 CE					
S1					
S2					
S3					
	S2	S1	CE	B1	B2
	x_1				

Figure 3-12: The form of a Fuzzy Rule Base [93].

(we assume that a linguistic rule also has a degree that is assigned by the human expert and reflect the experts belief of the importance of the rule); if there is more than one rule in one box of the fuzzy rule base, use the rule that has maximum membership degree.

In this way, both numerical and linguistic information is codified into a common framework – the combine fuzzy rule base. If a linguistic rule is an “and” rule, it fills only one box of the fuzzy rule base; but, if a linguistic rule is an “or” rule (i.e., a rule for which the THEN part follows if any condition of the IF part is satisfied), it fills all the boxes in the rows or columns corresponding to the regions of the IF part. For example, suppose we have the linguistic rule:

“IF x_1 is S1 or x_2 is CE, THEN y is B2” for the fuzzy rule base of Figure 3-12; then we fill the seven boxes in the column of S1 and the five boxes in the row of CE

with B2. The degree of all the B2's in these boxes equal the degree of this “or” rule [93].

Step 5 – Determine a mapping based on the combined fuzzy rule base.

We use the following defuzzification strategy to determine the output control y for given inputs (x_1, x_2) : first, for given inputs (x_1, x_2) , we combine the antecedents of the i^{th} fuzzy rule using product operation to determine the degree, m_{O^i} , of the output control corresponding to (x_1, x_2) , i.e. [93],

$$m_{O^i}^i = m_{I_1^i}(x_1)m_{I_2^i}(x_2) \quad (3.79)$$

Where O^i denotes the output region of Rule i and I_j^i denotes the input region of Rule i for the j^{th} component, e.g., Rule 1 gives [93].

$$m_{CE}^1 = m_{B1}(x_1)m_{S1}(x_2) \quad (3.80)$$

Then we use the following centroid defuzzification formula to determine the output [93].

$$y = \frac{\sum_{i=1}^K m_{O^i}^i \bar{y}^i}{\sum_{i=1}^K m_{O^i}^i} \quad (3.81)$$

Where \bar{y}^i denotes the centre value of region O^i (the centre of a fuzzy region is defined as the point that has the smallest absolute value among all the points which the membership function for this region has membership value equal to one), and K is the number of fuzzy rules in the combined fuzzy rule base [93].

3.5.2 Enhanced Wang-Mendel Method

Wang [94] introduced three rule extraction methods that are variations of the WM method described in the previous section. The variations are used to solve different problems for different purposes [38].

The first method is used to extract specific rules which target a specific region and rules with different resolutions with flexible choices of the MFs [38]. Specifically, given m input variables $(x_{i_1}, \dots, x_{i_m})$ selected from (x_1, \dots, x_n) fall into the fuzzy region characterized by “IF x_{i_1} is A_{i_1} and, ..., and x_{i_m} is A_{i_m} ” where the membership functions for the fuzzy sets A_{i_1}, \dots, A_{i_m} are given, the problem is to determine the fuzzy set B in the fuzzy IF-THEN rule [94]:

$$\text{IF } x_{i_1} \text{ is } A_{i_1} \text{ and } \dots \text{ and } x_{i_m} \text{ is } A_{i_m}, \text{ THEN } y \text{ is } B \quad (3.82)$$

Step 1: For each input-output pair $(x^p; y^p)$, $p = 1, 2, \dots, N$, compute.

$$w^p = \prod_{j=1}^m \mu_{A_{i_j}}(x_{i_j}^p) \quad (3.83)$$

Where A_{i_j} are the fuzzy sets in (3.82). If $\sum_{p=1}^N w^p = 0$, then no rule in the form (3.82) will be generated; the method stops. Otherwise, view w^p as a weight of y^p and compute the weighted average

$$av = \frac{\sum_{p=1}^N y^p w^p}{\sum_{p=1}^N w^p} \quad (3.84)$$

Step 2: A fuzzy IF-THEN rule in the form of (3.82) is generated with B determined according to the following two cases.

Case 1: Among the K fuzzy sets B^1, \dots, B^k defined in the output space R , find the B^{j^*} such that

$$\mu_{B^{j^*}}(av) \geq \mu_{B^j}(av) \quad (3.85)$$

For $j = 1, 2, \dots, K$. The B is chosen as B^{j^*} .

Case 2: Compute the weighted “variance”

$$\sigma = \frac{\sum_{p=1}^N |y^p - av| w^p}{\sum_{p=1}^N w^p} \quad (3.86)$$

And the fuzzy set B is the fuzzy set with triangular membership functions

$$\mu_B(y) = \mu(y; av, \sigma) \quad (3.87)$$

Step 3: Compute the degree of confidence(doc) of the rule generated in Step 2.

Specifically, in Case 1 the doc is defined as

$$doc = \left(1 - \frac{\sigma}{\max_{p,q=1}^N |y^p - y^q|} \right) \mu_{B^{j^*}}(av) \quad (3.88)$$

And in Case 2 it is defined as

$$doc = 1 - \frac{\sigma}{\max_{p,q=1}^N |y^p - y^q|} \quad (3.89)$$

The second method can be used to extrapolate the rules over regions not covered by the data. The details of the method are detailed below [94].

Step 1: For each extrapolating rule, determine how many neighbours it has from the set of data-generated rules. Find the group of extrapolating rules that have the maximum number of such neighbours and call this group the max-group.

Step 2: For each rule in the max-group, compute

$$y_c^l = \frac{\sum_{r=1}^b y_c^r doc^r dis^r}{\sum_{r=1}^b doc^r dis^r} \quad (3.90)$$

Where l is the index of the rule, b is the number of neighbours this rule has from the data-generated rules, y_c^r and doc^r is the distance between the input centre of this rules and its neighbour rule r . The extrapolating rules in this max group are generated as

$$IF x_{i1} \text{ is } A_{i1}^l \text{ and } \dots \text{ and } x_{im} \text{ is } A_{im}^l, THEN y \text{ is } B^l \quad (3.91)$$

Where A_{ij}^l are fuzzy sets corresponding to the entry of the extrapolating rule in the rule table, and B^l is a triangular fuzzy set $\mu_{B^l}(y) = \mu(y; y_c^l, \sigma^l)$ with y_c^l given in (3.90) and σ^l computed as

$$\sigma^l = \frac{\sum_{r=1}^b |y_c^r - y_c^l| doc^r}{\sum_{r=1}^b doc^r} \quad (3.92)$$

Step 3: Compute the degree of confidence of the extrapolating rule (3.91) as

$$doc^l = 1 - \frac{\sigma^l}{\max_{k,t=1}^b |y_c^k - y_c^t|} \quad (3.93)$$

Where k,t run over all the neighbour rules are the r in (3.90) and (3.92).

Step 4: View the extrapolating rules generated in Step 2 as the same of the data-generated rules and go to Step 1 to repeat the process until all the extrapolating rules are generated. The extrapolating rules plus the data-generated rules form a complete fuzzy rule set.

3.6 Summary

This chapter provided an overview of the Fuzzy Logic Systems explaining the different stages from the crisp input, to fuzzification, rule inference, the various defuzzification methods and the crisp output.

This chapter described the Type-1 and Type-2 Fuzzy sets and presented the many advantages of using type-2 fuzzy sets to represent the inputs and outputs of the FLS. It went on to describe the difference between the type-1 and type-2 FLS and explained the additional steps required, such as type-reduction. It then described the Fuzzy rule-based classification systems built using scaled confidence and support, that are required in cases where the classification data is skewed.

It also explained the similarity metric that allows the FLS to generate output in no rules fired situations that might be caused because of the short rules and small rule-bases that are used to preserve the interpretability of the FLS.

The next chapter gives an overview of the optimisation algorithms

Chapter 4. An Overview of Selected Optimization Algorithms

Optimisation is a process by which a large number of feasible solutions to a problem are systematically examined until we find the best solution. An objective or fitness function is used to determine the goodness or fitness of the solutions, and objectives are often cost, distance, time and other such factors.

We use stochastic search principles (where randomness is introduced into the algorithms so that a given input can produce different outputs depending on the system variables) to all the optimization algorithms to find globally optimal solutions more reliably [69].

4.1 Big Bang Big Crunch (BB-BC)

Big Bang Big Crunch method [95] is based on the idea that randomness can be seen as energy dissipating into the environment while convergence to an optimal point can be seen as gravitational attraction. These ideas are used as the basis for BB-BC algorithm where randomness is used to create totally new candidate solutions by creating disorder and convergence is used to pick the optimal solutions from this disorder.

BB-BC is similar to a Genetic Algorithm [95] in that both of these methods use randomness to generate their initial population. The creation of the initial population is referred to as the Big Bang Phase where the candidate solutions are spread uniformly across the solution space (depicted in Figure 4-1)

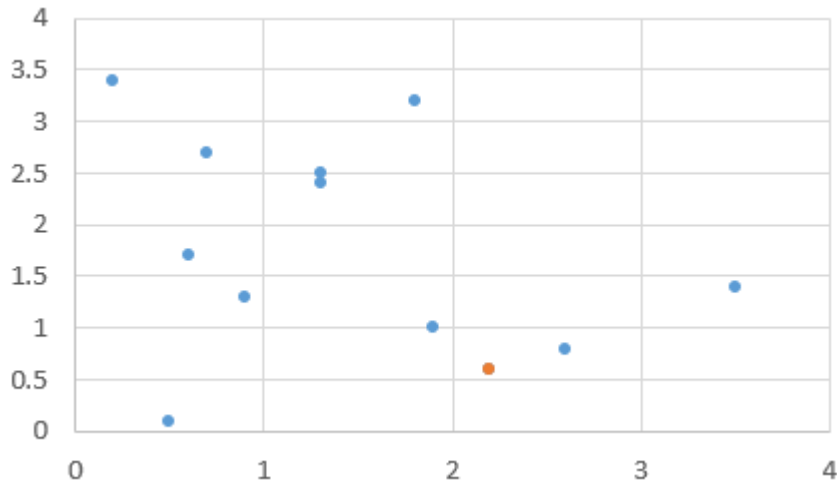


Figure 4-1: 2D depiction of the initial population of candidate solutions in BB-BC algorithm

The population size is generally kept fixed throughout the optimization process. However, the population size can be increased or decreased depending on the fitness of the solutions or the number of iterations.

The Big bang phase is followed by a Big Crunch phase. The Big Crunch phase is represented by a convergence operator which reduces the number of solutions to one, which is termed as the centre of mass. The centre of mass can be calculated using the following formula.

$$x_c = \frac{\sum_{i=1}^n \frac{1}{f_i} x_i}{\sum_{i=1}^n \frac{1}{f_i}} \quad (4.1)$$

Where x_i represents the individual candidate solution, f_i is the fitness of the i^{th} solution and n is the population size of the Big Bang Phase.

After the Big Crunch phase, the algorithm will create a new generation of candidates to be used in the next iteration. There are various ways to achieve this; the simplest way to do this is to generate a new generation randomly. This method has no benefits

as the knowledge gained in the previous step is lost. The BB-BC algorithm retains the knowledge from each generation by generating the new generation of candidate solutions around the optimal point or centre of mass that was discovered in the previous step. This is achieved by generating the new candidates using a normal distribution operation around the centre of mass and also by reducing the standard deviation of the normal distribution function as the number iterations increases [95] (depicted in Figure 4-2)

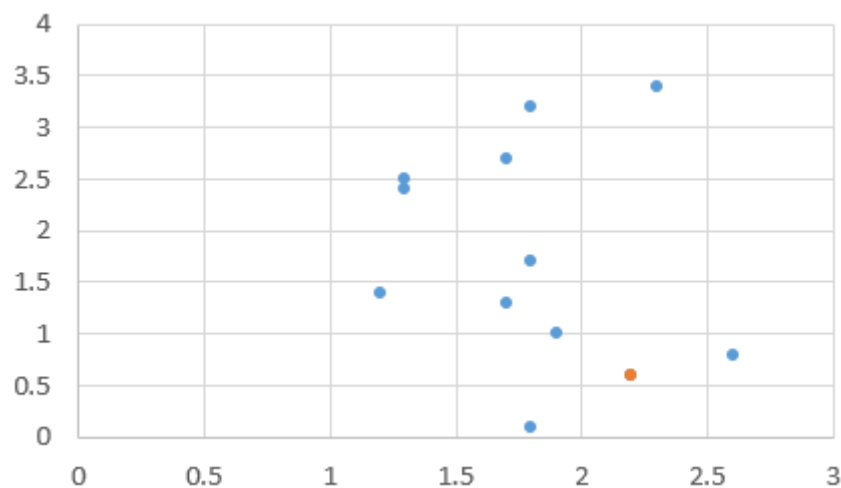


Figure 4-2: 2D depiction of the candidate solutions in the second Big Bang Phase in BB-BC algorithm

After the second Big Bang phase, the centre of mass is recalculated using the new candidate solutions. This process of, Big Bang followed by Big Crunch, is repeated until the stopping criteria are met. The knowledge gained in each Big Crunch phase is transmitted to the next Big Bang Phase in the form of the centre of mass and the standard deviation of the candidate solutions.

As the number of iterations increases the standard deviation becomes smaller and smaller, eventually reaching zero as the number of iterations reaches infinity. Thus, the new generation candidates who will be located far from the centre of mass will be

selected with decreasing probability thus potentially moving the centre of mass towards itself if it has higher fitness value than the remaining candidate solutions [95]. Thus ensuring the global convergence property of the BB-BC algorithm [95].

A summary of the BB-BC algorithm is presented below [95]:

- The initial generation of n candidates is randomly generated while respecting the limits of the search space.
- Calculate the fitness of all the candidate solutions generated in the previous step.
- Find the centre of mass using equation (4.1) or the best-fit individual among the candidate solutions is chosen.
- Calculate the new generation of candidate solutions around the centre of mass chosen in the previous step using the below equation.

$$x_{new} = x_c + \frac{lr}{k} \quad (4.2)$$

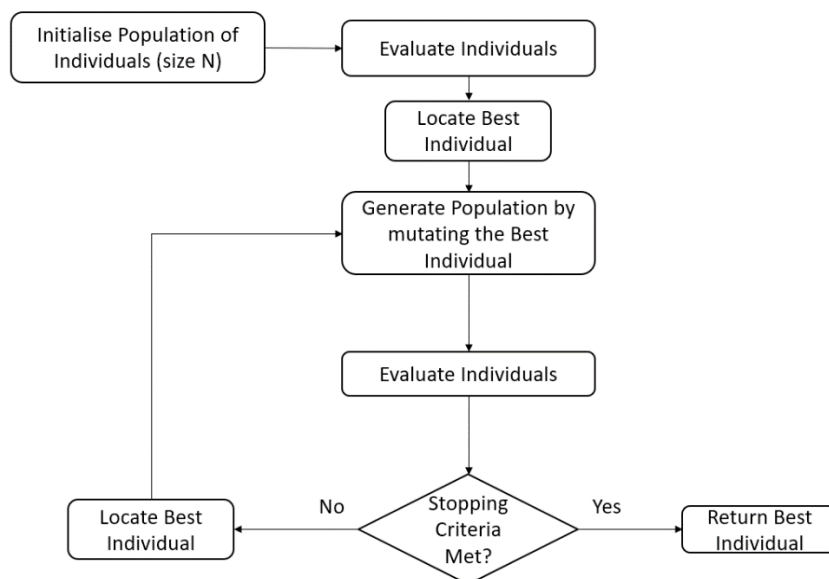


Figure 4-3: Flow Chart for the Big Bang Big Crunch Algorithm

where x_c is the centre of mass, l is the upper limit of the search space, r is the random number generated and k is the number of iterations completed.

- Return to step 2 if the stopping criteria are not met.

4.1.1 Implementation of the Big Bang-Big Crunch Algorithm

The flow chart for the BB-BC algorithm is show in Figure 4-3, and the pseudocode is presented in Figure 4-4.

```

k = 1 // Iteration count
Loop (until stopping criteria) {
  For i = 1 to number of individuals
    For j = 1 to dimensions
       $X_{id} = P_{(i-1)d} + lr/k$  //r is a random number, l is upper limit
    Next j
    If (Evaluate( $X_i$ ) > Evaluate( $P_{(i-1)}$ )) {
       $P_i = X_i$  //Pi is best in current iteration
    }
  Next i
   $P_{(i-1)} = P_i$  //P(i-1) best overall
  k = k + 1
}

```

Figure 4-4: Pseudocode for the BB-BC algorithm

4.2 Genetic Algorithms

Genetic Algorithms are based on the theory of evolution, i.e., the population of individuals evolves over time to adapt to a given environment. This means that individuals that are more suitable to the environment are more likely to survive and reproduce [96].

Genetic Algorithm is an iterative procedure that is implemented in the following way:

An initial population of individuals or chromosomes is generated either randomly or by using some method to embed prior knowledge into the population. Included prior knowledge is known to improve the rate of convergence of the genetic algorithm.

A fitness function is used to evaluate or determine the fitness of all the individuals in the population. The fitness function is designed based on the objectives of the optimisation process.

Then a new generation of individuals is generated by first selecting certain individuals from the population (usually depends on the fitness of the individual), and then the next generation of individuals are generated. The next generation is then evaluated for fitness, and if the stopping criteria are met, we stop the evaluation. Else, these individuals are used to produce the next generation of offspring's. This process is repeated until the stopping criteria are met.

4.2.1 Genetic Operators

4.2.1.1 Selection

In genetic algorithms, selecting the individual or chromosome that will produce the next generation of individuals is crucial. The selection process is generally proportional to the fitness of the individuals. There are many methods that are commonly used to accomplish this, Fitness proportional selection and Tournament selection are the most popular methods [69, 97, 98]

- **Fitness Proportional Selection:** In this method, the probability that a particular individual i is selected depends on the following equation

$$p_i = \frac{f_i}{\sum_{i=1}^n f_i} \quad (4.3)$$

Where f represents the fitness of a particular individual. Therefore the probability of an individual being selected is proportional to their fitness [21]. Hence, the selection process involves randomly generating a number and selecting individuals based on their fitness proportion and its correlation to the random number.

- **Tournament Selection:** The most common way in which tournament selection takes place is by randomly selecting k individuals from the population, and the fittest individual from these k individuals is selected [21].

4.2.1.2 Crossover

Crossover is the process in which the chromosomes of two individuals are merged by cutting the chromosomes at some chosen points [21]. The most common crossover techniques are below.

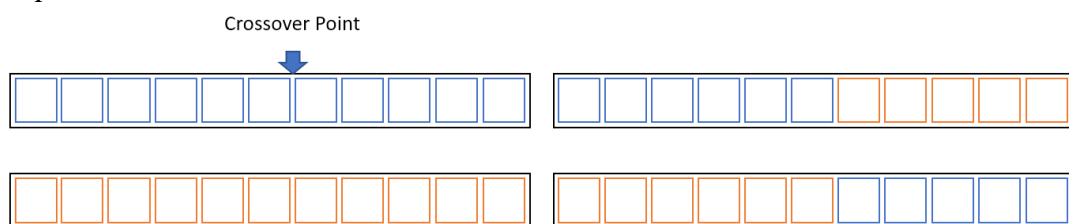


Figure 4-5: Single Point Crossover [69]

- **Single Point Crossover:** In this method, a single point is chosen for the crossover point. The chromosome after the chosen point is swapped between the two parent chromosomes to generate the new individuals. The single point crossover is illustrated in Figure 4-5.

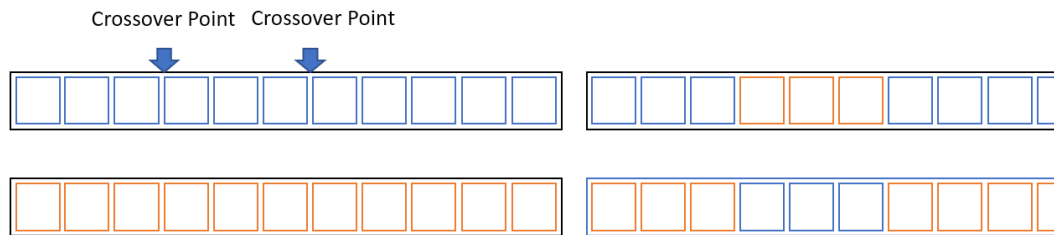


Figure 4-6: Multi-Point Crossover [69]

- **Multi-Point Crossover:** In this method, n points are chosen for the crossover points. The sections of the chromosomes between the n chosen points are swapped between the chromosomes of the two parents to create the new individuals. This process is illustrated in Figure 4-6.

4.2.1.3 Mutation

Mutation allows undirected jumps within the search space [69]. Mutation is used to maintain the genetic diversity of the population, and it is designed to occur very rarely with a probability that is set to below 10%. The mutation operator is generally interpreted as flipping a bit or generating a random bit [21]. In the case of real-valued formulations, mutation is generally interpreted as randomly generating a value within the search space of the problem to be solved [69].

4.2.2 Implementation of Genetic Algorithm

The flow chart of a genetic algorithm is shown in Figure 4-7, and the pseudo-code for implementing a genetic algorithm is presented in Figure 4-8

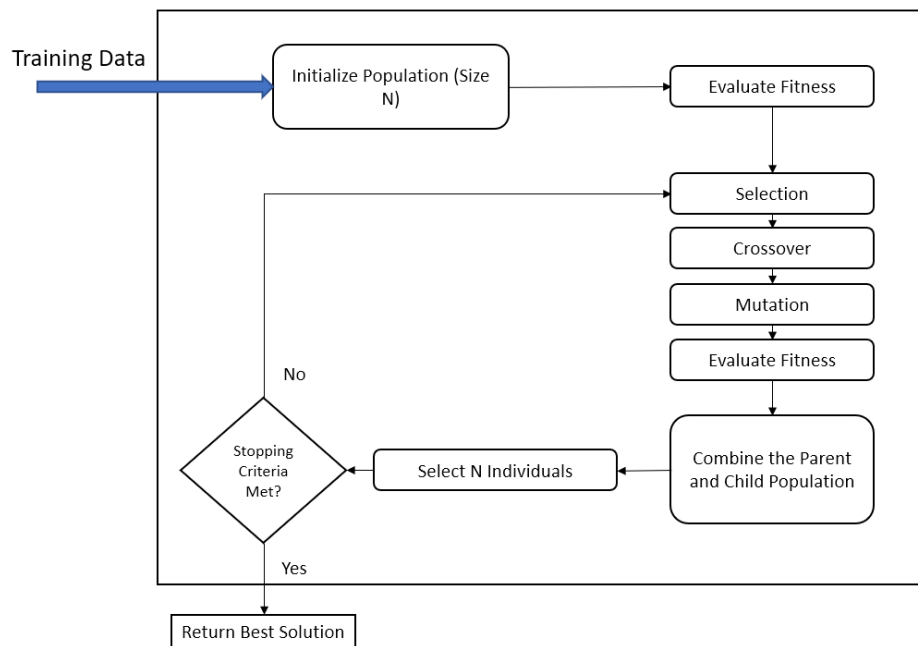


Figure 4-7: Flow Chart of a Genetic Algorithm

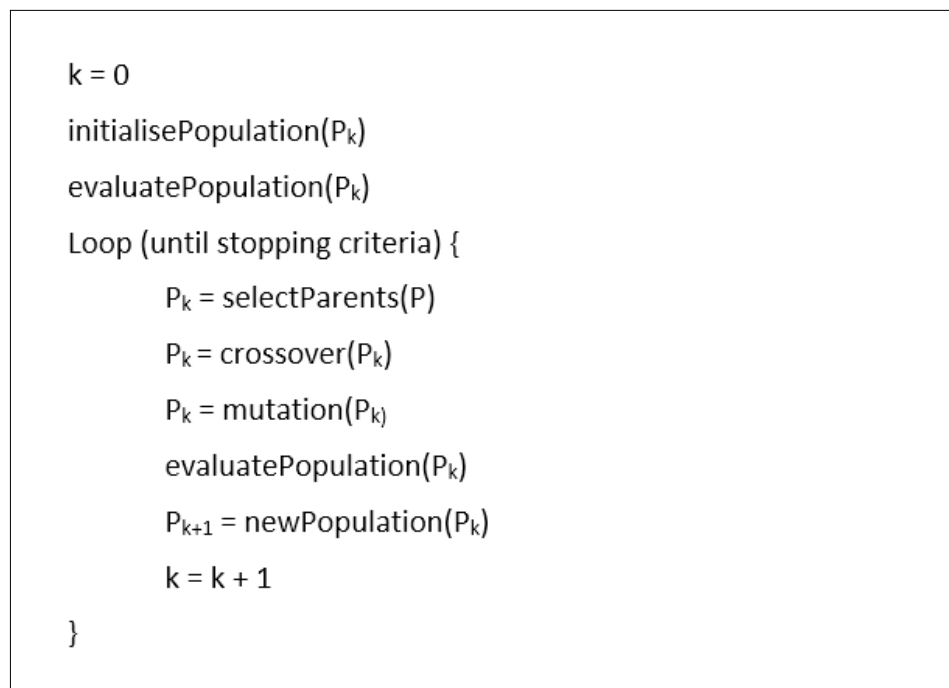


Figure 4-8: Genetic Algorithm Pseudo Code

4.3 Gradient Descent Optimization Algorithm

Gradient Descent is one of the most popular optimization algorithms which are used for optimizing Neural Networks and Deep Learning algorithms. In this thesis, the

gradient descent algorithm is used to train the stacked autoencoder, which is then used to pre-train the proposed Deep Type-2 fuzzy logic system. The training method using gradient descent is described in Chapter 7.

Gradient descent is an iterative algorithm that works on the principle of error-correction learning, and it is implemented in the following way:

In the initial step, the weights and bias of the neural network are generated randomly. The weight and bias values are either uniformly distributed between 0 and 1 or -1 and 1 based on the algorithm. Next, a single or multiple training data pairs are presented to the network, and an error is calculated using a cost function, which is chosen based on the type of problem that is being solved.

The error values are then used to identify the gradient of the cost function, which is then used to update the weights and bias of the network based on the learning rate using the following equation [99].

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta} J(\theta) \quad (4.4)$$

Where η is a constant that represents the learning rate, $\nabla_{\theta} J(\theta)$ represents the gradient of the cost function.

This process is repeated until all the training data pairs are presented to the network. The presentation of all the training data pairs to the network is termed as an epoch.

After each epoch, the performance of the network is evaluated against a validation dataset and then checked against the stopping criteria. The epochs are repeated until the stopping criteria are satisfied.

4.3.1 Adaptive Learning Rate

The learning rate is one of the most difficult hyperparameters to set because it significantly affects performance [100]. Using momentum in the algorithm somewhat mitigates this problem but at the expense of adding another hyperparameter. The solution to this problem is to automatically adapt the learning rate throughout the training process [100]. One of the most popular algorithms that use adaptive learning rate is described below.

4.3.1.1 Adam

Adaptive Moment Estimation (Adam) [101] is one of the most popular and recent adaptive learning algorithms. It is designed to combine the advantages of two other popular adaptive learning algorithms: AdaGrad [102] and RMSProp [103]. It is considered the best overall choice among the gradient descent algorithms [99]. Adam stores the exponentially decaying average of the past gradients as follows.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta) \quad (4.5)$$

Where β_1 is a hyperparameter and the authors suggest a default value of 0.9.

Adam also stores the decaying average of past squared gradients as follows.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_{\theta} J(\theta)^2 \quad (4.6)$$

Where β_2 is another hyperparameter, and the authors suggest a default value of 0.999.

As the m_t and v_t are initialized to zero, these values are biased towards zero. To counteract this effect, the authors have suggested the following correction.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (4.7)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (4.8)$$

This yields the final weight update function as follows.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (4.9)$$

Where the value ϵ is initialized to a small value such as 10^{-8} to ensure that there is no divide by zero error.

4.3.2 Implementation of Gradient Descent Algorithm

```

m0 = 0, v0 = 0, t = 0
β1 = 0.9, β2 = 0.999, ε = 10-8
For i = 1 to number of epochs
  Random.shuffle(data)
  t = t + 1
  For b in get_batches(data, 128)           //get batch of size 128
    grad = G(loss_function, b, p) //calculate gradient
    mt = β1 · mt-1 + (1 - β1) · grad //biased first moment estimate
    vt = β2 · vt-1 + (1 - β2) · grad2 //biased second raw moment estimate
    m̂t = mt / (1 - β1t) //bias corrected moment estimate
    v̂t = vt / (1 - β2t) //bias corrected second raw moment
    p = p - η · m̂t / (v̂t + ε) //update weights and bias
  Next b
  If(error(data_v, p_best) > error(data_v, p)) { //store weight and bias if error
    better p_best = p //is better than historic best
  }
Next i

```

Figure 4-9: Gradient Descent Pseudo Code [101]

The pseudo-code for the gradient descent algorithm used is presented in Figure 4-9.

The Adam algorithm is used for updating the weights and bias.

4.4 Summary

This chapter gave an overview of the optimization algorithms that are used in this thesis. These algorithms are Big Bang Big Crunch and genetic algorithms (GAs) which are used to train the proposed Deep Type-2 Fuzzy Logic System. The Stochastic Gradient Descent, which is used to train the stacked autoencoder and the outputs of hidden layers of this model is then used to train the proposed algorithm as an alternative training method.

The next chapter gives an overview of the problems or datasets used to evaluate the models examined in this thesis.

Chapter 5. Overview of the Datasets used in the Research

In this chapter, we will introduce all the datasets we use to test our XAI models. These datasets were collected with the goal of finding large real-world datasets with a high number of attributes/features to test the embedded feature selection capability of the proposed models presented in this thesis. We also collected datasets with a large number of instances and used a part of each dataset to pre-train the proposed models unsupervised, and then the models were retrained (supervised training) using the rest of the dataset.

Another goal was finding datasets from a variety of fields to check the applicability of our models to solve these problems. Hence, we collected datasets from a variety of fields such as Communication, Medical, Financial, Automotive etc.

We collected six datasets that relate to classification. We also collected five datasets that relate to regression. These datasets were collected from a variety of sources such as, BT, who provided us with two of these datasets one for classification and another for regression problems. The second source was the UCI machine learning repository [104], where we were able to identify four classification and three regression datasets.

Table 5-1: Summary of Datasets used in the Experiments

Dataset	Type	No of Attributes	No of Records	Records for Unsupervised Training	Records for Supervised Training
BT Customer Service	Classification	500	100,000	50000	50000
CLL Identification	Classification	3000	100000	50000	50000
IDA 2016	Classification	171	76000	16000	60000
Epileptic Seizure	Classification	178	11500	1500	10000
PD Speech	Classification	754	756	110	646
Santander CTP	Classification	200	400000	200000	200000
Wi-fi Localization	Regression	522	21100	2100	19000
Swiss Premium	Regression	199	53,000	25000	28000
CT Scan Region	Regression	385	53500	8000	45500
Song year	Regression	90	500000	50000	450000
BT PWA	Regression	44	30000	5000	25000

The third source was Kaggle, where we were able to identify two of their open-source datasets [105] [106]. The datasets are summarised in Table 5-1

5.1 Classification Problems

In this section, we present the list of datasets we used to perform classification to test the performance of our model in classification tasks.

5.1.1 BT Customer Service (BTCS)

The dataset is supplied by BT, and the data is about predicting whether a customer is contacting BT to report problems with their broadband connection or not. If the problem is related to their connection, i.e., slow connection, broadband not working etc, then an engineer will be sent to fix the issue. If the problem is not related to the connection itself, the problem could be about issues that could be solved over a phone call without the need for an engineer visit.

Suppose the customer's problem can be predicted in advance based on the customers' historical behaviour or demographic information. The customer's problem can be dealt with appropriately without the need for any unnecessary engineer visits to the customers' premises. This would be a huge cost saving for both the customer and BT as engineer visits are expensive. This would also lead to a more pleasant customer experience as many trivial problems can be resolved more quickly over a phone call leading to improved customer experience.

The data consists of 500 attributes including anonymised demographic information, services usage, previous fault details, broadband speeds, etc. with about 100,000 records. We used 50000 records for unsupervised training and 50,000 records for supervised training.

5.1.2 CLL Identification (CLL)

The dataset is about identifying Chronic Lymphocytic Leukemia (CLL) using aberrant chromatin features. CLL is a type of cancer, and it has been identified that there is a link between aberrant chromatin features and CLL. However, how they are related is still an open question. So, they collected genomic sequence from a set of healthy individuals and individuals with CLL. The goal of this analysis is to identify how the aberrant chromatin features relate to CLL. XAI is one method that can be used to identify this connection if the AI models can predict CLL using this dataset and predictions of the model can be interpreted then the relationship between the inputs and output can be explored using the explanations provided by the AI. Further experiments would be needed to confirm that there is a relationship. This dataset points towards one of the major uses of XAI systems.

The dataset has about 100000 records and 3000 inputs [107]. We used 50000 records for unsupervised training and 50000 records for supervised training.

5.1.3 IDA 2016 (IDA)

The data is collected from heavy Scania trucks in everyday usage. The data is collected from the Air Pressure system (APS), which generates pressurised air which is used in various functions on the truck, such as braking and gear changes. The data consists of a subset of all available data, selected by experts.

The data consists of truck failures, and the goal is to identify the truck failures which are related to the APS. i.e., the dataset's positive class consists of component failures for a specific component of the APS system. The negative class consists of trucks with failures for components not related to the APS.

The dataset has 76000 records with 171 attributes [104]. We use 16000 of these records for unsupervised training and 60000 records for the supervised training.

5.1.4 Epileptic Seizure (ES)

The dataset is a pre-processed dataset commonly used for epileptic seizure detection. The data has 178 data points, and each data point is the value of the EEG recording at a different point in time [108].

The original dataset from the reference consists of 500 files, with each file representing a single subject/person. Each file is a recording of brain activity for 23.6 seconds. The files are then converted into time-series by sampling the files into 4097 data points. Each data point is the value of the EEG recording at a different point in time. So, we have a total of 500 individuals, with each individual having 4097 data points [108].

Everyone's data is divided into 23 parts, each part contains 178 data points for 1 second, and each data point is the value of the EEG recording at a different point in time. So now we have $23 \times 500 = 11500$ pieces of information(row), each information contains 178 data points for 1 second(column). The last column represents the target, where the positive class represents the list of records where there is an epileptic seizure, and the Negative class represents the list of records where there is no problem [108] [104].

We used 1500 records for unsupervised training and 10000 records for the supervised training.

5.1.5 PD Speech (PDS)

This dataset is gathered from a study where the data was gathered from 188 patients with Parkinson's Disease (107 men and 81 women) with ages ranging from 33 to 87. The control group consists of 64 healthy individuals (23 men and 41 women) with ages varying between 41 and 82. During the data collection process, the microphone is set to 44.1 kHz and following the physician examination, the sustained phonation of the vowel "a" was collected from each subject with three repetitions [109].

Various speech signal processing algorithms including Time-Frequency Features, Mel Frequency Cepstral Coefficients (MFCCs), Wavelet Transform based Features, Vocal Fold Features and TWQT features have been applied to the speech recordings of Parkinson's Disease (PD) patients to extract clinically useful information for PD assessment. The data has about 756 records with 754 attributes [109].

We used 110 records for unsupervised training and 646 records for the supervised training.

5.1.6 Santander CTP (SCTP)

This dataset is from Santander. In this dataset, the goal is to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The data provided has the same structure as the real data that Santander has available to solve this problem. Once the customers who are likely to make a transaction are identified, we can use this information in a variety of ways. For example, if a transaction, that the model thinks is unlikely, happens then we might add additional checks to make sure it is the customer making this transaction. This could potentially identify and stop fraudulent transactions. This could also be used in targeted

marketing, by approaching the customers who are likely to make a transaction, leading to a higher chance of a product sale, etc.

The dataset consists of around 400,000 records with 200 features [105]. It is a binary classification task, and the dataset is skewed with only 20098 positive records while the rest of the records are negative. We used 200,000 records for unsupervised training and 200000 records for supervised training.

5.2 Regression Problems

Regression is a statistical measure that attempts to determine the strength of the relationship between one or more dependent variables and a series of independent variables. Here we list a set of datasets we use for performing regression analysis for testing the ability of our model.

5.2.1 Wi-fi Localization (WL)

This dataset is from an indoor user localization problem [110] [104]. Many applications need to know the location of a user in the world to provide their services. This means that automatic user localization has been a hot topic of research. The goal of the Automatic user localization is the estimation of the position of the user using an electronic device, usually a mobile phone. This problem can be solved easily when the user is outdoor thanks to the availability of GPS sensors on most mobile phones. However, this problem becomes much more difficult when the user is indoor, mainly due to the loss of GPS signal in indoor environments.

This database is focused on WLAN fingerprint-based solutions (also known as Wi-Fi Fingerprinting). Wi-Fi Fingerprinting can be characterized by the detected Wireless Access Points (WAPs) and the corresponding Received Signal Strength Intensities

(RSSI). The database was collected in 2013 at the Universität Jaume I by 20 different users using 25 android devices. This dataset consists of around 21,000 training records with 529 attributes.

- The first 520 of these attributes are the Wi-Fi fingerprint of 520 WAPs.
- Attribute 521 is the Longitude
- Attribute 522 is the Latitude
- Attribute 523 is the Floor where the datum was collected.
- Attribute 524 is the building (0 to 2)
- Attribute 525 is the Internal ID number to identify the Space (office, corridor, classroom) where the datum was collected.
- Attribute 526 is the Relative position with respect to Space (1 - Inside, 2 - Outside in Front of the door).
- Attribute 527 is the User identifier.
- Attribute 528 is the Android device identifier.
- Attribute 529 is the time when the datum was collected.

We use 522 of the above attributes in our experiments and ignore attributes 523 to 529.

We used around 2100 records for unsupervised training and 19000 records for supervised training.

5.2.2 Swiss Premium (SP)

This dataset is collected from the statistics published by the Swiss government. It consists of information about insurers, regions, healthcare information and demographic information. The goal of this data is to predict health insurance premiums

that will be charged. This is valuable because the premiums are communicated yearly at the end of September. However, if these figures could be predicted earlier, the consumers would have more time to plan for a possible change. This is helpful as Switzerland has one of the world's highest health insurance premium rates. With the use of XAI, we can also identify the features or factors that influence the premium rates. This is helpful as it allows the consumers to be informed about their lifestyle choices that could lead to higher premiums. For example, smokers could be charged higher premiums as they are more likely to get respiratory diseases.

This dataset consists of around 53,000 records with 199 inputs. We used 25000 records for unsupervised training and the other 28000 records for supervised training. For more details about this dataset, please refer to the following [106].

5.2.3 CT Scan Region (CTSR)

This dataset is about finding the relative location of computerized tomography (CT) slices on the axial axis. This data was used in [111] [112].

This dataset consists of 53500 CT images from 74 different patients. Each CT scan is represented by two histograms; the first histogram describes the location of the bone structure in the image, while the second histogram represents the location of the air incursion in the body. These two histograms are combined to form the final data vectors. The output was constructed manually by annotating ten different locations on the CT scan with known locations.

There are 385 attributes for each record; the first 240 attributes are the histogram that represents the location of the bone structure. The attributes from 241 to 385 are the histogram that represents the air incursions. The output is a value in the range 0 to 180,

with 0 representing the top of the head and 180 representing the soles of the patient's feet [104].

We used 8000 records for unsupervised training and the rest of the records for supervised training.

5.2.4 BT PWA (BTP)

This dataset is collected from British Telecom in the UK. The data was collected from around 520 Work Areas (WA) over a one-year period. Each datum contains performance information for each of the WA over a one-week period. With the following Attributes.

- Rank: Relative rank to the other WAs in the week where the performance was measured
- Missed Appointments: Number of tasks per day which could not be completed within the appointed time period
- Productivity: average productivity of the engineers working in the WA for the particular week being examined
- Service Level CL1: Percentage of high priority tasks completed on time
- Service Level CL2: Percentage of low priority tasks completed on time
- Travel: average travel time of the engineers
- Contractors: average number of contractors employed in the WA per week
- On day Utilization: Utilization of engineers per day measured at the end of each day
- Economic Utilization
- Overtime: average overtime per day by engineers in the WA

- Loans: average number of engineers loaned per day from other WAs

To predict the future performance of the WA at time interval t , we use the performance data at previous time intervals, i.e., $x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}$ where x_t contains the 11 abovementioned performance measures. The target is to predict the Rank of the WA at time interval t .

The data consists of around 30000 records and 44 attributes, and the data was used in the following paper [48] [96].

5.2.5 Song year (SY)

This dataset is a subset of the million song dataset [104] [113]. The target is to predict the year in which a song was released by analysing its audio features. Songs are mostly western, commercial tracks ranging from 1922 to 2011.

There are 90 attributes, 12 of these attributes represent the timbre average, and these features are extracted from the ‘timbre’ features from The Echo Nest API, currently owned by Spotify. These values are high-level abstractions of the spectral surface, ordered by degree of importance. For completeness, however, the first dimension represents the average loudness of the segment; the second emphasizes brightness; third is more closely correlated to the flatness of a sound; fourth to sounds with a stronger attack; etc. The average and covariance are taken over all segments of the songs; each segment is described by the 12-dimensional timbre vector. The rest of the 78 attributes are covariance over all the segments.

We used around 50000 records for unsupervised training and the other 450000 records for supervised training.

5.3 Summary

The Chapter presents the six categorical and five regression datasets used in the experiments for the rest of the thesis. It explains that these datasets were chosen because they are high dimensional and can be used to test the embedded feature selection capability of the proposed model. It explains that real-world datasets were chosen from BT and other sources to test the ability of the model to solve real-world problems.

The chapter also explains how a part of the datasets will be used for unsupervised training while the rest of the data will be used for supervised training of the proposed model.

The next chapter introduces the proposed system in detail.

Chapter 6. The Proposed Deep Type-2 Fuzzy Logic System

One of the problems of modelling datasets with many features using Fuzzy Logic Systems (FLS) is that the number or amount of rules required is very large, and this reduces the interpretability of the system. For example, for a system with 30 features if we use three antecedents per rule, then the number of possible rules in the rule base will be $3^{30} = 2 * 10^{14}$ rules. Such a large rule base presents us with two main problems. Firstly, the computational complexity involved in calculating all the rules for each prediction means that such a system will be slow. Secondly, although the individual rules will be intelligible to the end-users of these systems, the sheer number of rules means that these models effectively become opaque.

One way to resolve this is to reduce the number of rules. Some of the rules in the full set will be redundant and reducing the number of rules will not impact the accuracy of the model initially. However, as more and more rules are removed the accuracy of the model will reduce. Another challenge is that given that the system is operating on a reduced ruleset, there will be situations where none of the rules is fired, and thus no output is produced. There are a few methods to mitigate this problem; one such method is presented in [48, 49] where the similarity of rules, i.e., for each of the rules in the FLS, similar rules are created by checking if using the other membership functions or linguistic labels of the antecedents trigger the rules, and based on the distance between the linguistic labels, the firing levels are calculated. However, there will still be a need

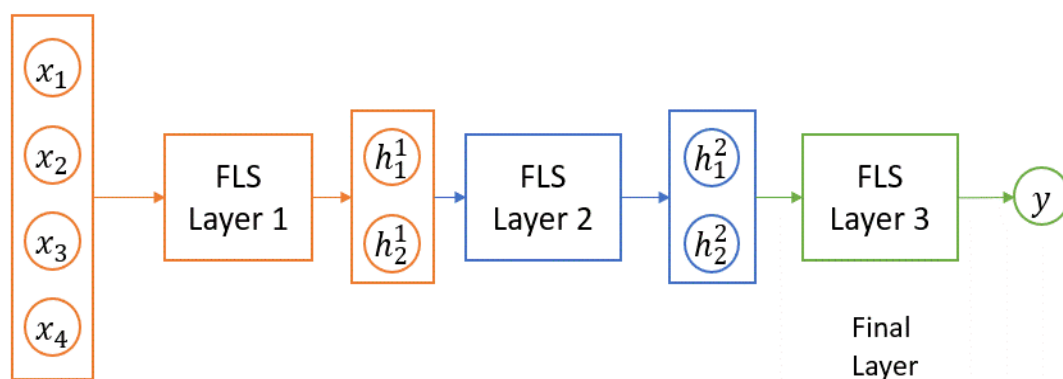


Figure 6-1: A Deep Type-2 Fuzzy Logic System Architecture

to perform feature selection for such systems to operate, and there is no way to learn from unlabelled data (input data with no target outcome).

Hence, we propose a Deep Type-2 Fuzzy Logic System (D2FLS) (depicted in Figure 6-1). This system comprises of two or more interval type-2 FLSs where the output of the first FLS is used as the input of the second FLS and the output of the second FLS is used as the input of the third FLS etc. This system is inspired by Stacked autoencoders (SAE) or multi-layer neural networks [61]. The goal here is for each of FLSs to aggregate the input features into more complex compound features that become the outputs of the FLS, thereby reducing the number of features in the subsequent FLS layers. This process is repeated by each of the FLSs in the D2FLS until we generate the final output. The advantage of such a system is that the total number of rules required to represent the whole model will be low, which helps in reducing the computational complexity of the system. Furthermore, since the number of rules will be small, the system will be more interpretable than an equivalent FLS which will have a larger number of rules.

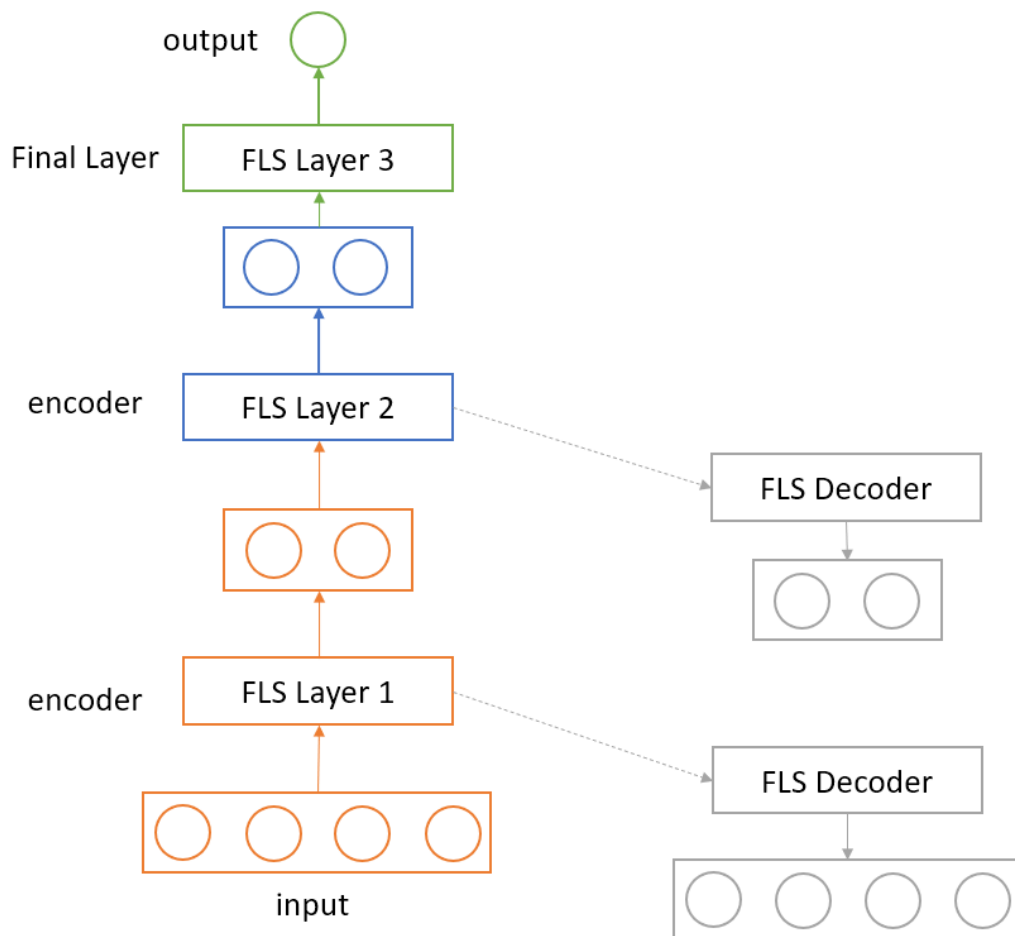


Figure 6-2: Layer Wise Training D2FLS

The training of the D2FLS presents a few problems, as there are multiple FLSs, we cannot use single-pass methods such as Wang Mender(WM) [38] [93] described in Section 3.5.1 to train the D2FLS because we do not have a simple way of determining the outputs of the hidden layer in advance (we only know the inputs and outputs of the final layer in advance). For the same reason, we cannot train the hidden layer separately. Hence, we propose to train the D2FLS using greedy layer-wise training [61] similar to a Stacked autoencoder (depicted in Figure 6-2). In detail, for a D2FLS with three FLS as depicted in Figure 6-1, we will train the first layer as a Fuzzy autoencoder (FAE) (depicted in Figure 6-3). Next, we discard the decoder and use the output of the

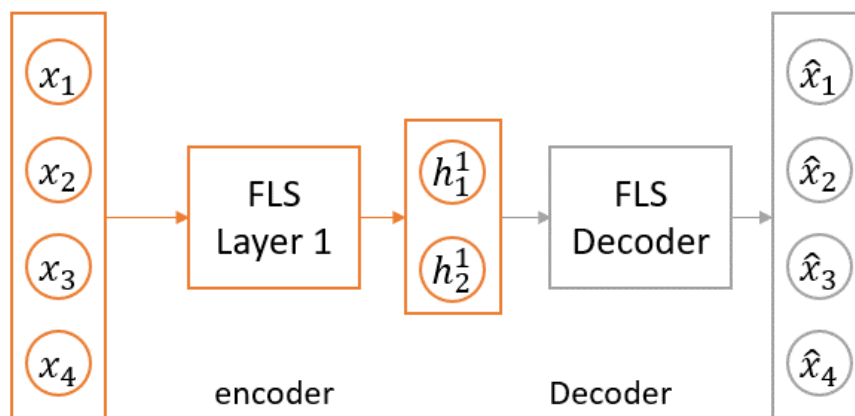


Figure 6-3: Fuzzy Autoencoder Architecture

encoder part of the FAE as the input for the second layer and train that layer as an FAE. Finally, we will add the final layer to the encoders of the first two FAEs and train the three FLSs in a supervised way to get the final output. The idea behind this approach is to use unsupervised training of the FAEs to learn essential features or combine features. The details of the training method are presented in the following sections.

6.1 Model Representation

A key issue in Genetic algorithms and BB-BC algorithm is the choice of the encoding scheme, i.e., how to represent a solution to the problem, in this case, an FLS, as a chromosome [69]. The choices are generally binary or floating-point. In our case, we use floating-point numbers to represent the D2FLS, and the details of the representation are presented in the following sections.

6.1.1 Representation of the Type-1 models

We encode the parameters of a type-1 FLS as real-valued numbers. There are three sets of parameters that can be used to characterise the type-1 FLS, first the membership

functions or linguistic variables that represent the input or antecedents, second the membership functions or the linguistic variables that represent the outputs or consequents and the third set of values represent the rules of the FLS. The MFs for the antecedents and the consequents are represented using the same encoding.

6.1.1.1 Membership Function Representation

The representation of the parameters of the membership functions changes based on the type of membership function; hence, below, we show the representation for the three types of membership functions.

6.1.1.1.1 Trapezoidal Membership function representation

A trapezoidal MF can be defined using the following formula.

$$\mu_j(x) = \mu_j(x; a, b, c, d) = \begin{cases} (x-a)/(b-a) & \text{if } a \leq x < b \\ 1 & \text{if } b \leq x \leq c \\ (d-x)/(d-c) & \text{if } c < x \leq d \\ 0 & \text{if } x > d \text{ or } x < a \end{cases} \quad (6.1)$$

Where $x \in X = [0,1]$, then the parameters of the linguistic variable M , which is defined by j trapezoidal membership functions, can be represented in the following format.

$$M = b_1, c_1, b_2, c_2, \dots, b_j, c_j \quad (6.2)$$

We can see from (6.2) that each fuzzy set is represented by two parameters, b , and c . The parameters a and c of the k^{th} membership function can be defined using the parameters b and c as follows:

$$\begin{aligned} a_k &= c_{k-1} \\ d_k &= b_{k+1} \end{aligned} \quad (6.3)$$

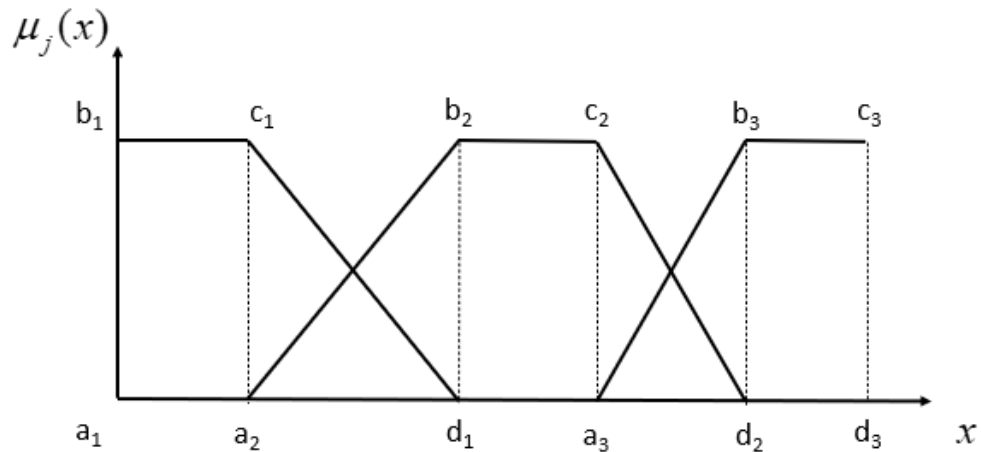


Figure 6-4: Representation of a Trapezoidal Type-1 Membership Function

Where $a_1, b_1 = 0$ and $c_j, d_j = 1$ as depicted in Figure 6-4.

In other words, from Figure 6-4, we can see that two values represent each fuzzy set and the start and endpoints of the MFs are anchored to the previous and the next MF, respectively. The start of the first MF and the end value of the last MF are set to 0 and 1 respectively. So, the first membership function is represented by four values, it starts at 0, the top two vertices are represented by b_1 and c_1 , and the final vertex at the end is represented by b_2 . The four vertices of the second membership function are represented by start= c_1 , top= b_2 and c_2 , and end = b_3 etc.

6.1.1.1.2 Triangular MF Representation

A Triangular Membership Functions is defined using the following formula [38].

$$\mu_j(x) = \mu_j(x; a, b, c) = \begin{cases} (x-a)/(b-a) & \text{if } a \leq x < b \\ (c-x)/(c-b) & \text{if } b \leq x \leq c \\ 0 & \text{if } x > c \text{ or } x < a \end{cases} \quad (6.4)$$

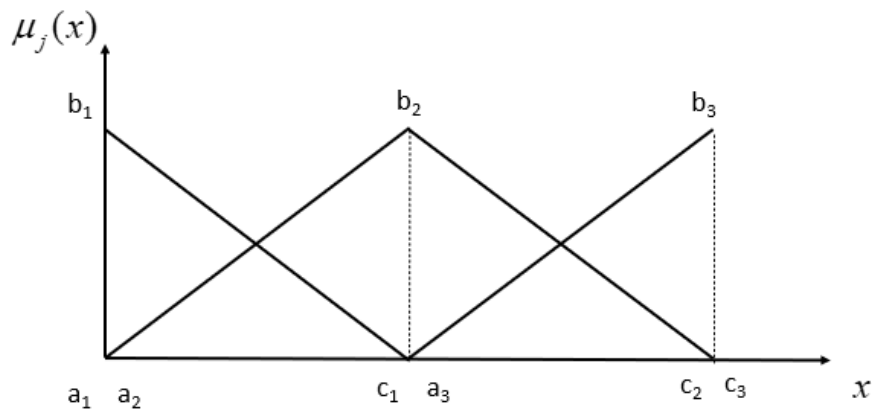


Figure 6-5: Representation of a Triangular Type-1 Membership Function

Where $x \in X = [0, 1]$ then the linguistic variable M which is defined by j Triangular membership functions can be represented in the following format.

$$M = b_1, b_2, \dots, b_j \quad (6.5)$$

We can see from (6.5) that each fuzzy set is represented by one parameter, b . The parameters a and c of the k^{th} membership function can be defined using the parameter b as follows.

$$a_k = b_{k-1} \quad (6.6)$$

Where $a_1, b_1 = 0$ and $b_j, c_j = 1$ as depicted in Figure 6-5.

In other words, from Figure 6-5, we can see that each fuzzy set is represented by one value and the start and endpoints of the MFs are anchored to the values of previous and the next MF, respectively, and the start of the first MF and the endpoint of the last MF are set to 0 and 1, respectively. So, the first membership function is represented by

three values, and it starts at 0, the top of the triangle is represented by b_1 , and the final

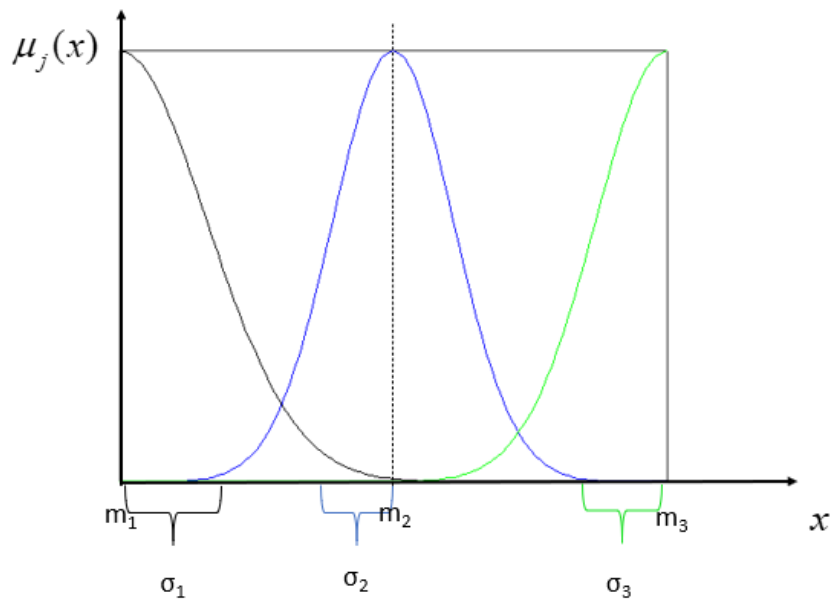


Figure 6-6: Representation of a Gaussian Type-1 Membership Function

vertices at the end is represented by b_2 . The three vertices of the second membership function are represented by start= b_1 , top= b_2 and end = b_3 etc.

6.1.1.1.3 Gaussian Membership Function Representation

A Gaussian Membership function is defined using the following formula [38].

$$\mu_j(x) = \mu_j(x; m, \sigma) = \exp \left[- \left(\frac{(x-m)}{2\sigma} \right)^2 \right] \quad (6.7)$$

Where $x \in X = [0,1]$, then the linguistic variable M , which is defined by j Gaussian membership functions (depicted in Figure 6-6), can be represented in the following format.

$$M = m_1, \sigma_1, m_2, \sigma_2, \dots, m_j, \sigma_j \quad (6.8)$$

We can see from (6.8) that each fuzzy set is represented by two-parameter the mean and standard deviation of the gaussian MF.

6.1.1.2 Rules Representation

Each Rule of the FLS has two components the antecedents of the rule and the consequents of the rule. The antecedents and consequents of each rule of the FLS are encoded using the below representation.

$$R_l = r_1^1, r_2^1, \dots, r_1^a, r_2^a, c_1, \dots, c_b \quad (6.9)$$

Where R_l represents the l^{th} rule of the FLS with a antecedents and b consequents per rule. Each antecedent is represented by 2 points; the first point represents the i^{th} input feature or linguistic variable; the second point represents the j^{th} membership function of that linguistic variable. The consequents are represented by a single point which represents the j^{th} membership function of the b^{th} consequent.

6.1.1.3 FLS Representation

Finally, to represent the parameters of an FLS, we combine the parameter representations of the MFs from (6.2), (6.5) or (6.8), based on the type of MF chosen, and rules from (6.9) to get the following representation.

$$N = M_1, \dots, M_i, \dots, M_{i+k}, R_1, \dots, R_l \quad (6.10)$$

Where M_i represents the membership functions for the i input features or antecedents of the FLS and M_{i+k} represents the MFs for the k^{th} output or consequent of the FLS using (6.2), (6.5) or (6.8).

6.1.2 Representation of IT2 Models

The representation of the parameters of an IT2 FLS is very similar to the type-1 FLS parameter representation explained in the previous section the only difference is that each of the fuzzy sets or MFs will have a Footprint of Uncertainty (FOU) value. The rules of the IT2 models and T1 models are identical.

6.1.2.1 Membership Function Representation

6.1.2.1.1 Trapezoidal IT2 MF representation

An IT2 fuzzy set can be represented by its left and right endpoints. The two endpoints are associated with two type-1 MFs that are referred to as upper and lower membership. For a trapezoidal MF, these two MFs can be defined by adding a small FOU to (6.1) as follows.

$$\bar{\mu}_j(x) = \bar{\mu}_j(x; \bar{a}, \bar{b}, \bar{c}, \bar{d}) = \begin{cases} (x - \bar{a}) / (\bar{b} - \bar{a}) & \text{if } \bar{a} \leq x < \bar{b} \\ 1 & \text{if } \bar{b} \leq x \leq \bar{c} \\ (\bar{d} - x) / (\bar{d} - \bar{c}) & \text{if } \bar{c} < x \leq \bar{d} \\ 0 & \text{if } x > \bar{d} \text{ or } x < \bar{a} \end{cases} \quad (6.11)$$

$$\underline{\mu}_j(x) = \underline{\mu}_j(x; \underline{a}, \underline{b}, \underline{c}, \underline{d}) = \begin{cases} (x - \underline{a}) / (\underline{b} - \underline{a}) & \text{if } \underline{a} \leq x < \underline{b} \\ 1 & \text{if } \underline{b} \leq x \leq \underline{c} \\ (\underline{d} - x) / (\underline{d} - \underline{c}) & \text{if } \underline{c} < x \leq \underline{d} \\ 0 & \text{if } x > \underline{d} \text{ or } x < \underline{a} \end{cases} \quad (6.12)$$

Where $x \in X = [0, 1]$, $\bar{\mu}_j(x)$ is the upper membership function, $\bar{a} = a + FOU / 2$, $\bar{b} = b + FOU / 2$, $\bar{c} = c + FOU / 2$ and $\bar{d} = d + FOU / 2$. $\underline{\mu}_j(x)$ is the lower membership function, $\underline{a} = a - FOU / 2$, $\underline{b} = b - FOU / 2$, $\underline{c} = c - FOU / 2$ and $\underline{d} = d - FOU / 2$.

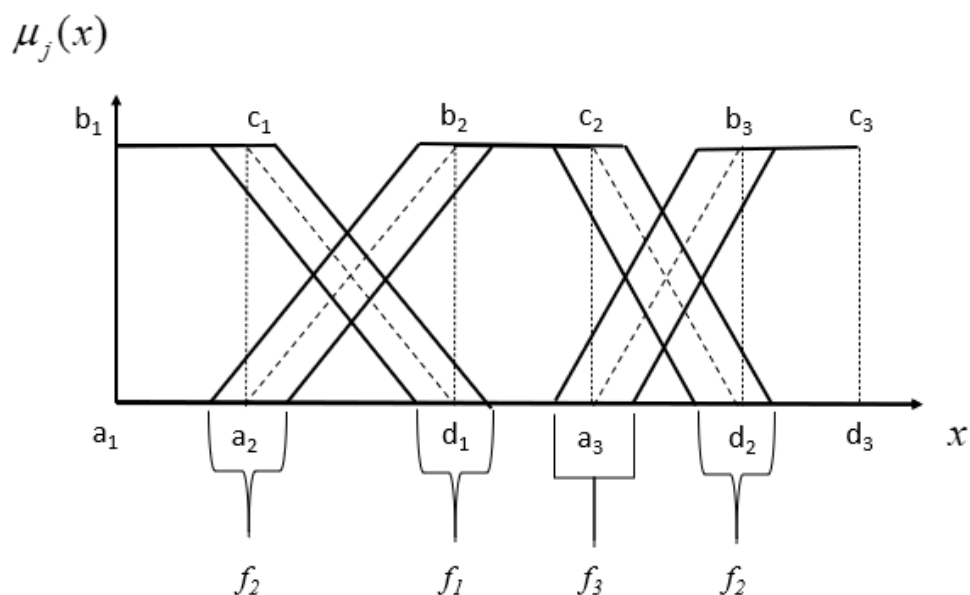


Figure 6-7: Representation of a Footprint of uncertainty (FOU) for Trapezoidal MFs

Hence, the parameters of the linguistic variable M , which is defined by j IT2 trapezoidal MFs (depicted in Figure 6-7), can be represented by modifying the representation of the type-1 Trapezoidal MF in (6.2) as follows.

$$T2 = b_1, c_1, b_2, c_2, \dots, b_j, c_j, f_1, f_2, \dots, f_j \quad (6.13)$$

Where f_j represents the FOUs of the j^{th} fuzzy set. Here, we add one FOU for each of the fuzzy sets.

6.1.2.1.2 Triangular IT2 MF Representation

An IT2 fuzzy set can be represented by its left and right endpoints. The two endpoints are associated with two type-1 MFs that are referred to as upper and lower membership. For IT2 Triangular MF (depicted in Figure 6-8) the two endpoints can be defined by modifying (6.4) as follows.

$$\bar{\mu}_j(x) = \bar{\mu}_j(x; \bar{a}, \bar{b}, \bar{c}) = \begin{cases} (x - \bar{a}) / (\bar{b} - \bar{a}) & \text{if } \bar{a} \leq x < \bar{b} \\ (\bar{c} - x) / (\bar{c} - \bar{b}) & \text{if } \bar{b} \leq x \leq \bar{c} \\ 0 & \text{if } x > \bar{c} \text{ or } x < \bar{a} \end{cases} \quad (6.14)$$

$$\underline{\mu}_j(x) = \underline{\mu}_j(x; \underline{a}, \underline{b}, \underline{c}) = \begin{cases} (x - \underline{a}) / (\underline{b} - \underline{a}) & \text{if } \underline{a} \leq x < \underline{b} \\ (\underline{c} - x) / (\underline{c} - \underline{b}) & \text{if } \underline{b} \leq x \leq \underline{c} \\ 0 & \text{if } x > \underline{c} \text{ or } x < \underline{a} \end{cases} \quad (6.15)$$

Where $x \in X = [0, 1]$, $\bar{\mu}_j(x)$ is the upper membership function, $\bar{a} = a + FOU/2$, $\bar{b} = b + FOU/2$ and $\bar{c} = c + FOU/2$. $\underline{\mu}_j(x)$ is the lower membership function, $\underline{a} = a - FOU/2$, $\underline{b} = b - FOU/2$ and $\underline{c} = c - FOU/2$.

Hence, the parameters of the linguistic variable M , which is defined by j IT2 triangular MFs (depicted in Figure 6-8), can be represented by modifying the representation of the type-1 Triangular MF in (6.5) as follows.

$$T2 = b_1, b_2, \dots, b_j, f_1, f_2, \dots, f_j \quad (6.16)$$

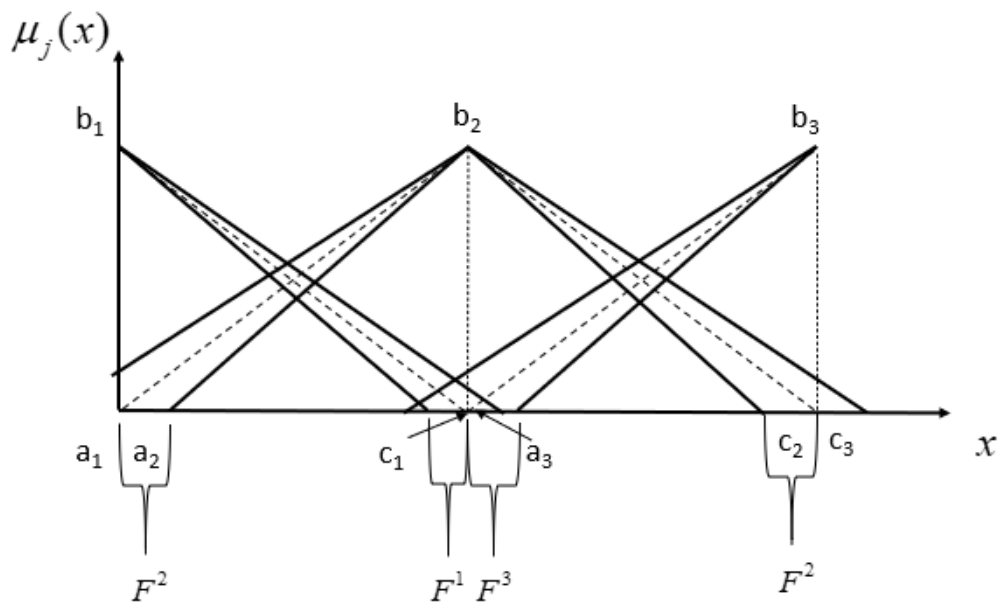


Figure 6-8: Representation of a Footprint of uncertainty (FOU) for Triangular MFs

Where f_j represents the FOU of the j^{th} fuzzy set. Here we add one FOU for each of the fuzzy sets.

6.1.2.1.3 Gaussian IT2 MF representation

An IT2 fuzzy set can be represented by its left and right endpoints. The two endpoints are associated with two type-1 MFs that are referred to as upper and lower membership. Thus, an IT2 Gaussian MF with uncertain standard deviation (depicted in Figure 6-9) can be defined as follows.

$$\bar{\mu}_j(x) = \bar{\mu}_j(x; m, \bar{\sigma}) = \exp \left[- \left(\frac{(x - m)}{2\bar{\sigma}} \right)^2 \right] \quad (6.17)$$

$$\underline{\mu}_j(x) = \underline{\mu}_j(x; m, \underline{\sigma}) = \exp \left[- \left(\frac{(x - m)}{2\underline{\sigma}} \right)^2 \right] \quad (6.18)$$

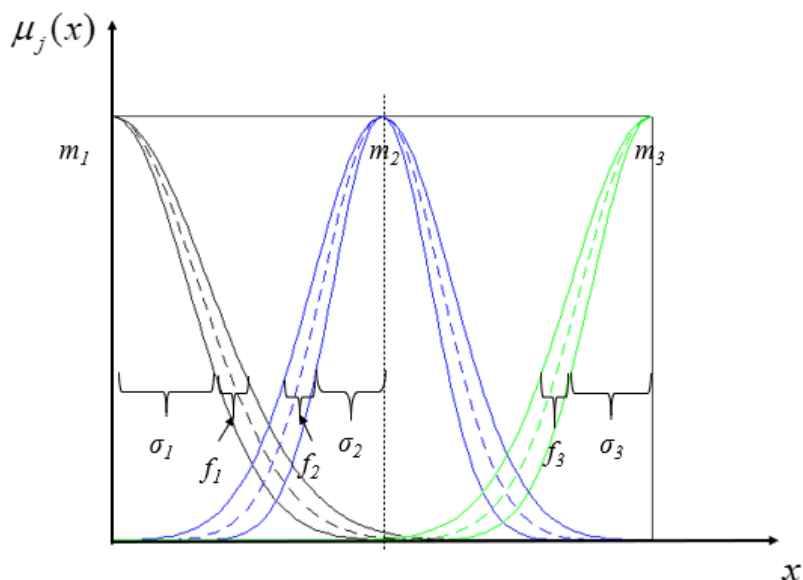


Figure 6-9: Representation of the FOU of a Gaussian IT2 MF

Where $x \in X = [0,1]$, $\bar{\mu}_j(x)$ is the upper membership function, $\bar{\sigma} = \sigma + FOU$. $\underline{\mu}_j(x)$ is the lower membership function and $\underline{\sigma} = \sigma$.

Hence, the parameters of the linguistic variable M which is defined by j IT2 Gaussian MF with uncertain standard deviation (depicted in Figure 6-9) can be represented by modifying the representation of the type-1 Gaussian MF in (6.8) as follows.

$$M = m_1, \sigma_1, m_2, \sigma_2, \dots, m_j, \sigma_j, f_1, f_2, \dots, f_j \quad (6.19)$$

In Figure 6-9 an input with three fuzzy sets is represented, we can see that the standard deviation represented in σ_j is used as the standard deviation for the lower MF and we add the FOU to this value to get the standard deviation of the upper MF.

6.1.2.2 IT2 FLS representation

To represent the IT2 FLS, we combine the MF representation in (6.13), (6.16) or (6.19) based on the type of MF and the representation of the rules in (6.9) to get the following representation.

$$T = T2_1, \dots, T2_i, \dots, T2_{i+k}, R_1, \dots, R_l \quad (6.20)$$

Where $T2_i$ represents the membership functions for the i input features or linguistic variables FLS and $T2_{i+k}$ represents the MFs for the k^{th} output or consequent of the FLS using (6.13), (6.16) or (6.19).

6.2 Layer Wise Training of The D2FLS

As discussed previously, we train the Membership Functions and the rulebase of the D2FLS using a method similar to the greedy layer-wise training method used for training Autoencoders [61]. This training method (depicted in Figure 6-2) can be divided into two phases; in the first phase, we train the hidden FLSs as a Fuzzy Autoencoders using unsupervised data. We then train the FLSs one layer at a time until we have trained all the FLS layers except the final output layer. In the second phase, we add the final output layer to the encoders of all the hidden layers and retrain the whole D2FLS using supervised data. The details of the training are given in the following subsections.

6.2.1 Fitness Function

6.2.1.1 Average Recall

We used Average Recall as the fitness function to train all the models for all the classification datasets.

$$Recall_{avg} = \frac{Recall_{positive} + Recall_{negative}}{2} \quad (6.21)$$

$$Recall_{positive} = \frac{tp}{tp + fn} \quad (6.22)$$

Where True positive tp is the number of correct positive predictions, and False negative fn is the number of incorrect negative predictions.

$$Recall_{negative} = \frac{tn}{tn + fp} \quad (6.23)$$

Where True Negative tn is the number of correct negative predictions, and False Positive fp is the number of incorrect positive predictions.

6.2.1.2 Mean Absolute Error

We used Mean Absolute Error (MAE) as the fitness function or cost function for the regression datasets.

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (6.24)$$

Where y_i is the desired output, \hat{y}_i is the actual output of the model and n is the number of inputs.

6.2.2 Hidden Layer Training

A Fuzzy Autoencoder (FAE), depicted in Figure 6-3, comprises of 2 FLSs that are trained to attempt to map its input to its output, i.e., the output of the first FLS (encoder) is the input for the second FLS (decoder) and the output of the second FLS is the

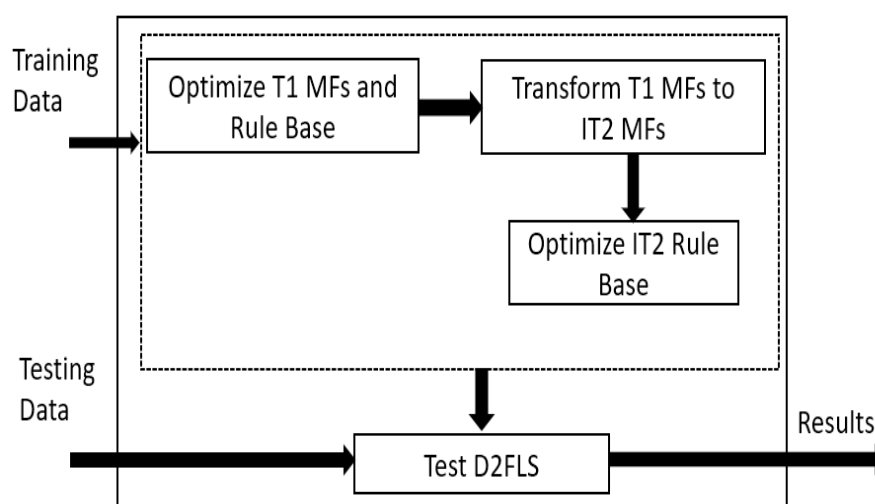


Figure 6-10: Training Algorithm for IT2FLS

reconstructed input of the first FLS. The idea here is to provide a set of constraints to force the FAE to prioritize and learn the most useful properties of the input data and to make sure the system does not merely learn the identity function. The constraints that we add to the FAE are as follows.

- Restrict the number of outputs of the encoder, and we will call it the hidden layer.
- We reduce the number of consequents or outputs of the encoder when compared to the inputs.
- We restrict the number of rules in the rule base of the 2 FLS that comprise the FAE
- We force the FAE to use the same linguistic labels for the consequents of the encoder and the antecedents of the decoder.
- We force the FAE to use the same linguistic labels for the antecedents of the encoder and the consequents of the decoder.

These constraints reduce the number of parameters that must be trained in the FAE during the optimization process. This simplifies the training process allowing it to be trained much more quickly and efficiently. The intuition behind this idea is derived from Autoencoders in neural networks [100]. The two parts of the FAE are represented as follows:

$$h = f(x_p) \tag{6.25}$$

Where h is a vector that represents the output of the encoder $f(x)$

$$\hat{x}_p = g(h) \tag{6.26}$$

Where \hat{x} is the output of the decoder, i.e., the reconstructed input.

To optimize the FAE, the MFs and the rule base of the FAE are optimized using an optimization algorithm and the training is divided into three steps, depicted in Figure 6-10. The goal of the optimization algorithm is to minimize a cost function such as MAE (6.24), which is modified as follows:

$$MAE = \frac{\sum_{i=1}^p |h^{k-1} - \hat{h}^{k-1}|}{p} \quad (6.27)$$

Where \hat{h}^{k-1} is the reconstructed input of the $k-1^{th}$ encoder, p is the number of instances in the training data and h^0 is the input vector of the training dataset.

6.2.2.1 Optimize Type 1 FAE

To optimize the type-1 FAE, we use an optimisation algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). The first step in these algorithms is to encode the parameters (to be tuned) of the two FLS (encoder and decoder) that comprise the FAE into individuals. Each individual represents a possible solution to the optimization problem.

There are three sets of parameters to be tuned for each of the FLS of the FAE, the MFs that describe the input features or linguistic variables, the MFs that describe the output linguistic variables and the rules of the FLS. In this step, we train all these parameters, which are encoded into the individual as real numbered values as follows.

$$\phi_{FAET1} = M_1, \dots, M_i, \dots, M_{i+k}, R_1^e, \dots, R_l^e, R_1^d, \dots, R_m^d \quad (6.28)$$

Where M_i represents the membership functions for the i input features or linguistic variables of the FLS and M_{i+k} represents the MFs for the k^{th} output or consequent of the FLS using (6.2), (6.5) or (6.8). R_l^e and R_m^d represents the l and m rules of the encoder and decoder, respectively using (6.9).

For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FAE is performed using the following steps.

Step 1: N individuals are initialised by randomly generating values for each of the parameters of ϕ_{FAET1} in (6.28).

Step 2: The N individuals are then decoded into FAEs using (6.28) , and the fitness of these individuals is calculated using the cost function in (6.27).

Step 3: The best individual among these N FAEs is selected, and the stopping criteria are checked against this solution. If this FAE satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this FAE.

Step 4: Else, a new generation of N individuals is generated by mutating this individual using (4.2) , and the steps from Step 2 are repeated.

6.2.2.2 Transform TIMFs to IT2MFs

In the second step, we train the FOU of the MFs of the antecedents and the consequents using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). To do this, we add a FOU to the representation of the MFs of the antecedents and the consequents of the FAE trained in

the previous step. The representation of the FAE in (6.28) is modified using (6.13), (6.16) or (6.19) (based on the type of MFs) as follows.

$$T_e = T2_1, \dots, T2_i, \dots, T2_{i+k}, R_1^e, \dots, R_l^e, R_1^d, \dots, R_m^d \quad (6.29)$$

Where $T2_i$ represents the membership functions for the i input features or linguistic variables FLS and $T2_{i+k}$ represents the MFs for the k^{th} output or consequent of the FLS using (6.13), (6.16) or (6.19). Since we only train the parameters of the MFs and their FOU in this step, the parameters of the MFs are encoded into an individual as follows.

$$\phi_{FAEIT2} = T2_1, \dots, T2_i, \dots, T2_{i+k} \quad (6.30)$$

The FOU of the FAE are then trained. For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FOU of the FAE is performed using the following steps.

Step 1: $N-1$ individuals are generated by randomly generating values within the search space for each of the parameters in ϕ_{FAEIT2} . The final individual is generated by choosing the parameters from the type-1 FAE, trained in the previous step of the training (the FOU of this individual are set to zero).

Step 2: N individuals are then decoded into the MFs of the FAEs using (6.30), the rules are then added to these FAEs by choosing them from the type-1 FAE trained in the previous step.

Step 3: The fitness of the N FAEs are calculated using the cost function (6.27). The best individual among these N individuals is selected, and the stopping criteria are checked against this individual. If this FAE satisfies the stopping criteria, then the

optimization algorithm is stopped, and further steps of the training process are performed on this FAE.

Step 4: Else, a new generation of N individuals are generated by mutating this solution using (4.2) and the steps from Step 2 are repeated.

6.2.2.3 Optimizing the Rule Base of the IT2 FAE

In the third step, we retrain the rules of the FAE generated in the previous section. The rules of the FAE are encoded using the representation described in Section 6.1.1.2 using (6.9) as follows.

$$\phi_{FAERules} = R_1^e, \dots, R_l^e, R_1^d, \dots, R_m^d \quad (6.31)$$

Where R_l^e and R_m^d represent the rules (encoded using (6.9)) of the encoder and decoder, respectively.

The rules of the FAE are then retrained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the retraining of the rules is performed using the following steps.

Step 1: One individual is generated by encoding the rules of the FAE generated in the previous step into a real-valued solution using (6.31). Along with this, $N-1$ individuals are generated by mutating the first individual.

Step 2: The N individuals are then decoded into the rules of the FAE using (6.31). The MFs and their FOU's generated in the previous section are then added to the FAE.

Step 3: The fitness of these N FAEs are calculated using the cost function in (6.27). The best solution among these N FAEs is selected, and the stopping criteria are checked against this solution. If this FAE satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this FAE

Step 4: Else, a new generation of N Individuals is generated by mutating this solution using (4.2) , and the steps from Step 2 are repeated.

6.2.3 Optimization Method for the Final Layer

To train the full D2FLS, as depicted in Figure 6-2, we start by stacking the encoders of the n FAEs trained in the previous phase. We then add another FLS, which will act as the final output layer of the D2FLS. The output of the D2FLS can then be represented as follows.

$$y = f(e^n(e^{n-1}(\dots(e^1(x_p)\dots))) \quad (6.32)$$

Where e^n represent the encoder of the n^{th} FAE, f represents the final output layer and x_p is the input vector.

We use an optimization algorithm to retrain all the layers using the three-step training process depicted in Figure 6-10. The goal of the optimization algorithm is to minimise a cost function such as MAE (6.24), which is modified as follows:

$$MAE = \frac{\sum_{i=1}^p |y - \hat{y}|}{p} \quad (6.33)$$

Where \hat{y} is the predicted output of the D2FLS from equation (6.32), y is the actual output from the training dataset, n is the number of instances in the training dataset.

6.2.3.1 Optimize the Type 1 D2FLS

In this step, we train the final layer as a Type-1 FLS while at the same time retrain the MFs and rules of the encoders. First, the parameters of the encoders of the n FAEs trained in the previous phase are encoded using (6.20) and the MFs and rules of the final layer FLS are added to these (depicted in Figure 6-2) and encoded in the following format.

$$\phi_{D2FLS} = T_e^1, \dots, T_e^n, M_1^f, \dots, M_{o+p}^f, R_1^f, \dots, R_g^f \quad (6.34)$$

Where T_e^n represents the membership functions, and the Rules of the n^{th} encoders created using (6.20), M_{o+p}^f represents the membership functions for the o input features, p consequents of the final layer and R_g^f represents the g rules of the final layer.

The D2FLS is then trained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the D2FLS is performed using the following steps.

Step 1: N individuals are generated, the initial values of the three parameters of the final layer of the D2FLS in (6.34) are randomly generated as real numbered values. These values are then added to the parameters of the encoders generated in the previous phase.

Step 2: The N individuals are then decoded into D2FLSs using (6.34), and the fitness of these individuals is calculated using the cost function in (6.33) and the best solution among these N D2FLS is selected.

Step 3: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this individual satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS.

Step 4: If the stopping criteria (number of generations and target fitness) are not satisfied, a new generation of N individuals are generated by mutating the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

6.2.3.2 Transform the TIMFs of the D2FLS into IT2MFs

In this step, we transform the type-1 MFs of the final layer into interval type-2 MFs by adding a FOU to each of the fuzzy sets. This is similar to the way we added the FOUs while training the FAEs, and the FOUs are depicted in Figure 6-7 or Figure 6-8 or Figure 6-9 based on the type of MF. We also retrain the FOUs of the encoder created during the training of the FAEs. The FOUs are added to the MFs of the final layer, and these parameters are encoded in the following format.

$$\phi_{D2FLSIT2} = T_e^1, \dots, T_e^n, T2_1^f, \dots, T2_{o+p}^f \quad (6.35)$$

Where T_e^n represents the IT2 representation of the MFs of the n encoders from (6.30) and $T2_{o+p}^f$ represents IT2 MFs for the o antecedents and p consequents of the final layer using (6.13) or (6.16) depending on the type of MF used.

The FOU's of the D2FLS are then trained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FOU's of the D2FLS is performed using the following steps.

Step 1: N individuals are generated by randomly generating the FOU's of the antecedents and the consequents of the final layer of the D2FLS and added to the D2FLS generated in the previous section and their parameters encoded using (6.35).

Step 2: The real-valued representation of the N individuals is then decoded into MFs of the D2FLS using (6.35). The rules are then added to these D2FLS by choosing them from the type-1 D2FLS trained in the previous step.

Step 3: The fitness of the N D2FLSs are calculated using the cost function in (6.33) and the best D2FLS among these is selected

Step 4: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this D2FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS

Step 5: If the stopping criteria (number of generations and target fitness) are not satisfied a new generation of the N candidate solutions are generated by mutating real-valued representation of the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

6.2.3.3 Optimizing the rule base for the D2FLS

In the final step, we retrain the rules of all the layers. The parameters for this step are encoded in the following format to create the candidate solutions.

$$\phi_{D2FLSRules} = R_1^{e1}, \dots, R_l^{e1}, \dots, R_l^{en}, R_1^f, \dots, R_g^f \quad (6.36)$$

Where R_l^e and R_g^f represent the l rules of the encoder and g rules of the final layer, respectively.

The rules of the D2FLS are then retrained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the retraining of the rules of the D2FLS is performed using the following steps.

Step 1: The rules of the D2FLS generated in the previous step are encoded into real-valued solution using (6.36). And N individuals are generated by mutating this solution.

Step 2: The N individuals are then decoded into the rules D2FLS, then the MFs and their FOU's generated in the previous section are added to the D2FLS. The fitness of these individuals is calculated using the cost function in (6.33), and the best individual among these is selected.

Step 3: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this D2FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS.

Step 4: If the stopping criteria (number of generations and target fitness) are not satisfied a new generation of the N individuals are generated by mutating the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

6.3 Experiments and Results

To test the proposed Deep Type-2 Fuzzy logic system, we use the datasets in Table 5-1 and compare the performance of our proposed system against the following AI models Stacked Autoencoder (SAE), Convolutional Neural network (CNN), Multi-Layer Perceptron (MLP) and Interval Type-2 FLS (IT2FLS).

Next, we aim to compare the various types of Membership function and choose the best MF type. We will also compare the BB-BC algorithm against the GA to evaluate the differences when using the two algorithms and the best algorithm of the two will then be used in the experiments.”

6.3.1 Training Parameters

For the experiments, we used the following training parameters for the Categorical datasets, and we used Average Recall (6.21) as the fitness function

1. SAE was trained using greedy layer-wise training [61]. We used two hidden layers with 400 and 30 neurons each. Adam Algorithm [101] was used for training the SAE, and we set the learning rate as 0.001, beta one as 0.9 and beta two as 0.999 and trained it for 200 epochs.
2. CNN was trained using the Adam algorithm [101]. We created a custom VGG net [114] with six convolutional layers with 32 filters and kernel size 3 in the first two layers, 64 filters in the next two and 128 filters in the last two layers. We added a max-pooling layer after every two convolutional layers. These were

then connected to the output layer. We used Dropout [115] to reduce the chance of overfitting. And we used the same parameters as SAE for the Adam algorithm.

3. We created a Multi-layer perceptron with one hidden layer with 65 neurons in the hidden layer and an output layer. And we used Adam Algorithm for training the MLP with the same parameters as the Sparse Stacked Autoencoder.
4. IT2FLS was trained using the three-step training method proposed for training the FAE. We used 200 rules and three antecedents per rule. We used BB-BC algorithm with 500 generations and 30 candidates per generation.
5. D2FLS was trained using the proposed layer-wise training method. With two layers and with each layer having 100 rules and three antecedent per rule. The hidden layer was trained with 50 consequents. We used BB-BC algorithm with 500 generations and 30 candidates per generation.

Similarly, for the regression datasets, we used the following training parameters, and we used Mean Absolute Error (MAE) as the fitness function (6.24).

1. SAE was trained using greedy layer-wise training [61]. We used two hidden layers with 100 and 30 neurons each. Adam Algorithm [101] was used for training the SAE, and we set the learning rate as 0.001, beta one as 0.9 and beta two as 0.999 and trained it for 500 epochs.
2. CNN was trained using the Adam algorithm [101]. We created a CNN with four layers: the first layer is a convolutional layer with 100 filters and a kernel size of 5, The second layer is a Max pooling layer with a pool size of 3, The third layer is a fully-connected layer with 16 neurons and finally the output layer. We

used Dropout [115] to reduce the chance of overfitting. And we used the same parameters as SAE for the Adam algorithm.

3. We created a Multi-layer perceptron with one hidden layer with 65 neurons in the hidden layer and an output layer. And we used Adam Algorithm for training the MLP with the same parameters as the Sparse Stacked Autoencoder.
4. IT2FLS was trained using the three-step training method proposed for training the FAE. We used 200 rules and three antecedents per rule. We used BB-BC algorithm with 500 generations and 30 candidates per generation.
5. D2FLS was trained using the proposed layer-wise training method. With two layers and each layer is trained with 100 rules and three antecedent per rule. The hidden layer was trained with 30 consequents. We used BB-BC algorithm with 500 generations and 30 candidates per generation.

Table 6-1: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Categorical Datasets with Average Recall as Fitness function

Data Set	AI Model	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Standard Deviation
Santander CTP	D2FLS	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	SAE	65	67.7	66.02	62.34	62.77	64.77	2.01
	CNN	67.56	67.31	69.84	67.48	69.99	68.44	1.21
CLL Identification	D2FLS	62.8	59.78	61.25	62.37	63.31	61.903	1.26
	SAE	57.21	56.09	58.42	57.65	57.02	57.28	0.76
	CNN	69	68.5	67.9	69.24	68.92	68.71	0.47
BT Customer Data	D2FLS	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	SAE	75.95	75.06	76.51	75.84	72.03	75.08	1.59
	CNN	73.91	75.85	83.25	76.21	75.8	77.004	3.22
PD Speech	D2FLS	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	SAE	70.13	66.56	68.2	67.75	64.48	67.43	1.87
	CNN	77.08	77.38	76.69	77.38	78.08	77.32	0.46
IDA2016	D2FLS	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	SAE	87.45	88.54	86.72	87.81	84.84	87.07	1.26
	CNN	80.29	83.36	84.37	78.55	80.76	81.47	2.12
EpiSeizure	D2FLS	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	SAE	90.6	91.22	89.8	89.48	90.32	90.29	0.61
	CNN	94.03	92.68	92.48	90.73	89.03	91.79	1.73

6.3.2 Deep Type 2 Fuzzy Logic system Vs Deep Neural Networks

The aim of these experiments is to compare the performance of the D2FLS against two Deep Neural Networks, a Stacked Autoencoder and a Convolutional neural network. The goal is to see if our D2FLS model performs reasonably well when compared to the state-of-the-art Deep Neural Networks.

We tabulate the performance of the D2FLS, SAE and CNN over five training runs on the Categorical datasets in Table 6-1. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the AI model with the best performance for the dataset.

From Table 6-1, we can see that the CNN provided the best performance in four of the six datasets and the D2FLS and SAE provide the best performance in one each of

Table 6-2: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Regression Datasets using Mean Absolute Error as the Fitness Function

Data Set	AI Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	SAE	0.0558	0.0559	0.0479	0.0401	0.0463	0.0492	0.006
	CNN	0.0494	0.037	0.0477	0.0422	0.0435	0.044	0.0044
Swiss Premium Prediction	D2FLS	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	SAE	0.0277	0.0258	0.0274	0.0294	0.0269	0.0275	0.0012
	CNN	0.0237	0.0259	0.0255	0.0271	0.0263	0.0257	0.0011
CT Scan Region Prediction	D2FLS	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	SAE	0.031	0.0374	0.0298	0.0345	0.0298	0.0325	0.003
	CNN	0.045	0.045	0.0454	0.0451	0.0458	0.0453	0.0003
Predict Song Year	D2FLS	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	SAE	0.072	0.072	0.0663	0.0658	0.066	0.0684	0.0029
	CNN	0.0745	0.0752	0.0745	0.0689	0.0691	0.0724	0.0028
BT PWA	D2FLS	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	SAE	0.0384	0.0379	0.0374	0.038	0.038	0.0379	0.0003
	CNN	0.0381	0.0383	0.041	0.0374	0.0379	0.0385	0.0013

the selected datasets. This is intuitive considering that CNN is the state-of-the-art AI algorithm for a variety of classification problem. The relevant point here is that in four of the datasets (Santander CTP, BT Customer Data, EpiSeizure and PD Speech) the difference between the performance of the D2FLS and the best algorithm is within 4-5 per cent. The D2FLS performance is 2-3% lower than the CNN and 2-3% better than the performance of the SAE. This indicates that the performance of the D2FLS is comparable to the performance of the best DNN algorithms.

Similarly, Table 6-2 shows the results of five training runs with Mean Absolute Error as the fitness function (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. We have also highlighted in bold the model with the best average MAE for each of the datasets.

From Table 6-2, we can see that the SAE performs the best in three of the datasets, and CNN provides the best performance in the other two datasets. But the difference in

Table 6-3: Comparison of the performance of the D2FLS vs Stacked Autoencoder vs CNN in Regression Datasets using Root Mean Square Error as the Fitness Function

Data Set	AI Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS	0.1546	0.1346	0.1486	0.1362	0.1222	0.1392	0.0113
	SAE	0.0726	0.0553	0.0607	0.0674	0.0592	0.063	0.0062
	CNN	0.0644	0.0564	0.0584	0.0629	0.0671	0.0618	0.0039
Swiss Premium Prediction	D2FLS	0.0569	0.078	0.0681	0.0663	0.0708	0.068	0.0068
	SAE	0.0398	0.0367	0.0356	0.0396	0.0366	0.0377	0.0017
	CNN	0.0395	0.0413	0.0391	0.0394	0.0413	0.0401	0.001
CT Scan Region Prediction	D2FLS	0.1425	0.14	0.1262	0.126	0.1397	0.1349	0.0072
	SAE	0.0542	0.0577	0.0559	0.0576	0.0558	0.0562	0.0013
	CNN	0.0737	0.0737	0.0732	0.0732	0.0735	0.0734	0.0002
Predict Song Year	D2FLS	0.1095	0.1115	0.1088	0.1106	0.1094	0.11	0.001
	SAE	0.1005	0.1029	0.1011	0.1009	0.1007	0.1012	0.0009
	CNN	0.1044	0.1028	0.1028	0.1022	0.1039	0.1032	0.0008
BT PWA Data Test	D2FLS	0.0749	0.0796	0.0687	0.0743	0.0844	0.0764	0.0053
	SAE	0.0546	0.0538	0.0541	0.0544	0.0543	0.0542	0.0003
	CNN	0.0555	0.0559	0.0547	0.0543	0.0569	0.0554	0.0009

Table 6-4: Snapshot of Rule base of the Hidden Layer of the D2FLS on the BT PWA Dataset

ID	Antecedents			Consequents				
	1	2	3	H00	H01	H02	H03	H04
1	High MSLCL1 2	Mid CONTRACTOR 2	Low RANK 0	Low	High	Very Very Low	Mid	Very Low
2	High MISSAPP 2	Low RANK 0	Low ON DAY UTILISATION 3	Very Very Low	Low	High	Low	Very Very Low
3	Low CONTRACTOR 1	High MSLCL2 3	Mid ECONOMIC UTILISATION 2	Very Low	High	Very Very Low	Low	High
4	High ON DAY UTILISATION 2	Low ECONOMIC UTILISATION 1	Low MSLCL2 0	Very Very Low	Very Very Low	Mid	High	High
5	Low MSLCL2 1	Low ON_DAY UTILISATION 1	Low ECONOMIC UTILISATION 3	Low	Very Very Low	Mid	Mid	Low
6	Low MSLCL2 3	High TRAVEL 0	Mid MISSAPP 0	High	Low	High	Very Low	High
7	Mid CALC PROD 1	Mid MSLCL2 2	Mid RANK 0	Low	High	High	Low	High
8	Mid MSLCL2 0	Mid MSLCL2 3	Low LOANS 2	Low	High	Very Very Low	Very Low	Very Very Low

performance between the SAE and CNN is small. The D2FLS has a higher error on average when compared to the SAE and the CNN for the regression datasets.

To make sure that the Mean Absolute Error fitness function is not affecting the results, we conducted another experiment on the Regression datasets by retraining all the data models using a different fitness function. In this case, the Root Mean Squared Error (RMSE) represented below in equation (6.37).

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (6.37)$$

Where y_i is the desired output, \hat{y}_i is the actual output of the model and n is the number of inputs.

The result of five training runs using RMSE as the fitness function are tabulated in Table 6-3. With mean and standard deviation over these training runs in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the AI model with the best performance for the dataset.

Table 6-5: Snapshot of Rule base of the Output Layer of the D2FLS on the BT PWA Dataset

ID	Antecedents			Consequent
	1	2	3	PWA Performance
1	Low H02	Mid H00	High H03	Very Very Low
2	High H00	Mid H03	Mid H01	Low
3	Low H00	High H01	Mid H02	High
4	Mid H03	High H02	Low H01	Very Very Low
5	Mid H02	High H00	Mid H03	Low
6	Low H03	Low H01	High H00	Very Very Low
7	High H02	High H01	Mid H00	High
8	High H01	Mid H02	High H03	High
9	Low H02	Low H00	Low H03	Very Low
10	Mid H00	High H02	Low H03	Very Very Low
11	High H02	Mid H03	Low H00	High
12	High H00	Mid H01	Low H03	Low
13	High H00	Low H02	High H03	Very Low
14	Low H00	Low H03	Mid H01	Very Low
15	Low H03	Mid H00	Mid H01	High

From Table 6-3, we can see that the SAE provides the best performance in four of the datasets, and CNN has the best performance in one of the datasets. This is similar to the behaviour seen when using the MAE as the fitness function; the only difference is in the Swiss Premium Prediction dataset where the CNN outperformed by the SAE. We can also see that the D2FLS has a higher error on average across the five datasets, which is very similar to the performance loss when using MAE as the fitness function. This means that changing the fitness function has only a small effect on the relative performance of the three models.

A snapshot of the rule base generated by one of the runs on the BT PWA dataset is shown in Table 6-4 and Table 6-5. Table 6-4 contains a snapshot of the rules (8 out of 100 rules) of the Hidden Layer of the D2FLS, it shows that the rules are short and comprise of three antecedents and five consequents that form the five outputs of the hidden layer (H00, H01, H02, H03, H04). Membership functions for the first two of

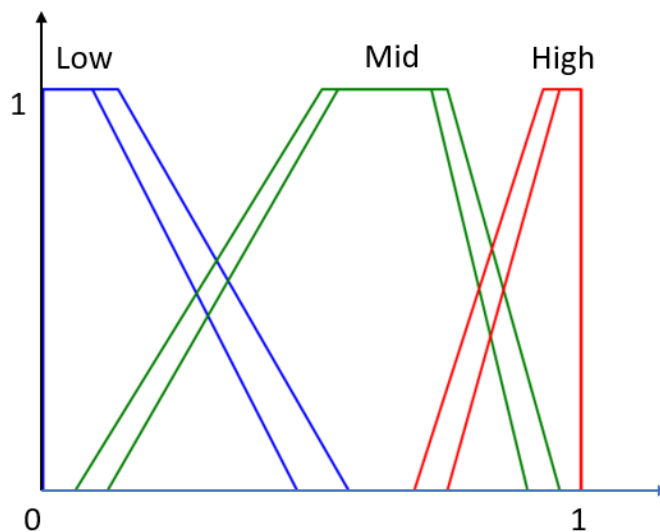


Figure 6-11: Fuzzy Set Generated by D2FLS Training for MSLCL1 2 feature of the BT PWA dataset

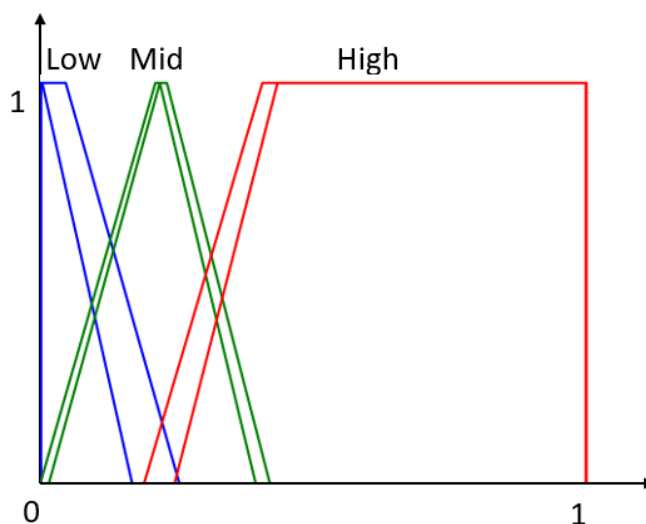


Figure 6-12: Fuzzy Set Generated by D2FLS Training for Contractor 2 feature of the BT PWA dataset

the antecedents of the first rule in Table 6-4 are depicted in Figure 6-11 and Figure 6-12. Table 6-5 contains a snapshot of the rules of the output layer of the D2FLS, and the rules of the output layers are also short and comprise of three antecedents and one consequent for the output. The inputs to the output layer (H00, H01, H02, H03, H04) of the D2FLS are synthetic variables created during the training.

The rules of both the layers are kept short (3 antecedents per rule) to maximise the interpretability of the model while also maintaining the accuracy of the predictions. We can then use these rules and membership functions to interpret the predictions of the D2FLS. Compared to the SAE and CNN, which are difficult to interpret using just the weight and biases used to define them. External tools and modifications such as the methods described in Chapter 2.1 and Chapter 2.3 are required to interpret the SAE and CNN. But these methods have limitations such as difficulty in interpreting the explanations in case of inputs with a large number of features. Hence, in cases where the interpretability of the AI model is essential, we can choose D2FLS over SAE or CNN without losing too much of predictive accuracy of the AI model.

6.3.3 D2FLS vs Shallow Neural Networks and an IT2FLS

In this experiment, we compare the performance of the D2FLS against a Multilayer perceptron and an Interval Type-2 Fuzzy Logic System. The goal is to see if our D2FLS model performs reasonably well when compared to these two shallow AI models.

Table 6-6, contains the results of five training runs on classification datasets. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold (column 8), it indicates that the row contains the AI model with the best performance for the dataset.

Table 6-6 shows that the D2FLS provided the best performance in three of the four datasets, and the MLP provides the best performance in two of the remaining datasets. The D2FLS outperforms the IT2FLS in all the datasets, and the performance improvement is more than 4% in three of the datasets and about 2% in the other two

Table 6-6: Comparison of the performance of the D2FLS vs Multi-layer perceptron vs IT2FLS in Classification Datasets using Average Recall as the Fitness Function

Data Set	AI Model	1	2	3	4	5	Average	Standard Deviation
Santander CTP	D2FLS	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	MLP	65.3	62.41	59.88	64.1	65.09	63.36	2.02
	IT2FLS	56.92	59.44	58.1	56.9	58.14	57.9	0.94
BT Customer Data	D2FLS	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	MLP	75.63	74.34	73.44	75.24	73.05	74.34	0.99
	IT2FLS	60.23	61	60.62	60.66	59.88	60.48	0.38
PD Speech	D2FLS	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	MLP	66.37	66.87	61.81	60.12	63	63.63	2.61
	IT2FLS	69.41	63.6	71.85	61.83	68.11	66.96	3.71
IDA2016	D2FLS	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	MLP	83.32	85.23	82.97	81.48	82.59	83.12	1.22
	IT2FLS	89.64	91.54	90.05	91.33	89.84	90.48	0.79
EpiSeizure	D2FLS	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	MLP	84.53	80.31	82.16	79.97	83.76	82.14	1.81
	IT2FLS	89.73	88.76	88.54	90.67	89.11	89.36	0.77

datasets with an average improvement of about 5%. The D2FLS outperforms the MLP in three of the datasets with about 9% improvement in performance. In the other two datasets where the MLP outperforms the D2FLS, the performance difference is only about 2%. The D2FLS performs better than the MLP overall with about 5% increased performance.

Table 6-7 displays the results of the five training runs on the regression datasets. We display the MAE over the testing data on the regression dataset in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold (column 8), it indicates that the row contains the AI model with the best performance for the dataset.

Table 6-7 shows that the MLP provides the best performance, and the D2FLS has the next best fitness when compared to the MLP. The D2FLS outperforms the IT2FLS in all four of the selected datasets. The D2FLS has at least 6% lower error rate when

Table 6-7: Comparison of the performance of the D2FLS vs Multi-layer perceptron vs IT2FLS in Regression Datasets using MAE as the Fitness Function

Data Set	AI Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	MLP	0.0387	0.0367	0.042	0.0389	0.0377	0.044	0.0044
	IT2FLS	0.129	0.1551	0.116	0.1188	0.1655	0.1369	0.0199
Swiss Premium Prediction	D2FLS	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	MLP	0.0262	0.0258	0.0263	0.0265	0.0265	0.0257	0.0011
	IT2FLS	0.0507	0.0512	0.0582	0.0518	0.0515	0.0527	0.0028
CT Scan Region Prediction	D2FLS	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	MLP	0.0353	0.0352	0.0359	0.0355	0.0333	0.0453	0.0003
	IT2FLS	0.1048	0.0935	0.0904	0.088	0.1012	0.0956	0.0064
Predict Song Year	D2FLS	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	MLP	0.0749	0.0753	0.0691	0.0673	0.0671	0.0724	0.0028
	IT2FLS	0.08	0.086	0.081	0.0793	0.0831	0.0819	0.0024

compared to the IT2FLS with the highest improvement in fitness seen in the Wi-Fi Localization dataset with a 30% lower error. And the D2FLS has 15% lower error on average when compared to the IT2FLS.

6.3.4 Comparison between the Three-Step Training process and Single Step Training process

Table 6-8: Comparison for the Three-Step Training Process and the Single Step Training Process for training the D2FLS with Mean Absolute Error as the Fitness Function

Data Set	Training Method	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Std
Wi-Fi Localization	TS Training	0.106	0.116	0.1049	0.1022	0.098	0.105	0.00589
	SS Training	0.147	0.148	0.149	0.178	0.154	0.155	0.0115
Swiss Premium Prediction	TS Training	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	SS Training	0.067	0.082	0.065	0.069	0.081	0.0729	0.0072
CT Scan Region Prediction	TS Training	0.095	0.091	0.089	0.084	0.09	0.0897	0.00334
	SS Training	0.117	0.136	0.13	0.126	0.126	0.127	0.0062
BT PWA	TS Training	0.048	0.057	0.051	0.045	0.057	0.0519	0.00474
	SS Training	0.078	0.05	0.106	0.063	0.077	0.0748	0.0187

The aim of this experiment is to test the effectiveness of the three-step training process (TS Training), depicted in Figure 6-10, proposed as the training method for training each layer of the D2FLS. We compare this training method to an alternative by combining all the steps of the TS training and train each layer in a single step (SS Training) by encoding the layer using the representation in (6.20) and training the MFs and FOU's of the antecedents and consequents along with the rules in a single step.

For the TS training process, we use BB-BC as the training algorithm with 500 generations and 30 particles per step. For the SS training process, we again use the BB-BC algorithm with 500 generations and 90 particles to train the D2FLS. We use these parameters for the optimization algorithm to ensure that the two training methods use a similar amount of CPU time.

The result of training the two methods over five training runs on four regression datasets is tabulated in Table 6-8. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns 3-7. The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold, it indicates that the row contains the training method with the best performance for the dataset.

The results presented in Table 6-8 show that the D2FLS trained using the TS training method performs better than the D2FLS trained using SS training method in all four datasets. Using the TS training method for the four datasets reduces the MAE by 47% on average across the four datasets. This supports the choice of the TS Training method, and it is utilised in all the other experiments.

6.3.5 Comparison Between D2FLS trained using the various Membership Function Types

The aim of this experiment is to test the impact of the various types of MF on the proposed D2FLS model. The MF type, which provides the best results, is then used in the rest of the experiments.

There are mainly three types of Membership functions that were tested; they are Triangular MF, Trapezoidal MF and Gaussian MF (depicted in Figure 6-7, Figure 6-8 and Figure 6-9). Figure 6-13 depicts, examples of the three types of IT2 MFs generated during training. Average Recall and Mean absolute error are used as the fitness functions the categorical and regression datasets, respectively. In all the cases, the BB-BC algorithm is used as the optimization algorithm with 500 generations and 30 particles per step.

Table 6-9: Comparison of performance of D2FLS for difference types of MFs on Categorical Datasets with Average Recall as the Fitness Function

Data Set	Type of MF	1	2	3	4	5	Average	Standard Deviation
Santander CTP	Trapezoidal	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	Triangular	61.07	62.6	61.09	61.23	63.05	61.807	0.84
	Gaussian	52.59	53.79	53.48	53.24	52.89	53.197	0.42
BT Customer Data	Trapezoidal	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	Triangular	68.24	71.87	70.99	70.56	71.98	70.73	1.35
	Gaussian	59.66	59.86	60.02	60.08	59.8	59.885	0.15
PD Speech	Trapezoidal	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	Triangular	58.94	68.17	65.74	67.99	70.43	66.253	3.95
	Gaussian	56.98	60.37	57.29	57.46	58.42	58.103	1.23
IDA2016	Trapezoidal	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	Triangular	91.73	89.17	92.98	94.4	92.78	92.213	1.58
	Gaussian	94.79	92.34	91.73	92.95	87.93	92.954	1.32
EpiSeizure	Trapezoidal	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	Triangular	87.68	87.32	80.7	86.004	88.22	85.983	2.74
	Gaussian	80.16	79.43	75.13	80	78.85	78.714	1.85

Table 6-10: Comparison of performance of D2FLS for difference types of MFs on Regression Datasets with Mean Absolute Error as the Fitness Function

Data Set	Type of MF	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Std
Wi-Fi Localization	Trapezoidal	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	Triangular	0.1105	0.1256	0.1486	0.1525	0.1477	0.1369	0.0163
	Gaussian	0.2345	0.2393	0.2366	0.2474	0.2418	0.2399	0.0045
Swiss Premium Prediction	Trapezoidal	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	Triangular	0.0596	0.065	0.0659	0.0581	0.0543	0.0606	0.0043
	Gaussian	0.1499	0.1499	0.1641	0.1579	0.1499	0.1543	0.0058
CT Scan Region Prediction	Trapezoidal	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	Triangular	0.1101	0.1136	0.1184	0.1217	0.1361	0.12	0.009
	Gaussian	0.122	0.1335	0.188	0.1359	0.1036	0.1366	0.0281
Predict Song Year	Trapezoidal	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	Triangular	0.0797	0.0785	0.08	0.0752	0.0751	0.0777	0.0021
	Gaussian	0.0987	0.0882	0.1553	0.0859	0.0903	0.1037	0.0262
BT PWA	Trapezoidal	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	Triangular	0.0642	0.0551	0.0556	0.0496	0.0625	0.0574	0.0053
	Gaussian	0.0776	0.0856	0.0746	0.0788	0.0729	0.0779	0.0044

We tabulate the performance of the D2FLS when using the three types of MFs over five training runs on the Categorical datasets in Table 6-9. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold, it indicates that the row contains the results for the Membership functions type with the best performance for the dataset.

We can see from Table 6-9 that the Trapezoidal MFs provide the best performance in four of the five datasets while the Gaussian MFs perform the best in one of the datasets (IDA 2016). The Triangular MFs provide slightly worse performance than the Trapezoidal MFs, about 3% loss in performance on average. While the Gaussian MFs perform the worst in four of the five datasets, about 10% loss in performance on average when compared to Trapezoidal MFs.

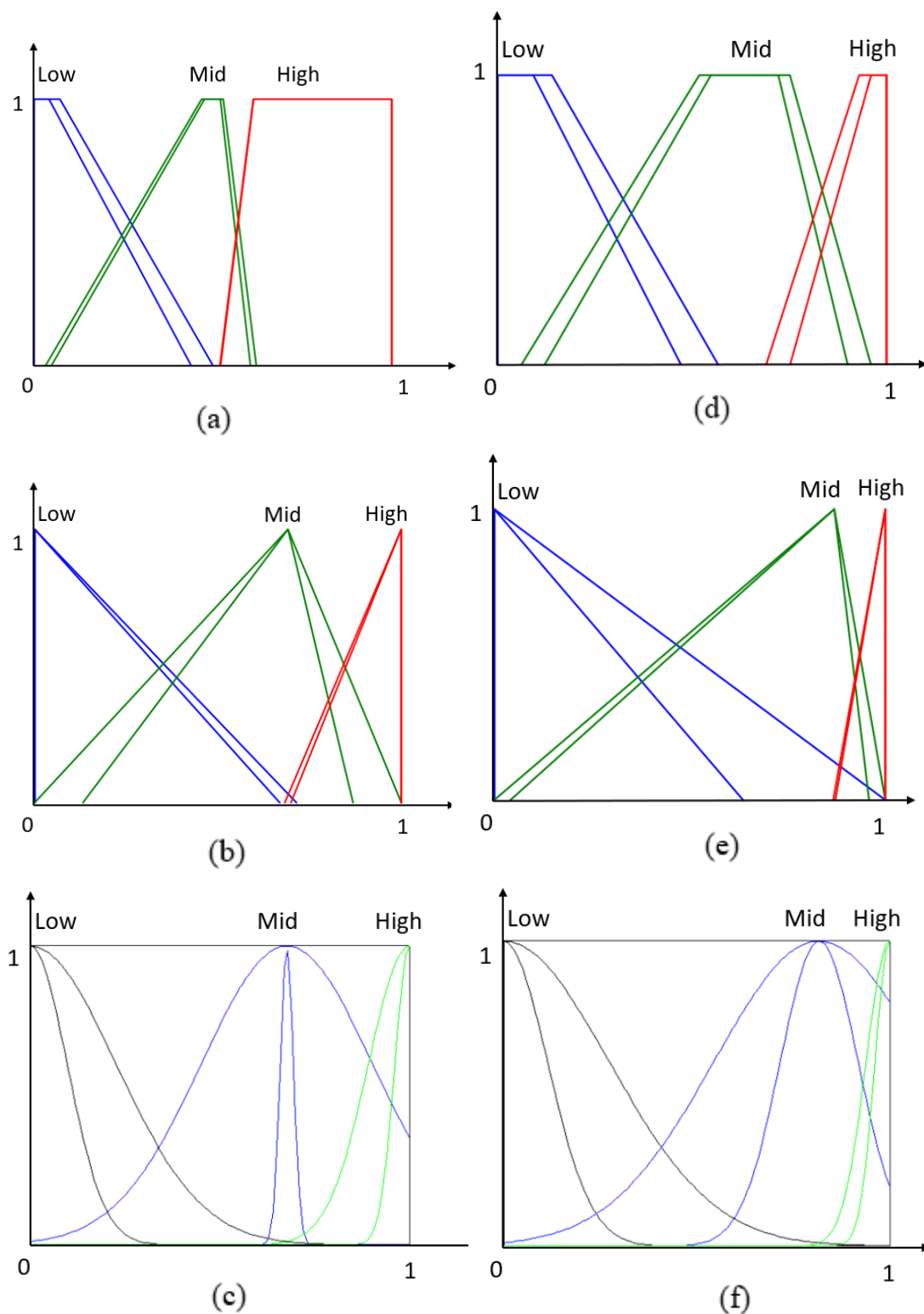


Figure 6-13: Examples of IT2 Fuzzy Sets Generated for D2FLS during Training where (a) (d) are Trapezoidal MF, (b) (e) are Triangular MF, and (c) (f) are Gaussian MF for Contractor 0 and MSLCL1 2 features respectively of the BT PWA dataset

Similarly, we tabulate the performance of the D2FLS for the Regression datasets in Table 6-10. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold, it indicates that the row contains the results for the Membership functions type with the best performance for the dataset.

From Table 6-10, we can see that the Trapezoidal MFs provide the best performance for all the regression datasets. The use to Triangular MFs and Gaussian MFs led to about 21% and 79% larger rate of error, respectively. Examples of the membership functions generated are depicted in Figure 6-13.

This result indicates that the Trapezoidal MFs is the best MF to use in D2FLS for the selected datasets. This might be because Trapezoidal MFs generally provide higher degrees of freedom compared to the Triangular and Gaussian MFs [75]. Hence, we use Trapezoidal MFs for the rest of the experiments.

6.3.6 Comparison of the performance of the BB-BC against Genetic Algorithms

Table 6-11: Comparison Between D2FLS Trained using BB-BC and Genetic Algorithm on Categorical Datasets with Average Recall as the Fitness Function

Data Set	Optimization Method	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Std
Santander CTP	BB-BC	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	GA	54.83	54.34	55.38	55.32	55.27	55.029	0.39
BT Customer Data	BB-BC	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	GA	67.46	67.39	73.48	72.64	74.35	71.062	3.02
PD Speech	BB-BC	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	GA	61.59	61.53	60.5	55.78	64.68	60.816	2.88
IDA2016	BB-BC	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	GA	89.29	90.98	90.28	92.3	89.13	90.394	1.17
EpiSeizure	BB-BC	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	GA	82.68	83.19	82.84	82.8	84.43	83.19	0.64

Table 6-12: Comparison Between D2FLS Trained Using BB-BC and Genetic Algorithm on Regression Datasets with Mean Absolute Error as the Fitness Function

Data Set	Optimization Method	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Std
Wi-Fi Localization	BB-BC	0.106	0.116	0.1049	0.1022	0.098	0.105	0.0058
	GA	0.2743	0.2221	0.2176	0.1837	0.1815	0.2158	0.0337
Swiss Premium Pred	BB-BC	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004
	GA	0.1406	0.1373	0.1365	0.1355	0.1273	0.1354	0.0044
CT Scan Region Pred	BB-BC	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003
	GA	0.171	0.1772	0.1577	0.1676	0.1565	0.166	0.0079
Predict Song Year	BB-BC	0.076	0.0741	0.074	0.074	0.079	0.075	0.002
	GA	0.1085	0.1144	0.1136	0.0853	0.0963	0.1036	0.0112
BT PWA	BB-BC	0.048	0.057	0.0511	0.045	0.057	0.0519	0.005
	GA	0.1086	0.1014	0.1391	0.102	0.1199	0.1142	0.0141

The aim of these experiments is to evaluate the differences when using the BB-BC algorithm or Genetic Algorithm (GA) for training the D2FLS. The best algorithm can then be used for further experiments.

Average Recall is used as the fitness function for all categorical datasets and Mean absolute error is used for all the regression datasets. In all the cases where BB-BC algorithm is used as the optimization algorithm, its parameters are 500 generations and 30 particles per step. Similarly, The GA was run for 150 generations, with 100 individuals per generation with a crossover probability of 0.4 and a mutation rate of 0.1. These values are chosen to ensure that both the BB-BC and GA are trained for the same amount of time.

We tabulate the performance of the D2FLS trained using the BB-BC and GA over five training runs on the Categorical datasets in Table 6-11. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold (column 8), it indicates that the row contains the optimization algorithm with the best performance for the dataset.

Table 6-13: Impact of number of Generations on D2FLS trained using Genetic Algorithm

Number of Generations	150	200
	0.108567	0.118009
	0.101369	0.138971
	0.139087	0.129255
	0.102	0.120627
	0.119919	0.130588
Average	0.114189	0.12749
Standard Dev.	0.014123	0.007504

We can see from Table 6-11 that the D2FLS trained using BB-BC algorithm perform significantly better than the GA in all five of the categorical datasets with a 6% higher average recall on average.

Similarly, we tabulate the performance of the D2FLS for the Regression datasets in Table 6-12. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold, it indicates that the row contains the results for the optimization algorithm with the best performance for the dataset.

From Table 6-12, we can see that the D2FLS trained using BB-BC algorithm perform significantly on all five of the Regression datasets with a 49% smaller error on average when compared to the GA.

We then investigated why the performance of the GA is lower than the BB-BC algorithm. First, we conducted an experiment to check if increasing the number of generations will improve performance. We ran the GA for 150 and 200 generations with 100 individuals in each generation on the BT PWA Data dataset. The results of this are tabulated in Table 6-13. From Table 6-13, we can see that running the GA for

Table 6-14: D2FLS optimized by Genetic Algorithm using a single step training with MAE as fitness function

Number of Generations	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Standard Dev
100	0.1819	0.1886	0.1773	0.1791	0.1816	0.1817	0.0038
200	0.1848	0.1671	0.168	0.1805	0.1713	0.1743	0.0071
300	0.1687	0.1782	0.181	0.1511	0.1691	0.1696	0.0105
400	0.1566	0.1544	0.1713	0.1482	0.1724	0.1606	0.0096
500	0.1396	0.1594	0.1491	0.1677	0.1482	0.1528	0.0098

longer does not improve the performance. This might be because of the multiple steps used during the training.

Hence, In the next experiment, we checked the effect of training the D2FLS in a single step. i.e., we trained the FOU along with the MFs and rules of the D2FLS at the same time. The results of this training over a variety of generations are tabulated in Table 6-14. As we can see, the performance of the D2FLS improves as we increase the number of generations. This indicates that increasing the number of generations will provide improved performance. But even after training for 500 generations, the performance of the Single Step training process is lower than the three-step training process we used for other experiments; the three-step process is run for 100 generations per step, i.e., 100 generations per step which gives us a total of 300 generations. Therefore, we stopped any further investigation as the number of generations required to get similar performance to the BB-BC from GA seems to be too high. Hence, we decided to use BB-BC algorithm for all the other experiment.

6.3.7 Deep Type-2 FLS vs Deep Type-1 FLS

In this experiment, we compare the performance of a Deep Type-2 FLS against a Type-1 version of the same model, i.e., the FOU of the D2FLS are not trained. The goal is to see how much the predictive accuracy of the D2FLS can be improved by adding Interval Type-2 fuzzy sets.

Table 6-15: Deep Type-2 FLS vs Deep Type-1 FLS on Classification Datasets

Data Set	Type of Fuzzy Set	1	2	3	4	5	Average	Std
Santander CTP	IT2	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	Type-1	57.74	59.66	60.49	62.36	59.58	59.97	1.5
CLL Identification	IT2	62.8	59.78	61.25	62.37	63.31	61.9	1.26
	Type-1	61.13	54.18	61.81	62.06	62.01	60.24	3.05
BT Customer Data	IT2	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	Type-1	59.8	59.98	62.07	61.36	58.31	60.3	1.31
PD Speech	IT2	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	Type-1	62.77	68.85	58.76	67.82	66.09	64.86	3.68
IDA2016	IT2	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	Type-1	92.63	91.37	91.99	93.14	91.87	92.2	0.62
EpiSeizure	IT2	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	Type-1	89.7	89.52	89.89	92.31	89.48	90.18	1.08

The D2FLS and its Type-1 counterpart are trained using BB-BC, with Average recall as the fitness function, five times. And the Average Recall on the testing data of the categorical datasets along tabulated in Table 6-15. The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold (column 8), it indicates that the row contains the Type of FLS with the best performance for the dataset.

From Table 6-15, we can see that the Deep Type-2 Fuzzy logic system outperforms its Type-1 counterpart in all the selected classification datasets. This is logical as the D2FLS is trained by adding FOU's to an optimised Type-1 system; hence, the D2FLS system will perform at least as well as its type-1 counterpart. The main point to note here is the magnitude of the performance improvement when the Type-1 version of the system is converted into the IT2 version. As we can see, there is almost 9 per cent improvement in the PD Speech dataset and 12 per cent improvement in the BT Customer Data dataset. With an average performance improvement of 4.3% by switching from Type-1 to Interval Type-2 system in Classification datasets.

Table 6-16: Deep Type-2 FLS vs Deep Type-1 FLS on Regression Datasets

Data Set	Type of Fuzzy set	1	2	3	4	5	Average	Std
Wi-Fi Localization	IT2	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	Type-1	0.119	0.1106	0.1343	0.1137	0.1366	0.1228	0.0107
Swiss Premium Pred	IT2	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	Type-1	0.1188	0.1361	0.1104	0.1156	0.147	0.1256	0.0138
CT Scan Region Pred	IT2	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	Type-1	0.1215	0.1258	0.1114	0.0958	0.1081	0.1125	0.0106
Predict Song Year	IT2	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	Type-1	0.0747	0.083	0.0784	0.078	0.0812	0.0791	0.0029
BT PWA	IT2	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	Type-1	0.0579	0.052	0.0581	0.0833	0.0631	0.0629	0.0108

Similarly, the D2FLS and its type-1 counterpart are trained using BB-BC with MAE as the fitness function five times each. The MAE on the testing data of the regression datasets is tabulated in Table 6-16. The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold (column 8), it indicates that the row contains the Type of FLS with the best performance for the dataset.

From Table 6-16, we can see that similar to the classification datasets the D2FLS outperforms its type-1 counterpart in all the selected regression datasets. The D2FLS provides about 47% lower error when compared to its Type-1 counterpart with the best improvement in the Swiss Premium Prediction dataset where it has less than half the error. This indicates that using Interval type-2 Fuzzy sets provides a significant improvement over type-1 fuzzy sets.

6.4 Summary

This chapter discussed the Deep Type-2 Fuzzy Logic System and a novel training method to train this AI model. It discussed the shortcomings of the IT2FLS system for

datasets with a large number of features and the suitability of the D2FLS for these datasets. To fully evaluate the proposed system, we compared it against SAE, CNN, MLP and IT2FLS.

The results show that in the classification datasets the proposed D2FLS provides 2% improvement when compared to the SAE on average and it was within 2% of the performance of the CNN. The results also show the D2FLS has about 4-5% higher performance compared to the MLP and the IT2FLS. This shows that the D2FLS is competitive against state-of-the-art Deep Learning algorithms in categorical datasets and outperforms the MLP and IT2 FLS in the categorical datasets.

In the regression datasets, the results show that the proposed D2FLS has shown higher error on average when compared to the MAE of the SAE, MLP and CNN, respectively. The D2FLS has better performance when compared to the IT2FLS (15% lower error).

The chapter also shows the interpretability of the D2FLS in the form of the membership functions and snapshots of the rules of the D2FLS. The interpretability of the D2FLS will be further evaluated through a survey in Chapter 8.3. The D2FLS comprises of a small number of rules with a small number of antecedents per rule, thus maximising the interpretability of the model. The SAE, CNN and MLP models, on the other hand, require external tools or modifications to provide explainability, but these tools have their own limitations and might not always be suitable.

This chapter also discusses the impact of using IT2 MFs in the D2FLS and the results show that the use of IT2 MFs has a significant effect on the performance of the D2FLS.

With about 5% improvement in performance in the categorical datasets and 47% lower error rate when compared to its Type-1 counterpart in regression datasets.

This chapter also explored the use of GA to train the D2FLS. The results show that the D2FLS trained using BB-BC has a 4.3% higher average recall in the categorical datasets and about 47% lower MAE in the regression datasets when compared to the D2FLS trained using a GA.

In the next chapter, we discuss an alternative training method for the D2FLS by extending the training method used to train a Fuzzy Stacked Autoencoder (FSAE).

Chapter 7. Deep Type-2 FLS trained using a Stacked Autoencoder

An alternative to the training method described in section 6.2 is to first train a stacked autoencoder and use the outputs at the intermediate layers of these autoencoders to pre-train the hidden layers of the D2FLS. A final output layer is then added to it, and the whole model is trained.

The inspiration for this training method comes from the Fuzzy Stacked Autoencoder (FSAE) [116], depicted in Figure 7-1, in which the final layer of a stacked autoencoder is replaced by an FLS. The way this is done is by first training the hidden layers of the FSAE as SAEs using a greedy layer-wise training algorithm [61]. Once the system is trained the last layer of the stacked autoencoder is replaced by a Fuzzy Logic System (T1/IT2 FLS) and trained using an optimization algorithm like BB-BC using the output of $n-1$ layer of the SAE as the input.

There is one advantage to using this method over the training method presented in the previous chapter. Since, we use gradient descent algorithms to train the stacked autoencoder, the training is much faster than random search-based algorithms such as BB-BC. The disadvantage is that this training method is much more complex than the

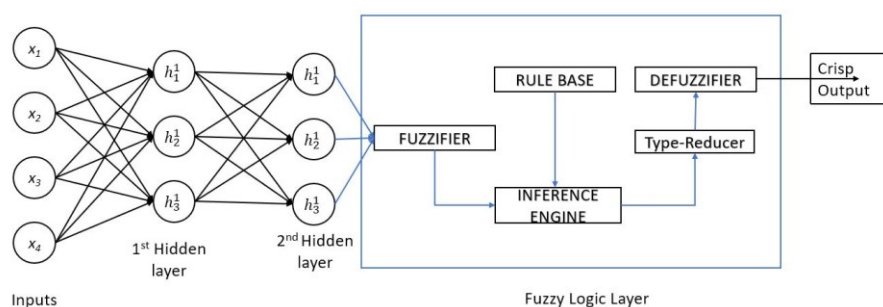


Figure 7-1: Fuzzy Stacked Autoencoder [116]

method presented in the previous chapter. This training method consists of three phases: the first phase is the training of the SAE, the second phase is pretraining of the hidden layers of the D2FLS and finally in the third phase the final or output layer is added to the D2FLS and all the hidden layers are retrained. The details of this training method are presented in the following sections.

7.1 Autoencoder Training

An autoencoder is a neural network that is designed to reconstruct an approximation of the input at the output, i.e., the target output of the network is the input. The idea here is to constrain the network in such a way that the autoencoder is forced to learn the important characteristics of the inputs. Some of the ways in which we will constrain

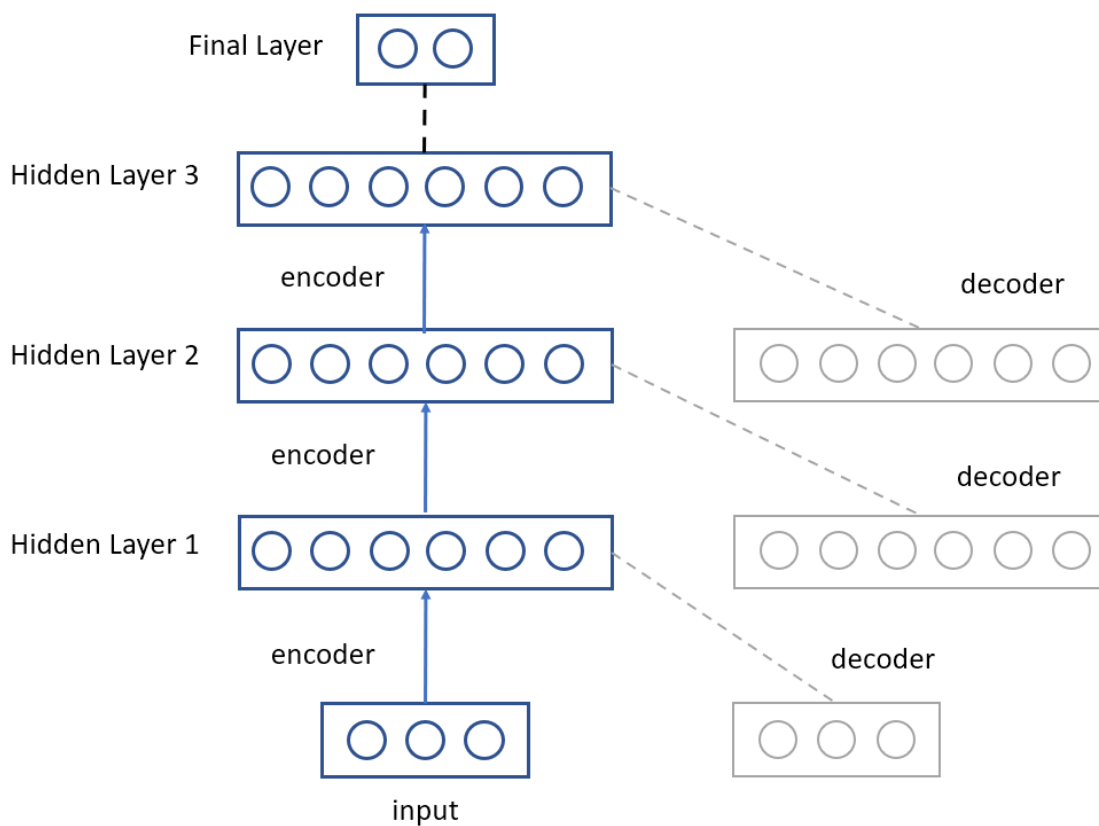


Figure 7-2: Stacked Autoencoder Training

the network is by restricting the number of neurons in the hidden layers which will force the network to learn a sparse representation, add noise to the inputs etc. These constraints are used to ensure that the autoencoder does not merely learn the identity function of the input.

Mathematically the autoencoder can be represented as follows, given a set of training samples $[x_1, x_2, \dots, x_p]$ where $x_p \in R$, the autoencoder first encodes the input x_p to a hidden representation h_k , then it decodes the hidden representation to \hat{x}_p as shown below.

$$h_k = f(W_1 x_p + b_1) \quad (7.1)$$

$$\hat{x}_p = g(W_2 h_k + b_2) \quad (7.2)$$

Where W_1 and b_1 represent the weights and bias of the encoder layer, W_2 and b_2 represent the weights and bias of the decoder layer. The weights and bias of the encoder and decoder are shared in the autoencoder. We use the Adam algorithm [101] (described in 4.3.1.1) to train the autoencoder. It is a Stochastic Gradient Descent algorithm which calculates the adaptive learning rates based on estimates of the first and second-order moments of the gradients [101].

As depicted in Figure 7-2, once the first autoencoder is trained, the decoder is discarded, and the output of the encoder is used to train the second layer of the neural network as an autoencoder. This process is repeated for all the hidden layers, and once this is done the final layer is added to the network and the whole network is retrained using supervised learning.

7.2 Pretraining D2FLS

Once the SAE is trained the outputs of the hidden layers of the SAE are used to pre-train the hidden layers of the D2FLS. i.e., the inputs and outputs of the first hidden layer of the SAE are used to train the first hidden layer of the D2FLS. Next, the inputs and outputs of the second hidden layer of the SAE are used to train the second hidden layer of the D2FLS etc. This process is repeated until all the hidden layers of the D2FLS are pre-trained.

Let F_k be k^{th} the layer of the D2FLS to be trained and given the training samples $[x_1, x_2, \dots, x_p]$ where $x_p \in R$. The D2FLS SAE pretraining can be represented as follows.

$$h_k = F_k(h_{k-1}) \quad (7.3)$$

Where h_k is the output of the k^{th} layer of the SAE trained in the previous phase, h_{k-1} is the output of the $k-1^{th}$ layer of the SAE. When $k=1$ and $h_{k-1} = x_p$ for the training samples.

To optimize the hidden layers, the MFs and rule base are optimized using the BB-BC algorithm and the training is divided into three steps, this is similar to the way FAEs are trained, and the process is depicted in Figure 6-10. The goal of the optimization algorithm is to minimize a cost function such as MAE (6.24), which is modified as follows:

$$MAE = \frac{\sum_{i=1}^p |h_k - \hat{h}_k|}{p} \quad (7.4)$$

Where \hat{h}_k is the output of the hidden layer of the D2FLS, h_k is the desired output which is the output of the k^{th} hidden layer of the SAE, p is the number of instances in the training data.

7.2.1.1 Optimize Hidden layer as Type 1 FLS

To optimize the type-1 Hidden FLS, we use an optimisation algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). The first step in these algorithms is to encode the parameters (to be tuned) of the hidden FLS into individuals. Each individual represents a possible solution to the optimization problem.

There are three sets of parameters to be tuned for the hidden FLS, the MFs that describe the input features or linguistic variables, the MFs that describe the output linguistic variables and the rules of the FLS. In this step, we train all these parameters, which are encoded into the individual as real numbered values as follows.

$$\phi_{HLT1} = M_1, \dots, M_i, \dots, M_{i+k}, R_1, \dots, R_l \quad (7.5)$$

Where M_i represents the membership functions for the i input features or linguistic variables of the FLS and M_{i+k} represents the MFs for the k^{th} output or consequent of the FLS using (6.2), (6.5) or (6.8). R_l represents the l rules of the FLS using (6.9).

For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FAE is performed using the following steps.

Step 1: N individuals are initialised by randomly generating values for each of the parameters of ϕ_{HLT1} in (7.5).

Step 2: The N individuals are then decoded into an FLS using (6.28), and the fitness of these individuals is calculated using the cost function in (7.4).

Step 3: The best individual among these N FLS is selected, and the stopping criteria are checked against this solution. If this FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this FLS.

Step 4: Else, a new generation of N individuals are generated by mutating this individual using (4.2), and the steps from Step 2 are repeated.

7.2.1.2 Transform TIMFs to IT2MFs

In the second step, we train the FOU of the MFs of the antecedents and the consequents using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). To do this, we add an FOU to the representation of the MFs of the antecedents and the consequents of the hidden FLS trained in the previous step. The representation of the FLS in (7.5) is modified using (6.13), (6.16) or (6.19) (based on the type of MFs) as follows.

$$T = T2_1, \dots, T2_i, \dots, T2_{i+k}, R_1, \dots, R_l \quad (7.6)$$

Where $T2_i$ represents the membership functions for the i input features or linguistic variables FLS and $T2_{i+k}$ represents the MFs for the k^{th} output or consequent of the FLS using (6.13), (6.16) or (6.19). Since we only train the parameters of the MFs and their FOU in this step, the parameters of the MFs are encoded into an individual as follows.

$$\phi_{HLIT2} = T2_1, \dots, T2_i, \dots, T2_{i+k} \quad (7.7)$$

The FOU's of the hidden FLS are then trained. For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FOU's of the hidden FLS is performed using the following steps.

Step 1: $N-1$ individuals are generated by randomly generating values within the search space for each of the parameters in ϕ_{HLIT_2} . The final individual is generated by choosing the parameters from the type-1 hidden FLS, trained in the previous step of the training (the FOU's of this individual are set to zero).

Step 2: N individuals are then decoded into the MFs of the FLS using (7.7), the rules are then added to these FLSs by choosing them from the type-1 hidden FLS trained in the previous step.

Step 3: The fitness of the N FLSs are calculated using the cost function in (7.4). The best individual among these N individuals is selected, and the stopping criteria are checked against this individual. If this FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this FLS.

Step 4: Else, a new generation of N individuals are generated by mutating this solution using (4.2) and the steps from Step 2 are repeated.

7.2.1.3 Optimize the Rule Base of the Hidden Layer

In the third step, we retrain the rules of the hidden FLS generated in the previous section. The rules of the FLS are encoded using the representation described in Section 6.1.1.2 using (6.9) as follows.

$$\phi_{HLRules} = R_1, \dots, R_l \quad (7.8)$$

Where R_l represent the l rules of the hidden FLS.

The rules of the FLS are then retrained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the retraining of the rules is performed using the following steps.

Step 1: One individual is generated by encoding the rules of the hidden FLS generated in the previous step into a real-valued solution using (7.8). Along with this, $N-1$ individuals are generated by mutating the first individual.

Step 2: The N individuals are then decoded into the rules of the hidden FLSs using (7.8). Then the MFs and their FOU's generated in the previous section are added to the FLSs.

Step 3: The fitness of these N FLSs are calculated using the cost function in (7.4). The best solution among these N FLSs is selected, and the stopping criteria are checked against this solution. If this hidden FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this hidden FLS

Step 4: Else, a new generation of N Individuals are generated by mutating this solution using (4.2) , and the steps from Step 2 are repeated.

7.3 Optimization Method for the Final Layer

To train the full D2FLS, we use the hidden layers trained using the method explained in the previous section and added another FLS that will act as the final output layer of the D2FLS. The output of the D2FLS can then be represented as follows.

$$y = f(h^n(h^{n-1}(\dots(h^1(x_p)\dots))) \quad (7.9)$$

Where h^n represent the n^{th} hidden FLS, f represents the final output layer and x_p is the input vector.

We use an optimization algorithm to retrain all the layers using the three-step training process depicted in Figure 6-10. The goal of the optimization algorithm is to minimise a cost function such as MAE (6.24), which is modified as follows:

$$MAE = \frac{\sum_{i=1}^p |y - \hat{y}|}{p} \quad (7.10)$$

Where \hat{y} is the predicted output of the D2FLS from equation (7.9), y is the actual output from the training dataset, n is the number of instances in the training dataset.

7.3.1.1 Optimize the Type 1 D2FLS

In this step, we stack the hidden layers trained in section 7.2 and add a final layer. We train the final layer as a Type-1 FLS while at the same time, we retrain the MFs and rules of the hidden layers using the BB-BC algorithm. In this step, the inputs and outputs are extracted directly from the data. The parameters of the MFs and rules of the D2FLS are represented in the following format.

$$\phi_{D2FLST1} = T_h^1, T_h^2, \dots, T_h^n, M_1^f, \dots, M_{o+p}^f, R_1^f, \dots, R_g^f \quad (7.11)$$

Where T_h^n represent the MFs and rules of the n hidden layers created using (6.20). M_{o+p}^f represents the MFs for the o input features, p consequents of the final layer and R_g^f represents the g rules of the final layer.

The D2FLS is then trained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the D2FLS is performed using the following steps.

Step 1: N individuals are generated, the initial values of the three parameters of the final layer of the D2FLS in (7.11) are randomly generated as real numbered values. These values are then added to the parameters of the encoders generated in the previous phase.

Step 2: The N individuals are then decoded into a D2FLS using (7.11) , and the fitness of these individuals is calculated using the cost function in (7.10) and the best solution among these N D2FLS is selected.

Step 3: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this individual satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS.

Step 4: If the stopping criteria (number of generations and target fitness) are not satisfied, a new generation of N individuals are generated by mutating the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

7.3.1.2 Transform TIMFs of the final layer to IT2MFs

In this step, we add the FOU to the MFs of the antecedents and consequent membership functions of the final layer generated in the previous steps and retrain FOU's of all the layers using the BB-BC algorithm. The parameters for the FOU's are encoded using and represented as follows.

$$\phi_{D2FLST2} = T_h^1, T_h^2, \dots, T_h^n, T2_1^f, \dots, T2_{o+p}^f \quad (7.12)$$

Where T_h^n represents the IT2 representation of the MFs of the h encoders from (6.30) and $T2_{o+p}^f$ represents the MFs and FOU's of the o antecedents and the p consequents (6.20).

The FOU's of the D2FLS are then trained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the training of the FOU's of the D2FLS is performed using the following steps.

Step 1: N individuals are generated by randomly generating the FOU's of the antecedents and the consequents of the final layer of the D2FLS and added to the D2FLS generated in the previous section and their parameters encoded using (7.12).

Step 2: The real-valued representation of the N individuals is then decoded into MFs of the D2FLS using (7.12). The rules are then added to these D2FLS by choosing them from the type-1 D2FLS trained in the previous step.

Step 3: The fitness of the N D2FLSs is calculated using the cost function in (7.12) and the best D2FLS among these is selected

Step 4: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this D2FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS

Step 5: If the stopping criteria (number of generations and target fitness) are not satisfied a new generation of the N candidate solutions are generated by mutating real-valued representation of the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

7.3.1.3 Optimizing the Rule Base of the D2FLS

In this step, we retrain the rules of all the hidden layers and the final layer using BB-BC algorithm. The parameters for this step are encoded in the following format to create the candidate solutions of the BB-BC algorithm.

$$\phi_{D2FLSRules} = R_1^{h1}, \dots, R_l^{h1}, \dots, R_l^{hn}, R_1^f, \dots, R_g^f \quad (7.13)$$

Where R_l^{hn} represents the rules of the n hidden layer and R_g^f represents the g rules of the final layer.

The rules of the D2FLS are then retrained using an optimization algorithm such as BB-BC (described in Chapter 4.1) or Genetic algorithm (described in Chapter 4.2). For example, if the BB-BC algorithm is used as the optimization algorithm, the retraining of the rules of the D2FLS is performed using the following steps.

Step 1: The rules of the D2FLS generated in the previous step are encoded into real-valued solution using (7.13). And N individuals are generated by mutating this solution.

Step 2: The N individuals are then decoded into the rules of the D2FLS, then the MFs and their FOU's generated in the previous section are added to the D2FLS. The fitness of these individuals is calculated using the cost function in (7.12), and the best individual among these is selected.

Step 3: The stopping criteria (number of generations and target fitness) are checked against the individual selected in the previous step. If this D2FLS satisfies the stopping criteria, then the optimization algorithm is stopped, and further steps of the training process are performed on this D2FLS.

Step 4: If the stopping criteria (number of generations and target fitness) are not satisfied a new generation of the N individuals are generated by mutating the individual selected in Step 2 using (4.2) and then the steps from Step 2 are repeated.

7.4 Experiments and Results

7.4.1 D2FLS vs Fuzzy Stacked Autoencoder

In this experiment, we compare the performance of the D2FLS against the FSAE. In [96] FSAE is shown to have performance that is between an IT2FLS and an SAE, so, this is an interesting comparison as we can use this to learn how the D2FLS stacks up against the IT2FLS and SAE models trained using the methods described in [96].

Table 7-1: Comparison between D2FLS and FSAE on Categorical Datasets with Average Recall as Fitness Functions

Data Set	AI Model	1	2	3	4	5	Average	Std
Santander CTP	D2FLS	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	FSAE	78.39	77.4	76.92	78.28	78.05	77.81	0.56
CLL Identification	D2FLS	62.8	59.78	61.25	62.37	63.31	61.9	1.26
	FSAE	56.19	56.33	86.94	86.61	87.28	74.67	15.03
BT Customer Data	D2FLS	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	FSAE	70.79	72.66	72.13	65.04	64.24	68.97	3.6
PD Speech	D2FLS	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	FSAE	73.13	64.85	73.73	76.1	79.85	73.53	4.94
IDA2016	D2FLS	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	FSAE	93.64	94.74	96.47	94.92	96.58	95.27	1.12
EpiSeizure	D2FLS	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	FSAE	95.32	95.13	96.34	96.51	95.87	95.83	0.54

The D2FLS model is trained on the classification datasets using BB-BC algorithm with 500 generations and 30 particles per step. The FSAE model is trained using a greedy layer-wise training [61]. We used two hidden layers with 400 and 30 neurons each. Adam Algorithm [101] was used for training the SAE, and we set the learning rate as 0.001, beta one as 0.9 and beta two as 0.999 and trained it for 200 epochs. The final layer of the FSAE is trained using BB-BC with 500 generations and 30 particles per step.

Table 7-2: Comparison between D2FLS and FSAE on Regression Datasets with Mean Absolute Error as the fitness function

Data Set	AI Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	FSAE	0.0384	0.0457	0.0658	0.0577	0.0438	0.0503	0.01
Swiss Premium Pred	D2FLS	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	FSAE	0.0534	0.0767	0.0629	0.0584	0.0652	0.0633	0.0078
CT Scan Region Pred	D2FLS	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	FSAE	0.0561	0.0486	0.055	0.06	0.0581	0.0556	0.0039
Predict Song Year	D2FLS	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	FSAE	0.0738	0.0727	0.074	0.0775	0.0763	0.0748	0.0018
BT PWA	D2FLS	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	FSAE	0.0436	0.04	0.048	0.046	0.0455	0.0446	0.0027

The results of training the two models with Average Recall as the fitness function over five training runs on the categorical datasets is tabulated in Table 7-1. The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table. Where a result is in bold, it indicates that the row contains the AI model with the best performance for the dataset.

We can see from Table 7-1 that the FSAE performance better than the D2FLS in five of the datasets while the D2FLS outperforms it in only one of the datasets. The highest performance difference is in the Santander CTP and CLL Identification datasets, where the FSAE performance the best. The FSAE also outperforms the CNN and the SAE (hidden layers of the model used as inputs to the FSAE) in these datasets (from Table 6-2). If we do not consider these two datasets, the performance difference is small within 2-3 % of each other.

Similarly, the results of training the two models with Mean Absolute Error as the fitness function over five training runs on the regression datasets is tabulated in Table 7-2. The mean and standard deviation of the five training runs is displayed in the eighth

Table 7-3 : D2FLS Pretrained using FAE vs D2FLS Pretrained using SAE on Classification Datasets with Average Recall as the fitness function

Data Set	Training Method	1	2	3	4	5	Average	Std
Santander CTP	D2FLS FAE	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	D2FLS SAE	62.93	58.91	60.18	61.71	61.53	61.05	1.38
CLL Identification	D2FLS FAE	62.8	59.78	61.25	62.37	63.31	61.9	1.26
	D2FLS SAE	62.47	62.96	62.3	62.23	57.12	61.42	2.16
BT Customer Data	D2FLS FAE	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	D2FLS SAE	60.34	64.43	59.93	65.19	60.28	62.04	2.28
PD Speech	D2FLS FAE	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	D2FLS SAE	70.21	72.55	77.03	65.79	72.14	71.55	3.64
IDA2016	D2FLS FAE	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	D2FLS SAE	94.69	92.28	95.27	94.99	95.31	94.51	1.14
EpiSeizure	D2FLS FAE	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	D2FLS SAE	90.71	89.76	90.39	92.11	90.25	90.64	0.79

and ninth columns of the table. Where a result is in bold, it indicates that the row contains the results of the AI model with the best performance for the dataset.

Again, we see similar results in that the FSAE outperforms the D2FLS in four of the five datasets. But the average increase in error when using the D2FLS is only about 14% which is a relatively small performance difference.

The small loss in performance between the D2FLS and FSAE is reasonable considering that the FSAE has lower interpretability than the D2FLS and that we generally get better performance if we use the Deep learning counterparts such as SAE and CNN.

7.4.2 D2FLS Comparison between The Two Training Methods

In this experiment, we compare the performance of the D2FLS when it is pre-trained as an FAE (D2FLS FAE) against a D2FLS when it is pre-trained using the hidden layers

Table 7-4: D2FLS Pretrained using FAE vs D2FLS Pretrained using SAE on Regression datasets with MAE as the fitness function

Data Set	Training Method	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS FAE	0.106	0.116	0.1049	0.1022	0.098	0.105	0.0059
	D2FLS SAE	0.1241	0.1127	0.1073	0.1231	0.1075	0.115	0.0073
Swiss Premium Pred	D2FLS FAE	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.0041
	D2FLS SAE	0.0474	0.0533	0.0503	0.058	0.0586	0.0535	0.0043
CT Scan Region Pred	D2FLS FAE	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.0033
	D2FLS SAE	0.0932	0.0803	0.0906	0.096	0.0883	0.0897	0.0053
Predict Song Year	D2FLS FAE	0.076	0.0741	0.074	0.074	0.079	0.075	0.0022
	D2FLS SAE	0.0731	0.0746	0.0728	0.0747	0.0768	0.0744	0.0014
BT PWA	D2FLS FAE	0.048	0.057	0.0511	0.045	0.057	0.0519	0.0047
	D2FLS SAE	0.0476	0.0523	0.0468	0.0644	0.0646	0.0551	0.0079

of an SAE (D2FLS SAE). The goal of this experiment is to find the best pretraining method for the proposed D2FLS model.

We tabulate the results of the D2FLS trained using both the training methods five times on the classification datasets using BB-BC in Table 7-3. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the D2FLS training method with the best performance for the dataset

From Table 7-3, we can see that the D2FLS FAE outperforms the D2FLS SAE in five of the six of the datasets. The D2FLS SAE performs better in only one dataset, but the performance difference between the two methods is small. The biggest difference

Table 7-5: Snapshot of Rule base of the Hidden Layer of the D2FLS trained using SAE on the BT PWA Dataset

ID	Antecedents			Consequents				
	1	2	3	H00	H01	H02	H03	H04
1	Mid LOANS 1	Low MSLCL2 2	High MSLCL2 3	Very High	High	High	Low	Very Low
2	Mid MSLCL1 0	Mid MISSAPP 3	Low ECONOMIC UTILISATION 3	Very High	Mid	High	High	Very High
3	High ECONOMIC UTILISATION 2	Mid CONTRACTOR 1	Low TRAVEL 1	Mid	Low	Mid	High	Very Low
4	Low MSLCL1 1	Low ON DAY UTILISATION 2	Mid ON DAY UTILISATION 1	High	Very High	Very Low	Very High	Very High
5	Mid MSLCL1 2	High TRAVEL 3	High CALC PROD 3	Low	Very Low	Very Low	Very High	Low
6	High MSLCL2 3	Mid MISSAPP 3	Low ECONOMIC UTILISATION 0	Low	Very Low	Very High	Mid	High
7	Mid MISSAPP 3	Low OT HOURS 1	Mid MSLCL2 2	Very High	Very Low	Very High	Mid	Low
8	High RANK 3	High ON DAY UTILISATION 1	Low MSLCL2 0	Very High	High	Very High	High	Low
9	High OT HOURS 3	Low MSLCL2 0	High ON DAY UTILISATION 2	Low	High	High	Mid	High
10	Mid ON DAY UTILISATION 3	Low MISSAPP 0	Low MSLCL2 2	Very High	Low	High	Very High	High
11	Mid TRAVEL 3	Mid ECONOMIC UTILISATION 3	Low MSLCL2 2	Mid	Mid	High	Very High	Very Low
12	High CALC PROD 0	Low TRAVEL 2	High OT HOURS 1	Mid	Low	Low	Very High	Mid
13	High ON DAY UTILISATION 0	Mid MISSAPP 2	Low TRAVEL 2	Very Low	Low	Low	High	Very High
14	Mid RANK 1	Low MSLCL2 3	Mid TRAVEL 3	Very Low	Mid	High	High	High
15	Low ECONOMIC UTILISATION 3	Mid RANK 2	Mid CALC PROD 3	Very Low	Very Low	High	Low	Mid

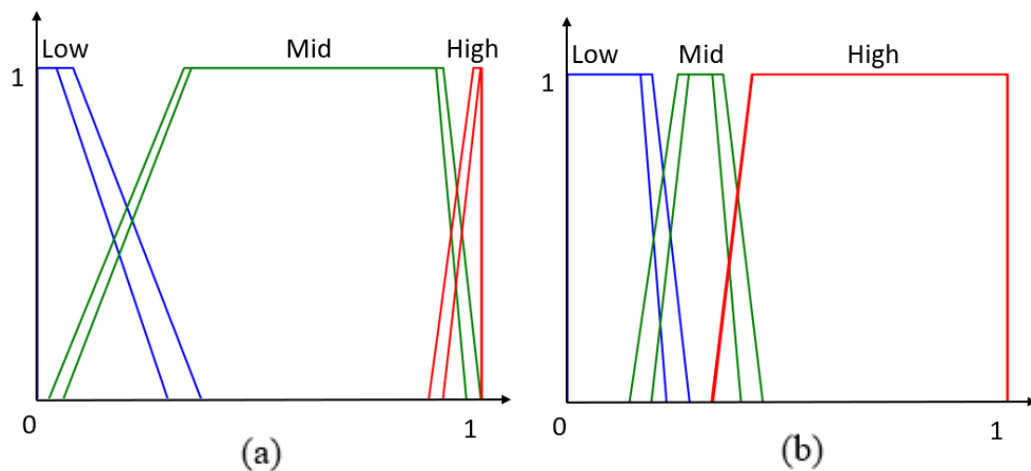


Figure 7-3: Fuzzy Sets Generated by D2FLS SAE Training for (a) Loans 1 feature and (b) MSLCL2 2 feature of the BT PWA dataset

between the two training methods is in the BT Customer Dataset where the D2FLS FAE outperforms the D2FLS SAE by about 10%, but the difference in performance between the two training methods is only about 2% performance on average.

We tabulate the results of D2FLS trained using both the training methods five times on the regression datasets using BB-BC is tabulated in Table 7-4. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the D2FLS training method with the best performance for the dataset.

Table 7-4 shows that like the classification datasets the D2FLS FAE outperforms the D2FLS SAE method in four of the five regression datasets. The D2FLS FAE has only about 6% lower error rate compared to D2FLS SAE on average.

A snapshot of the rule base generated for D2FLS trained using SAE on the BT PWA dataset is shown in Table 7-3 and Table 7-6. Table 7-3 contains a snapshot of the rules

(15 out of 100 rules) of the Hidden Layer of the D2FLS; it shows that the rules are short and comprise of three antecedents and five consequents that form the five outputs of the hidden layer (H00, H01, H02, H03, H04). Membership functions for the first two antecedents of the first rule in Table 7-3 are depicted in Figure 7-3 (a) and (b). Table 7-6 contains a snapshot of the rules of the output layer of the D2FLS, and the rules of the output layers are also kept short and comprise of three antecedents and one consequent for the output. The inputs to the output layer (H00, H01, H02, H03, H04) of the D2FLS are synthetic variables created during the training.

Comparing the D2FLS FAE rules in Table 6-4 and Table 6-5, and the rules of the D2FLS trained using both the methods provide similar interpretability and the small performance difference between the two training methods means that both the training methods are viable and can be chosen based on the user preference. As an example, the alternative training method presented in this chapter outperformed the training method

Table 7-6: Snapshot of Rule base of the Output Layer of the D2FLS pre-trained using SAE on the BT PWA Dataset

Rule No	Antecedents			Consequent
	1	2	3	PWA Performance
1	High H01	High H02	Mid H03	Mid
2	Low H00	Mid H01	Low H03	Mid
3	Low H01	Mid H00	Mid H02	Low
4	Mid H02	Mid H00	High H03	Very Low
5	Low H01	Mid H03	Low H00	High
6	High H02	High H01	Mid H00	Mid
7	Mid H00	High H01	High H03	Very Low
8	High H02	Low H03	High H01	Very Low
9	High H01	Mid H02	Mid H03	Mid
10	Low H02	Mid H01	Low H00	Low
11	High H02	Low H01	High H00	Low
12	Mid H02	Low H01	Low H00	Low
13	Low H02	Mid H01	Low H00	Very Very Low
14	Mid H03	High H01	Mid H00	Mid
15	High H01	Mid H03	Mid H00	Low

presented in the previous chapter by 2% in the IDA 2016 dataset, the change in average recall was from 92.5 to 94.5%. While the D2FLS FAE outperformed the D2FLS SAE in the BT Customer dataset by about 10% in the BT Customer dataset.

The performance of the FSAE is better than the D2FLS and since the alternative training method is an extension of the FSAE we believe there is scope for further improvement in the performance of the D2FLS when trained using SAE.

7.4.3 Effectiveness of Pre-training on the D2FLS

In this experiment, we aim to verify the effectiveness of pretraining on the D2FLS for the selected datasets. This raises an interesting question, why do we need to pretrain the D2FLS? as we can train the D2FLS without any pretraining and directly train the D2FLS as a multilayer Fuzzy Logic System.

We tabulate the results of the D2FLS FAE and a D2FLS trained without any pre-training, trained five times on the classification datasets using BB-BC in Table 7-7. The results of the training runs are presented as Average Recall (equation (6.21)) in

Table 7-7: Comparison of D2FLS with hidden layers pre trained using FAE against D2FLS without pretraining on Classification Datasets with fitness function Average Recall

Data Set	Pre-Training	1	2	3	4	5	Average	Std
Santander CTP	D2FLS	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	No Pretraining	60.06	59.66	61.53	60.86	60.75	60.57	0.65
CLL Identification	D2FLS	62.8	59.78	61.25	62.37	63.31	61.9	1.26
	No Pretraining	59.4	59.78	61.85	61.55	61.8	60.87	1.06
BT Customer Data	D2FLS	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	No Pretraining	59.9	60.8	59.23	60.41	60.43	60.15	0.54
PD Speech	D2FLS	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	No Pretraining	71.92	68.85	70.78	71.49	64.33	69.48	2.78
IDA2016	D2FLS	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	No Pretraining	91.37	91.99	90.45	92.74	92.81	91.87	0.88
Epi Seizure	D2FLS	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	No Pretraining	88.87	87.07	87.44	90.14	87.02	88.11	1.22

Table 7-8: Comparison of D2FLS with hidden layers pre trained using FAE against D2FLS without pretraining on Regression datasets with MAE as the fitness function

Data Set							Average	Std
Wi-Fi Localization	D2FLS	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	No Pretraining	0.1157	0.1106	0.157	0.1563	0.2003	0.148	0.0326
Swiss Premium Pred	D2FLS	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	No Pretraining	0.0429	0.0607	0.0538	0.0658	0.0527	0.0552	0.0078
CT Scan Region Pred	D2FLS	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	No Pretraining	0.0935	0.1014	0.0968	0.0961	0.0999	0.0976	0.0028
Predict Song Year	D2FLS	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	No Pretraining	0.0822	0.083	0.0751	0.0755	0.0747	0.0781	0.0037
BT PWA	D2FLS	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	No Pretraining	0.0629	0.0668	0.0528	0.064	0.0455	0.0584	0.008

columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the D2FLS pre-training method with the best performance for the dataset.

From Table 7-7, we can see that the D2FLS pre-trained using FAE always outperforms the D2FLS with no pre-training in all the selected classification datasets. The best improvement is in the BT Customer Dataset, where pretraining provides a 12% improvement in performance. In PD Speech, Epi Seizure, and the Santander CTP datasets pre-training improves the performance by about 2-4%. In the IDA 2016 dataset, pre-training improves performance by less than 1%. While the overall improvement in performance in the selected classification datasets is about 4% on average.

Similarly, we tabulate the results of the D2FLS FAE and a D2FLS trained without any pre-training, trained five times on the Regression datasets using BB-BC in Table

7-8. The results of the training runs are presented as MAE (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the D2FLS pre-training method with the best performance for the dataset

From Table 7-8, we can see that pre-training improves the performance in all the selected regression datasets. With the most improvement seen in the Wi-Fi localization dataset, about 40% lower error rate in the pre-trained model. The next significant improvement is in Swiss premium prediction and BT PWA datasets with about 17% and 12.5% lower error rate in the pre-trained model. The improvement is smaller in the other two datasets with an overall average improvement of about 17% lower error rate.

This indicates that pre-training provides significant benefits over a simple multi-layer fuzzy logic system.

7.5 Summary

This chapter presented the Fuzzy Stacked Autoencoder and an alternative to the D2FLS training method we presented in the last chapter. It demonstrated that the two training methods perform similar to each other. With the D2FLS FAE outperforming the D2FLS SAE in classification dataset by an average of 2% and in the regression datasets the D2FLS FAE has about 6% lower error on average. With both the methods providing similar interpretability, the choice of the training method depends on the dataset that is being used. As an example, the alternative training method presented in this chapter outperformed the training method presented in the previous chapter by 2% in the IDA 2016 dataset, the change in average recall was from 92.5 to 94.5%. While

the D2FLS FAE outperformed the D2FLS SAE in the BT Customer dataset by about 10%.

The chapter also shows that pretraining offers significant benefits. The chapter shows that pretraining improves the performance of the D2FLS by 4% in classification datasets. And pre-trained D2FLS has 17% lower error rate in regression dataset.

The next chapter discusses the proposed methods for achieving local interpretability in the D2FLS.

Chapter 8. Local Interpretability Enhancement for Deep Type-2 Fuzzy Logic Systems

The D2FLS proposed in the previous two chapters is a rule-based system and can be considered interpretable. However, the intermediate variables connecting the various modules or layers of the D2FLS are with only limited semantic support; that is, they are synthetic variables. For example, consider one of the rules of the output or final layer of a D2FLS from Table 7-6 (displayed in (8.1)).

$$\textit{IF } H02 \textit{ is Mid and } H00 \textit{ is Mid and } H03 \textit{ is High THEN PWA Performance is Very Low} \quad (8.1)$$

The rule itself is quite simple with only three inputs, however, given that the user does not know what $H02$, $H00$ and $H03$ mean, they cannot interpret such a rule in isolation, so the user will need to analyse the D2FLS system as a whole. The user will have to inspect the rules of all the layers of the D2FLS to understand $H02$, $H00$ or $H03$. Consequently, the rules of the D2FLS could be challenging to understand for a lay user.

Hence, there is the need to provide linguistic meaning to the intermediate variables such as $H02$, $H00$ and $H03$. This is related to the idea of cointension; cointension refers to a relation between concepts, such that two concepts are cointensive if they refer to the same objects[117-119]. Therefore, a knowledge base will be interpretable if its semantics are cointensive with the knowledge that a user builds in his/her mind after reading the knowledge representation, expressed in natural language [117-119].

Hence, in this chapter, we propose a method to provide linguistic meaning to the intermediate variables for individual predictions. We note that interpretability is a fluid

concept and depends on the target audience [13]. For example, machine learning experts may be able to interpret the output based on the firing levels of the rules, but a layman may be more comfortable with a small number of weighted features as an explanation [40]. Hence, we also propose to create another set of explanations for each of the input-output pairs. We designate these two explanations as rule-based explanations and feature importance, respectively. The algorithms for creating these explanations are for a two-layer D2FLS, but they can be easily extended for multiple layers.

Additionally, since we use the algorithm of the model itself to generate the explanations, these explanations are more accurate when compared to model agnostic methods such as LIME [32] and SHAP [43] where the algorithms are independent of the model used. Since simple algorithms are used to generate these explanations, they are much easier to understand and verify when compared to complex algorithms used to generate explanations for DNN such as LRP [30], Deep Lift [31] etc.

8.1 Rule-Based Explanations

As explained previously, one of the problems with D2FLS rules is that the linguistic variables or labels of the intermediate variables that connect the various layers of the D2FLS are synthetic variable (created using data). This results in increased complexity in interpreting the rules of the D2FLS as the rules of all the layers must be analysed as a whole.

One way to reduce this complexity is by naming the intermediate variables or the outputs of each layer based on the features that contributed most to that output; these names are termed *compound inputs*. This is done by finding the rule which contributed

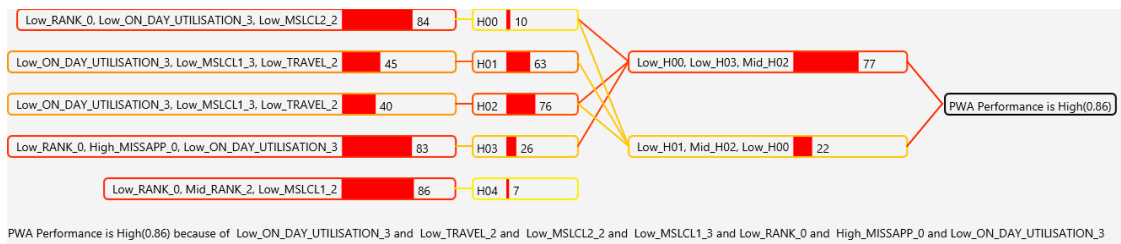


Figure 8-1: Rule-Based Explanation depicting the Rule contributions to the outputs of the D2FLS layers

the most to an output and the antecedents of this rule are used to name this variable. As the rule contribution can change based on the input, the linguistic label for the intermediate variables will also change based on the prediction. Hence, these explanations are locally interpretable and might not be valid at a global level.

The contribution of the rules to the output is calculated based on the product of the firing level and centroid of the consequent of the rule. This calculation changes based on the type of problem, for regression datasets the rule contributions are described in Section 8.1.1. In the case of classification datasets, the rule contributions are described in section 8.1.2.

For example, in Figure 8-1, we depict a local explanation for a two-layer D2FLS with the first layer having five outputs or intermediate variables designated H00, H01, H02, H03 and H05.

The output H00 of the first layer is a compound input consisting of Low Rank 0, Low on Day Utilisation 3 and Low MSLCL2 2 because this rule contributes the most

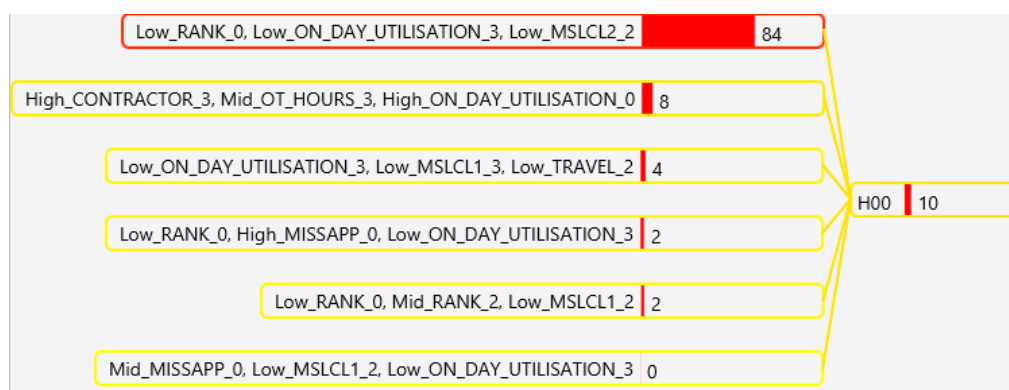


Figure 8-2: Rule Contributions for the Intermediate Variable H00

(84%) to this output or consequent. There are six other rules that contribute to this output (depicted in Figure 8-2), but their contribution is much smaller.

Similarly, all the outputs of the first layer can be named as compound inputs, using the contributions depicted in Figure 8-1, as follows.

- H00=Low Rank 0, Low on Day Utilisation 3 and Low MSLCL2 2
- H01=Low on Day Utilisation 3, Low MSLCL1 3 and Low Travel 2
- H02=Low on Day Utilisation 3, Low MSLCL1 3 and Low Travel 2
- H03=Low Rank 0, High MISSAPP 0 and Low on Day Utilisation 3
- H04=Low Rank 0, Mid Rank 2 and Low MSLCL1 2

These compound inputs are then used to find the relationship between the first and second layers of the D2FLS. For example, in Figure 8-1, the first rule of the output layer, which contributes 77% to the output, has three antecedents or inputs H00, H03 and H02. Which can be interpreted as compound inputs (Low Rank 0, Low on Day Utilisation 3 and Low MSLCL2 2), (Low Rank 0, High MISSAPP 0 and Low on Day Utilisation 3) and (Low on Day Utilisation 3, Low MSLCL1 3 and Low Travel 2) respectively.

We can use the same method for the second rule (22% contribution) which has three antecedents H01, H02 and H00. These three inputs to the second rule are named as follows (Low on Day Utilisation 3, Low MSLCL1 3 and Low Travel 2), (Low on Day Utilisation 3, Low MSLCL1 3 and Low Travel 2) and (Low Rank 0, Low on Day Utilisation 3 and Low MSLCL2 2) etc.

The rule that contributed the most to the final output can then be used to determine the input features that contribute the most to the output. As depicted in Figure 8-1, we

can say that the output (PWA performance) is High because of Low Rank 0, Low on Day Utilisation 3, Low MSLCL2 2, Low Rank 0, High MISSAPP 0, Low MSLCL1 3 and Low Travel 2.

The above example illustrates how the rule contributions can be used to provide a simple explanation for the output based on the rules of the various layers of the D2FLS. In the following sections, we describe the algorithm used to calculate the rule contributions.

8.1.1 Regression Datasets

To calculate these rule contribution values for regression datasets, we use the following equations.

$$\bar{R}_n^g = \bar{F}_n * c_n^g \quad (8.2)$$

$$\underline{R}_n^g = \underline{F}_n * c_n^g \quad (8.3)$$

Where \bar{F}_n and \underline{F}_n represent the upper and lower firing levels of the n^{th} rule and c_n^g represents the centroid of the g^{th} consequent of the n^{th} rule of the fuzzy layer.

$$R_{navg}^g = (\bar{R}_n^g + \underline{R}_n^g) / 2 \quad (8.4)$$

$$R_{nval}^g = R_{navg}^g / \sum_{n=1}^n R_{navg}^g * 100 \quad (8.5)$$

We use Equation (8.5) to calculate the rule contribution values for each of the rule and consequent combinations for all the layers.

8.1.2 Classification Datasets

To calculate these rule contribution values for classification datasets, we must modify the equations (8.2) and (8.3) for the final layer to the following.

$$\bar{R}_n^g = \bar{F}_n * \bar{C}_n^g * \bar{S}_n^g \quad (8.6)$$

$$\underline{R}_n^g = \underline{F}_n * \underline{C}_n^g * \underline{S}_n^g \quad (8.7)$$

Where \bar{F}_n and \underline{F}_n represent the upper and lower firing levels of the n^{th} rule, \bar{C}_n^g and \underline{C}_n^g represent the upper and lower confidence of the g^{th} consequent of the n^{th} rule of the final layer, \bar{S}_n^g and \underline{S}_n^g represent the upper and lower support of the g^{th} consequent of the n^{th} rule of the final layer. All other equations remain the same between regression and classification problems.

8.2 Feature Importance Explanations

Interpretability is a fluid concept and depends on the target audience, and, a lay user might not be comfortable with the rule-contribution based explanations provided in the previous section and may be more comfortable with a small number of weighted features as an explanation [40]. Hence, in this section, we detail a method to generate a simple explanation which shows the relationship between the input features and the output as Feature importance scores.

Essentially, we would show how much a feature influences the output. For example, in Figure 8-3 we see that the most important feature is Low Travel 2 (i.e., Travel 2 feature having a Low Linguistic Label) and the second most important input feature is Low MSLCL1 3 (i.e. the feature Mean Service Level CL1(MSLCL1) 3 having a Low

Output : PWA Performance is High(0.86)					
I...	Name	Low	Mid	High	Importance
0	TRAVEL_2	✓			100
1	MSLCL1_3	✓			100
2	ON_DAY_UTILISATION_3	✓			100
3	RANK_0	✓			79
4	MISSAPP_0			✓	70
5	CONTRACTOR_3			✓	42
6	MSLCL2_2	✓			41
7	ON_DAY_UTILISATION_0			✓	36
8	OT_HOURS_3		✓		36
9	MSLCL1_2	✓			25
10	RANK_2		✓		20
11	MISSAPP_0		✓		4

Figure 8-3: Feature Importance Scores

Linguistic Label) and the third most important input feature is Low on Day Utilization 3 (i.e. the feature On Day Utilization having a Low Linguistic Label) etc.

From these feature importance score, in Figure 8-3, it becomes immediately apparent which features are the most important to a particular output. These feature importance scores are similar to the DNN explanation methods such as LRP [30], Deep Lift [31] etc. (detailed in Chapter 2.1.2)

The advantage of the proposed method is that the feature importance also provides linguistic label along with the importance score, that is, from Figure 8-3 we can easily deduce that low travel, Low MSLCL1 3 or Low on Day Utilization 3 leads to High PWA Performance, while the methods described in Chapter 2.1.2 only provide an importance score for each of the relevant features, and the features must be examined in detail to understand how changes to the input affect the output.

Another advantage is that these feature importance scores are generated using the AI model algorithm itself, which allows the users to quickly verify the validity of the generated score by using the algorithms described in the next two sections, if required.

The calculation of these feature importance scores changes based on the type of problem being solved. For Regression and Classification tasks, the calculations are described in the next two sections.

8.2.1 Regression Datasets

To calculate the feature importance scores for regression datasets, we use Algorithm 1. The idea here is to calculate the contribution of each input feature to the output by using the rule firing levels and the membership grades of various layers of the D2FLS. In the case of a two-layered D2FLS, this is done by finding the rule that contributes the most to the output in the output layer. Then the contribution of the feature to each of the outputs in the hidden layer is calculated. The max of the product of these two values gives us the feature importance score.

The algorithm is used to calculate the feature importance score for each feature for each of the outputs of the D2FLS, that is, in case of a single output system there will be a single importance score for feature x , and, in case of a system with two outputs there will be two importance scores for feature x . Once the feature importance scores are calculated for all the features, we normalize the values between 0 and 100 for each output and display them, as shown in Figure 8-3.

Algorithm 1: Simplified Explanations, our proposed algorithm for calculating the feature importance score. This algorithm is valid for Regression problems

For $g = 1$ to number of consequents of the final layer
For $i=1$ to number of inputs
 $\bar{I}_i^g, \underline{I}_i^g \leftarrow 0$ (upper and lower feature importance scores)
For $b = 1$ to number of antecedents/inputs of the final layer
 $\bar{R}_{\max}^b, \underline{R}_{\max}^b \leftarrow 0$
For $n=1$ to number of rules
If n rule contains antecedent b
 $\bar{R}_n^b = \bar{F}_n * \bar{c}_n^g$ (Upper firing level, centroid)
 $\underline{R}_n^b = \underline{F}_n * \underline{c}_n^g$ (Lower Firing Level, centroid)
 $\bar{R}_{\max}^b = \max(\bar{R}_{\max}^b, \bar{R}_n^b)$
 $\underline{R}_{\max}^b = \max(\underline{R}_{\max}^b, \underline{R}_n^b)$
next n
For $l=1$ to number of rules of hidden layer
If l rule contains input feature i
 $\bar{R}_l^i = \bar{F}_l * \bar{c}_l^b * \bar{R}_{\max}^b * \bar{\mu}_a^l(x)$
 $\underline{R}_l^i = \underline{F}_l * \underline{c}_l^b * \underline{R}_{\max}^b * \underline{\mu}_a^l(x)$
 $\bar{I}_i^g = \max(\bar{I}_i^g, \bar{R}_l^i)$
 $\underline{I}_i^g = \max(\underline{I}_i^g, \underline{R}_l^i)$
next l
next b
 $I_{i\text{avg}}^g = (\bar{I}_i^g + \underline{I}_i^g) / 2$ (feature importance score per output)
next i
next g
Return I

8.2.2 Categorical Datasets

To calculate the feature importance scores for categorical datasets, we use Algorithm 2. Like the previous section, the idea here is to calculate the contribution of each input feature to the output by using the rule firing levels and the membership grades of the input features. This is done by finding the rule whose vote contributes the most to the category of the output in the output layer. Then like the regression algorithm, the contribution of the feature to each of the outputs in the hidden layer is

calculated. The max of the product of these two values gives us the feature importance score.

The algorithm is used to calculate the feature importance score for each feature for each of the outputs of the D2FLS, that is, for each output of the system there will be a and importance score for each of the features. Once the feature importance scores are calculated for all the features, we normalize the values between 0 and 100 per output and display them, as shown in Figure 8-3.

Algorithm 2: Simplified Explanations, our proposed algorithm for calculating the feature importance score for Classification problems.

For g = 1 to number of consequents of the final layer

For i=1 to number of inputs

$\bar{I}_i^g, \underline{I}_i^g \leftarrow 0$ (upper, lower feature importance scores)

For b = 1 to number of antecedents/inputs of the final layer

$\bar{R}_{\max}^b, \underline{R}_{\max}^b \leftarrow 0$

For n=1 to number of rules

If n rule contains antecedent b

$\bar{R}_n^b = \bar{F}_n * \bar{C}_n^g * \bar{S}_n^g$ (upper firing level, confidence and support)

$\underline{R}_n^b = \underline{F}_n * \underline{C}_n^g * \underline{S}_n^g$ (lower firing level, confidence and support)

$\bar{R}_{\max}^b = \max(\bar{R}_{\max}^b, \bar{R}_n^b)$

$\underline{R}_{\max}^b = \max(\underline{R}_{\max}^b, \underline{R}_n^b)$

 next n

For l=1 to number of rules of hidden layer

If l rule contains input feature i

$\bar{R}_l^i = \bar{F}_l * \bar{C}_l^b * \bar{R}_{\max}^b * \bar{\mu}_a^l(x)$

$\underline{R}_l^i = \underline{F}_l * \underline{C}_l^b * \underline{R}_{\max}^b * \underline{\mu}_a^l(x)$

$\bar{I}_i^g = \max(\bar{I}_i^g, \bar{R}_l^i)$

$\underline{I}_i^g = \max(\underline{I}_i^g, \underline{R}_l^i)$

 next l

 next b

$I_{iavg}^g = (\bar{I}_i^g + \underline{I}_i^g) / 2$ (feature importance score per output)

 next i

next g

Return I

8.3 Survey

In the previous sections, we examined two methods of extracting local explanations from the D2FLS described in Chapter 6. In this section, we evaluate these explanations. To evaluate the explainability of an AI system, we must consider the audience that will consume or examine these explanations. Hence, in this section, we conduct a survey with participants divided into three different sets, each representing a different type of audience.

- **AI Experts:** The first set of participants is experts with experience in researching AI algorithms. So, these participants are able to understand the intricacies of how the underlying AI algorithms work.
- **Domain Experts:** The second set of participants are experts in one or more of the fields from which the datasets used to build the AI models are selected. They are experts who have experience and examining and taking decisions based on the data used to build the AI models. So, these are the audience that is most likely to use these explanations.
- **Lay Users:** The third set of participants are individuals with no experience in the AI algorithms or the datasets.

We surveyed six individuals or subjects who are AI experts and two subjects who are Domain experts and two more subjects who are lay users.

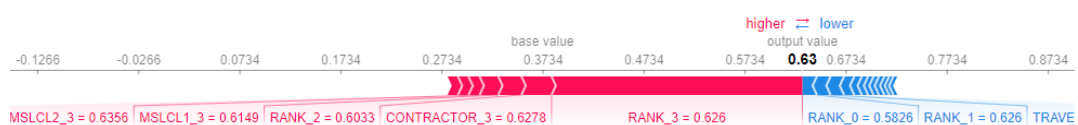


Figure 8-4: SHAP explanation for Sparse Stacked Autoencoder

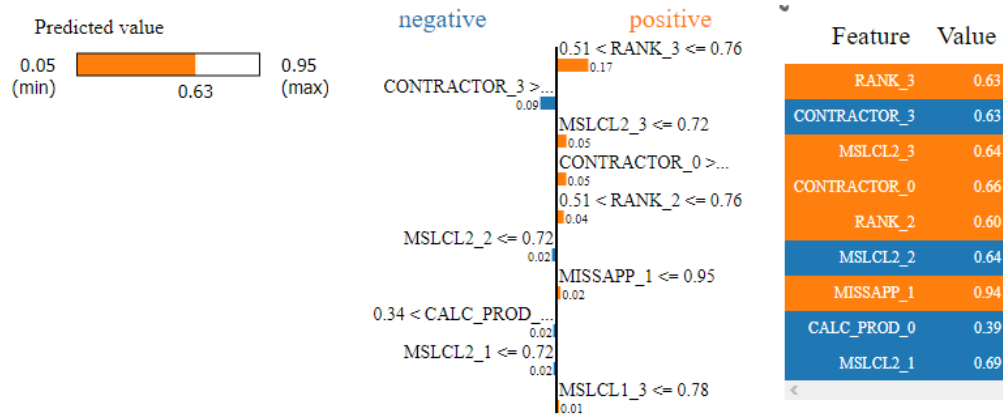


Figure 8-5: LIME explanation for Sparse Stacked Autoencoder

We used SHAP [43] (depicted in Figure 8-4) described in Chapter 2.3.3 and Lime [40] (depicted in Figure 8-5) described in Chapter 2.3.1, for generating the explanations for the SAE. Then we use the methods described in this chapter to generate the explanations for the D2FLS (depicted in Figure 8-1, Figure 8-2 and Figure 8-3) and IT2FLS.

We generated the explanations for a small set of data using the 3 AI models, and then we asked the participants of the survey to examine the explanations provided by the four XAI methods. And then rank them based on how good they are on a scale of 1 to 10 (10 being the best and 1 being the worst).

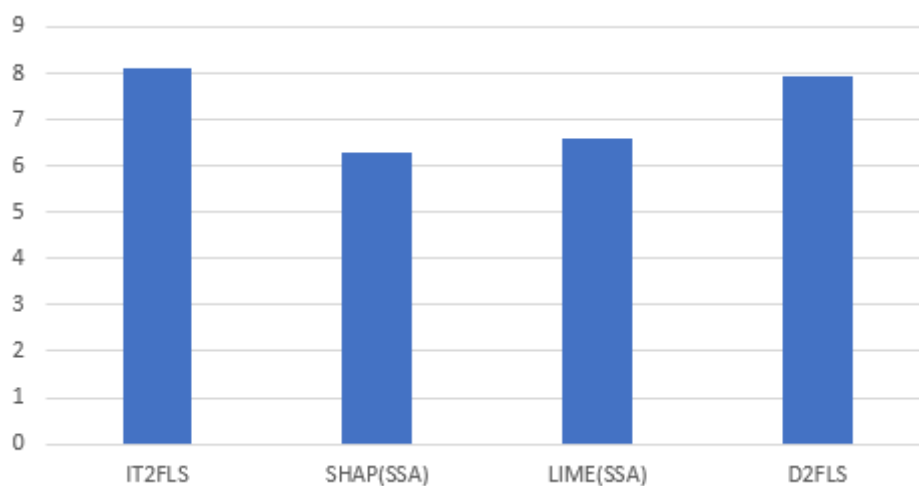


Figure 8-6: Survey Plot

Table 8-1: Survey Results

	Shap Tool	Lime Tool	D2FLS	IT2FLS
AI Expert	8	8	9.5	9
AI Expert	8	6	7	9
AI Expert	7	6	7	9
AI Expert	6	7	8	8
AI Expert	3	6	10	10
AI Expert	8	9	10	10
Domain Expert	4	4	5	3
Domain Expert	8	9	10	9
Lay User	7	7	7	8
Lay User	4	4	6	6
Average	6.3	6.6	7.95	8.1
Standard dev.	1.9465	1.7764	1.8326	2.1318

The results of the survey were tabulated in Table 8-1, and the average of these responses are plotted in Figure 8-6. We can see from the results that the explanations provided by the D2FLS are comparable (within 2%) to the explanations provided by a highly interpretable AI model, IT2FLS. At the same time, the performance of the D2FLS model is about 5% better than the IT2FLS in categorical datasets (From Table 6-6) and 15% lower error than the IT2FLS models in regression datasets (From Table 6-7). And the explanations provided by the D2FLS are better than the explanations provided by popular model agnostic methods such as LIME and SHAP, 20% and 17% respectively. At the same time, the performance of D2FLS model is only about 4-5% lower than the SAE model in categorical datasets (From Table 6-1) and 40% higher error in the regression datasets (From Table 6-2).

This shows that the D2FLS model, in conjunction with the explanation's methods detailed in this chapter, provides a significant improvement over DNN models in terms of interpretability while showing only a small loss in performance.

8.4 Summary

This chapter introduced two methods of extracting locally interpretable explanations from the D2FLS to decrease the complexity of interpreting the rules of the D2FLS. This is to ensure that the D2FLS explanations are accessible to a broader audience. It also discusses the need to tailor the explanations based on the audience who are likely to examine these explanations and how this is accomplished using the two methods.

This chapter also evaluated the explanations provided by the D2FLS against popular Deep learning XAI tools LIME, SHAP to interpret SAE and against explanations provided by an IT2FLS. This was done by conducting a survey, in which three different sets of the audience were asked to examine the explanations provided by these methods and rank them according to how easy they are to understand.

The results of this survey show that the explanations provided by the D2FLS are comparable to the explanations provided by the IT2FLS (with only about 2% lower interpretability) while the performance of the D2FLS is 5% in categorical datasets and 15% lower error in regression datasets. It also provides explanations which are better than the explanations provided by the tools LIME and SHAP, about 20% and 17% better interpretability, respectively. At the same time, the performance of D2FLS model is only about 4-5% lower than the SAE model in categorical datasets and 40% higher error in the regression datasets. This is a small loss in performance compared for significantly higher interpretability making D2FLS a suitable replacement for the other AI models in applications where interpretability is paramount.

In the next chapter, we extend the explanation extraction methods to provide Global Interpretability to the D2FLS.

Chapter 9. Enhanced Deep Type-2 Fuzzy Logic System for Global Interpretability

In regulated applications such as financial, medical, justice etc., where the reliability of the model must be guaranteed it might be necessary to build and use globally interpretable AI models. An AI model is Globally Interpretable if the user can comprehend the entire model [3]. To explain the global model output, you need to understand the trained model, knowledge of the algorithm and the data. This level of interpretability is about understanding how the model makes decisions, based on a holistic view of its features and each of the learned components such as rules, membership functions etc. Which features are essential and what kind of interactions between them take place? Global model interpretability helps to understand the distribution of your target outcome based on the features.

Global interpretability can, of course, be achieved by using interpretable models such as decision trees, Bayesian rules, fuzzy logic, etc [12]. However, these models can be less accurate when compared to the black box models, and they can also become opaque for high dimensional inputs [3, 14].

The D2FLS proposed in the previous two chapters is a rule-based system and can be considered globally interpretable. But, as discussed in Chapter 8, the intermediate variables connecting the various modules or layers of the D2FLS are without any semantic support; that is, they are synthetic variables (variables created using the data). This means that the rules of a single layer of the D2FLS cannot be examined in isolation, and the rules of all the layers of the D2FLS system have to be examined together to gain an understanding of the model at the global level.

For example, consider one of the rules in of the output or final layer of a D2FLS from Table 7-6 displayed in (9.1), where the output PWA performance is High.

IF H01 is Low and H03 is Mid and H00 is low
THEN PWA performance is High (9.1)

IF MSLCL2 3 is High and MISSAPP 3 is Mid and Eco Util 0 is Low
THEN H00 is Low and H01 is Very Low and H02 is Very High (9.2)
and H03 is Mid and H04 is High

If we examine the rules of the hidden layer from Table 7-5, we see that the rule which most closely matches the linguistic terms of the antecedents of the rule in (9.1) is the rule displayed in (9.2). So, from these two rules, we can say that when MSLCL2 3 is High, and MISSAPP 3 is Mid, and ECONOMIC UTILISATION 0 is Low then the PWA performance is High.

All the rules of the D2FLS in all its layers have to be examined to check for matching linguistic terms to gain a global understanding of the D2FLS. This process can be a time consuming and difficult for a lay user. Even if the user is able to analyse the rules, they might not be able to remember all the complex interactions within the D2FLS model well enough to be able to predict the behaviour of the model in all circumstances.

Such problems are encountered in a lot of algorithms which are Globally interpretable when there are many features in the input. This means that global interpretability is challenging to achieve in practice for inputs with many features. This has led to a decline in research on globally interpretable models and a focus on local interpretability [12]. A new set of methods have been proposed to gain global insight into models by examining multiple instances of local explanations. One such method is SP-LIME [40] where the input features which explain many different instances are

scored higher and based on these scores instances which are not redundant, i.e., which share fewer features are picked and shown to the user. This method is described as Local Interpretability for a Group of Predictions [32].

Hence, in this chapter, we propose to extract simple interpretable explanations at the modular level to provide a holistic assessment of the D2FLS model, i.e., provide a qualitative understanding of the relationship between the input features and the output for the model at a modular level. The module, in this case, is determined by using the linguistic terms of the antecedent or consequent MFs of the inputs and the outputs of the D2FLS model which are then used to create a set of input-output pairs. We propose to extend the two local explanations methods for D2FLS described in Chapter 8 by calculating the average rule contribution and average feature importance score for the input-output pairs of a module. The users of these models can then use the linguistic labels to generate the explanations at a modular level and use these explanations to make an informed decision on the feasibility of the model.

Another problem we encounter while trying to analyse the rules of the D2FLS is that the linguistic terms of outputs of one layer and the linguistic terms of the inputs of the next layer might not match. The solution to this is to examine the membership functions that define these linguistic terms. But this is another layer of complexity for a lay user to analyse. Hence, we propose a method to enhance the D2FLS to simplify the process of analysing the rules of the D2FLS. We do this by constraining the D2FLS during training to use the same membership functions for the consequents of a hidden FLS in the D2FLS and the antecedents of subsequent FLS.

9.1 Global Interpretability at Modular Level

9.1.1 Modular Rule-Based Explanation

The Modular Rule-Based Explanation propose (depicted in Figure 9-1) is an extension of the Rule-Based explanation discussed in Section 8.1. As explained previously, to gain a global understanding of the D2FLS, all its rules across various layers have to be examined. This process can be time-consuming and difficult for a lay user. Even if the user can analyse the rule, they might not be able to remember all the complex interactions within the D2FLS model well enough to be able to predict the behaviour of the model in all circumstances.

One way to reduce this complexity is by extracting simple rule-based explanations at the modular level. The module, in this case, is determined by using the linguistic labels of the antecedent or consequent MFs of the inputs and the outputs of the model. For example, for a categorical model that does binary classification as depicted in Figure 9-1, there could be two modules based on the output categories—one positive for broadband faults and another negative for when the output is not a broadband fault.

To calculate the rule that contributes the most to a module, a set of input-output pairs is selected based on the linguistic variable of the input and output membership functions. And the rule contribution is calculated for each rule for all the input-output

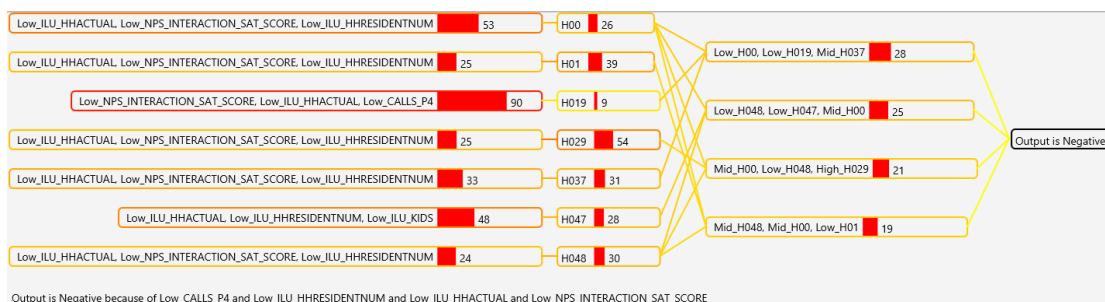


Figure 9-1: Rule-Based Explanation

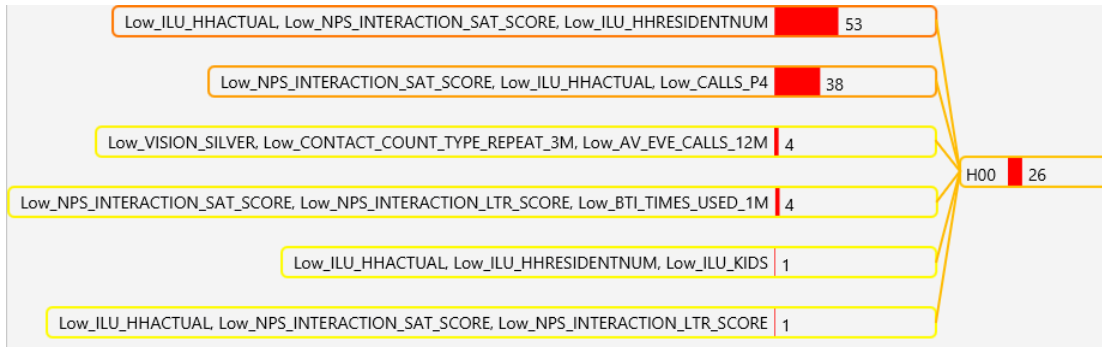


Figure 9-2: Rule-Based Explanation for one hidden output

pairs, and the average of these values is calculated for each rule and consequent using (9.6).

For example, in Figure 9-1, we generated Modular Detailed Explanations for all the input-output pairs where the linguistic term of the output is Negative. And we can see that the output H00 of the first layer is named Low ILU HH actual, Low NPS Interaction Sat Score, Low ILU HH num of residents because this rule contributes the most (53%) to this output, there are six other rules that contribute to this output (depicted in Figure 9-2), but their contributions are much smaller.

We can then use these compound inputs to find out the relationship between the input features and the output. For example, in Figure 9-1, the rule which contributes 28% to the output is composed of 3 compound inputs (Low ILU HH Actual, Low NPS Interaction Sat Score, Low ILU HH Resident Num), (Low NPS Interaction Sat Score, Low ILU HH Actual, Low Calls P4) and (Low ILU HH Actual, Low NPS Interaction Sat Score, Low ILU HH Resident Num).

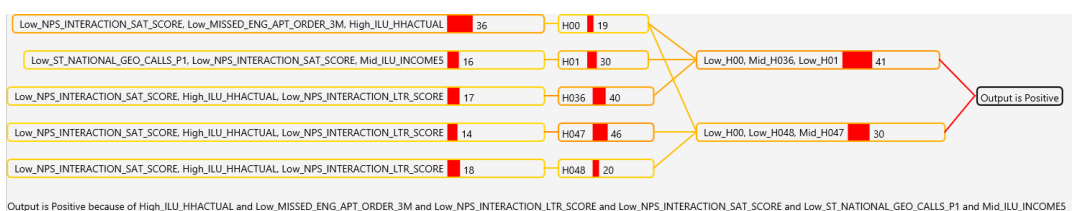


Figure 9-3: Rule Base Explanation for ILU HH Actual is High module

The above example illustrates how we can use a set of local interpretable explanations to find out which inputs contribute to the output of a module. In this case, the output is Negative because input feature ILU HH Actual is Low, ILU HH Resident Num is Low, NPS Interaction Sat Score is Low and Calls P4 is Low.

We could examine this further by restricting the input feature ILU HH actual to High. The rule-based explanation for this module is depicted in Figure 9-3. From this explanation, we can see that when ILU HH actual is high, then the output is generally “Positive” which confirms the idea that Low ILU HH Actual leads to an output of “Negative” in this D2FLS model.

The above example illustrates how the average rule contributions can be used to provide a simple rule-based explanation for the output of a module based on the rules of the various layers of the D2FLS. In the following sections, we describe the algorithm used to calculate the average rule contributions

9.1.1.1 Regression Datasets

For regression datasets, we calculate the rule contributions by using the below equations

$$\bar{R}_{avg}^n = \sum_{k=0}^j \bar{F}_n * \bar{c}_n / j \quad (9.3)$$

$$\underline{R}_{avg}^n = \sum_{k=0}^j \underline{F}_n * \underline{c}_n / j \quad (9.4)$$

Where \bar{F}_n and \underline{F}_n represent the upper and lower firing levels of the n^{th} rule and c_n represents the consequent of the n^{th} rule. j represents the total number of inputs selected.

We calculate the average rule firing strength across all the inputs in the dataset for both the upper and the lower membership functions using the below equations.

$$R_{avg}^n = (\bar{R}_{avg}^n + \underline{R}_{avg}^n) / 2 \quad (9.5)$$

$$R_{nval}^g = R_{avg}^n / \sum_{n=1}^n R_{avg}^n * 100 \quad (9.6)$$

9.1.1.2 Classification Datasets

For classification problem we modify equations (9.3) and (9.4) as follows.

$$\bar{R}_{avg}^n = \sum_{k=0}^j \bar{F}_n * \bar{C}_n * \bar{S}_n / j \quad (9.7)$$

$$\underline{R}_{avg}^n = \sum_{k=0}^j \underline{F}_n * \underline{C}_n * \underline{S}_n / j \quad (9.8)$$

Where \bar{F}_n and \underline{F}_n represent the upper and lower firing levels of the n^{th} rule, \bar{C}_n^g and \underline{C}_n^g represent the upper and lower confidence of the g^{th} consequent of the n^{th} rule of the final layer, \bar{S}_n^g and \underline{S}_n^g represent the upper and lower support of the g^{th} consequent of the n^{th} rule of the final layer. All other equations remain the same between regression and classification problems.

9.1.2 Modular Feature Importance Explanations

In this section, we propose to extend the Feature Importance explanations provided by the D2FLS from section 8.2. Similar to the Feature Importance explanations, we propose to show the relationship between the inputs and the outputs. But in this case, the inputs are a set of input-output data pairs selected based on a set of filter criteria that can be selected by the user. These filter criteria are linguistic labels of the input membership functions or the output membership functions. Essentially, we are trying

to find the features that contribute to a module of the model and assign importance to these features. These modules are determined using the linguistic labels of the input and output membership functions.

For example, in Figure 9-4, we generated Modular Feature Importance Explanations for input-output pairs where the linguistic label of the output is positive. We see that the model is accurate about 67% of the time and Number of Visits is High in 77% of the inputs, Downloads is Medium in 99% of the inputs, and these two inputs are the most important features. Similarly, in Figure 9-5, we generated the explanations for input-output pairs where the linguistic label of the output is negative. We see that the model is accurate 73% of the time, and ILU HH actual is the most important feature.

When we compare the two explanations in Figure 9-4 and Figure 9-5, we can see Number of Visits is high in 77% of the inputs when the output is positive, and it is low in 68% of the inputs when the output is negative. This indicates that the number of visits is a vital input feature and that a high number of visits indicates that the output is likely to be positive (We are trying to determine if there is a broadband fault).

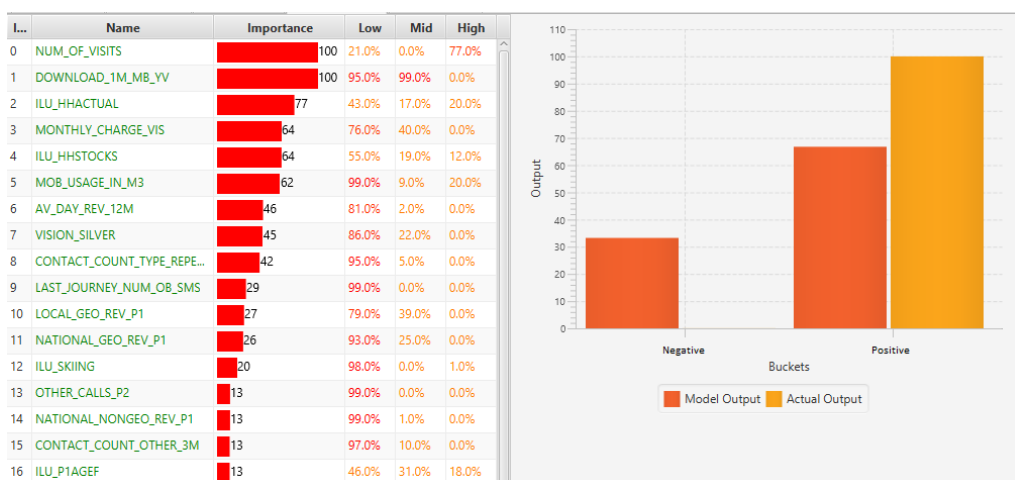


Figure 9-4: Modular Simple Explanation for Positive Outputs

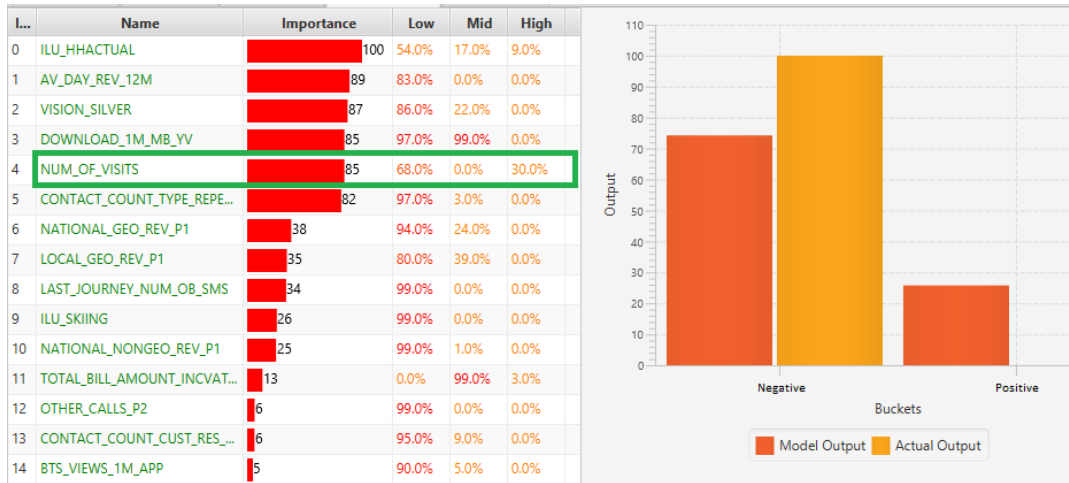


Figure 9-5: Modular Simple Explanation for Negative Outputs

The above example illustrates how we can use the modular feature importance score to understand which inputs contribute to the output at the modular level.

To calculate these values, we use the algorithms in section 8.2 to calculate the \bar{I}_i^g and \underline{I}_i^g values for each of the input features. Once we get these values, we use the below equations to calculate the average upper and lower importance score.

$$\bar{I}_{iavg}^g = \sum_0^j \bar{I}_i^g / j \quad (9.9)$$

$$\underline{I}_{iavg}^g = \sum_0^j \underline{I}_i^g / j \quad (9.10)$$

In the final step, we calculate the final importance score for the module for each of the features by taking the average of the upper and lower importance score using the below equation.

$$I_{avg}^i = (\bar{I}_{avg}^i + \underline{I}_{avg}^i) / 2 \quad (9.11)$$

After calculating the I_{avg}^i value for all the features, we normalize the values and display them, as shown in Figure 9-4.

To calculate the feature linguistic label values, we use the below equations

$$\bar{P}_i^k = \sum_1^j \bar{\mu}_i^k(x) \quad (9.12)$$

$$P_i^k = \sum_1^j \underline{\mu}_i^k(x) \quad (9.13)$$

Where, $\bar{\mu}_i^k(x)$ and $\underline{\mu}_i^k(x)$ are the upper and lower membership function for the linguistic label k of the i^{th} input feature. j is the number of input-output pairs. Finally, we use the below equation to calculate the percentage value.

$$P_i^k = \frac{(\bar{P}_i^k + P_i^k) / 2}{j} * 100 \quad (9.14)$$

9.2 Enhanced Deep Type-2 Fuzzy Logic System

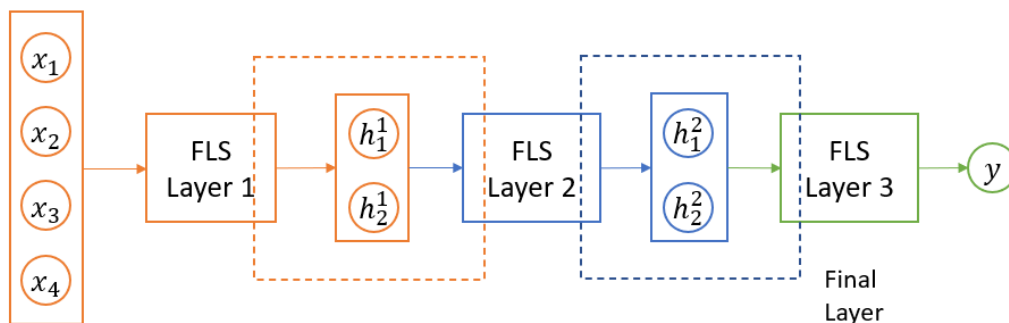


Figure 9-6: Enhanced Deep Type-2 Fuzzy Logic System. The Dotted Rectangles indicate the two FLSs that are constrained to use the same Linguistic Labels for the Consequents as the Antecedents of subsequent FLS

As discussed earlier, one of the problems we encounter while trying to analyse the rules of the D2FLS is that the linguistic terms of outputs of one layer and the linguistic terms of the inputs of the next layer might not match. The solution to this is to examine the membership functions that define these linguistic terms. But this is another layer of complexity for a lay user to analyse.

Hence, in this section, we propose to enhance the D2FLS (as depicted in Figure 9-6) by constraining the D2FLS during training to use the same linguistic labels for the consequents of a hidden FLS in the D2FLS and the antecedents of subsequent FLS. There are several advantages to this system.

- It reduces the number of parameters that need to be trained.
- It improves the readability of the rules of the various layers of the D2FLS.

The disadvantage of such a system is that it reduces the degree of freedom with likely consequences to the performance of the model.

To achieve this, we modify the training methods of the D2FLS as follows. We still use the two-stage layer-wise training method proposed in Section 6.2 with the following modifications.

9.2.1 Hidden layer training

In the hidden layer training, we still use the same three-step training with the following modifications to the representation of the FAE in the three steps.

9.2.1.1 Optimize Type 1 FAE

In the first step, we still train the Membership functions and rules of the two FLS that comprise the FAE as a Type-1 FLSs using the same procedure that is described in section 6.2.2.1. But the encoder part of the FAE is modified as follows.

$$N_e^n = M_1^{n-1}, \dots, M_i^{n-1}, M_{i+1}^n, \dots, M_{i+k}^n, R_1, \dots, R_l \quad (9.15)$$

Where M_i^{e-1} represent the antecedent MFs which are the consequent MFs of the encoder of the preceding FAE. And the FAE representation is modified as follows

$$E_n = N_e^n, R_1^d, \dots, R_m^d \quad (9.16)$$

Where R_m^d represents the rules of the decoder. We only train the rules of the decoder as we add a constraint to share the linguistic variables between the encoder and decoder.

Note that we do not train the antecedent MFs, if there are any preceding hidden layers, as they are the same as the consequent MFs of the preceding hidden layer.

9.2.1.2 Transform TIMFs to IT2MFs

In the second step, we still train the FOU of the membership functions of the antecedents and the consequents using an optimization algorithm. And the FAE representation changes as follows.

$$T_e^n = T2_1^{n-1}, \dots, T2_i^{n-1}, T2_{i+1}^n, T2_{i+k}^n, R_1, \dots, R_l \quad (9.17)$$

Where $T2_i^{n-1}$ represents the IT2 MFs of the i^{th} input of FAE, which is shared with the i^{th} consequent of the $n-1$ FAE.

$$E2_n = T_e^n, R_1^d, \dots, R_m^d \quad (9.18)$$

Where T_e^n represents the parameters of the encoder of the n^{th} layer. Only the MFs and FOUs are optimized in this step, while the rules of the FAE are not modified.

Note that we do not train the antecedent MFs and FOUs, if there is a preceding hidden layer, in this step as they have been trained during the training of the preceding hidden layer.

9.2.1.3 Optimizing the Rule Base of the IT2 FAE

In the third step, there are no changes to the representation of the FAE in this step. We retrain the rules of the IT2 FAE using the same procedure described in section 6.2.2.3.

9.2.2 Optimization Method for the Final Layer

Similar to the method described in Section 6.2.3, to train the full D2FLS, we start by using the encoders of the FAE systems trained in the previous phase. We add another FLS that will act as the final output layer of the FLS. We use an optimization algorithm to retrain all the layers and like the optimization method used for training the FAE depicted in Figure 6-10, i.e., we train it in 3 steps. In this part of the training process, we use supervised training.

9.2.2.1 Optimize the Type 1 D2FLS

In this step, we train the final layer as a Type-1 FLS while at the same time retrain the MFs and rules of the encoders using an optimization algorithm. The MFs and rules of the two FLS are encoded in the following format.

$$E_{all} = N_e^1, \dots, N_e^n, M_{o+1}^f, \dots, M_{o+p}^f, R_1^f, \dots, R_n^f \quad (9.19)$$

Where N_e^n represents the membership functions, and Rules of the n encoders created using (9.15), M_{o+p}^f represents the membership functions for the p consequents of the final layer and R_n^f represents the n rules of the final layer. For the o inputs of the final layer, we use the consequent MFs of the preceding hidden layer.

9.2.2.2 Transform the TIMFs of the D2FLS into IT2MFs

In this step, we transform the type-1 MFs of the final layer into interval type-2 MFs by adding a FOU to each of the fuzzy sets. This is similar to the way we added the FOU while training the FAEs, and it is depicted in Figure 6-7. We also retrain the FOU of the encoder created during the training of the FAEs. This is encoded in the following format.

$$T_f = T_e^1, \dots, T_e^n, T2_{o+1}^f, \dots, T2_{o+p}^f, R_1^f, \dots, R_n^f \quad (9.20)$$

Where T_e^n represents the IT2 representation of the n encoders from (9.17) and $T2_{o+p}^f$ represents IT2 MFs for the p consequents of the final layer which are built using (6.13), (6.16) or (6.19) depending on the type of MF used. For the o inputs of the final layer, we use the consequent MFs of the preceding hidden layer

9.2.2.3 Optimizing the rule base for the D2FLS

In the final step, we retrain the rules of the encoder and final layer using the BB-BC algorithm. The parameters for this step are encoded in the following format to create the candidate solutions.

$$RL_{all} = R_1^{e1}, \dots, R_l^{e1}, \dots, R_l^{en}, R_1^f, \dots, R_n^f \quad (9.21)$$

Where R_l^e and R_n^f represent the l rules of the encoder and n rules of the final layer, respectively.

9.2.3 Extracting rules

To extract rules from the Enhanced D2FLS, we use the fact that the linguistic variables of the inputs of a layer of the D2FLS are constrained to use the linguistic

variables of the preceding layer. This means that the rules of the two layers can be combined by replacing each the linguistic variables of the antecedents of a layer of the D2FLS with the antecedents of the rules of the preceding layer with a consequent that matches the linguistic variable.

This can be explained more clearly using an example, in Table 9-3 and Table 9-4, we have the rules of the hidden and final layer of a two-layer D2FLS system. The hidden layer, in this case, has five outputs or consequents and the linguistic labels of the consequents of the hidden layer and the antecedent of the final layer are the same.

In this case, for the first rule in Table 9-4, the first antecedent of the rule High H03 represents the fourth input of the final layer with the linguistic label high. Since the linguistic label is shared with the consequents of the hidden layer, we can replace the High H03 with the compound input created from the rules that have a High consequent for the fourth output or consequent of the hidden layer. In this case, there are seven rules (fourth to seventh, eleventh, twelfth and fourteenth rules) that have the High consequent for the fourth output. We can choose the 4th rule in Table 9-3 to create the compound input.

$$\textit{High H03} = (\textit{Mid MSLCL2 0}, \textit{High CALC PROD 3}, \textit{Low RANK 0}) \quad (9.22)$$

Similarly, for the second antecedent, Low H02, we can replace it with the compound inputs created from the rules that have Low linguistic label for the third output of the hidden layer. In this case, there are three rules (second, third and ninth rules from Table 9-3).

$$\textit{Low H02} = (\textit{Low CONT 2}, \textit{High MSLCL1 1}, \textit{Low MSLCL2 3}) \quad (9.23)$$

In a similar fashion for the third antecedent, Mid H01, we can replace it with the compound inputs created from 12th and 13th rules of the hidden layer.

$$\text{Mid H01} = (\text{Mid MISSAPP 0, Low OT HOURS 3, High RANK 0}) \quad (9.24)$$

When we combine the three together we can rewrite the first rule of the final layer as IF (Mid MSLCL2 0, High CALC PROD 3, Low RANK 0) and (Low CONT 2, High MSLCL1 1, Low MSLCL2 3) and (Mid MISSAPP 0, Low OT HOURS 3, High RANK 0) THEN y is Very High.

Similarly, we can use the same method to rewrite all the rules of the final layer.

9.3 Experiments and Results

9.3.1 Comparison between a D2FLS pre-trained as an FAE vs Enhanced D2FLS pre-trained as an FAE

In this experiment, we compare the performance of the Enhanced D2FLS with the D2FLS pre-trained as an FAE. The goal of this experiment is to find the loss in performance of the Enhanced D2FLS due to the additional constraints that were added.

Table 9-1: Comparison of the D2FLS with the Enhanced D2FLS on Categorical Datasets with Average Recall as the fitness function

Data Set	Model	1	2	3	4	5	Average	Std
Santander CTP	D2FLS FAE	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	Enhanced D2FLS FAE	60.6	59.32	58.03	59.42	60.12	59.5	0.87
BT Customer Data	D2FLS FAE	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	Enhanced D2FLS FAE	60.16	63.23	62.95	60	60.6	61.39	1.41
PD Speech	D2FLS FAE	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	Enhanced D2FLS FAE	64.16	67.5	75.94	70.1	73.8	70.3	4.23
IDA2016	D2FLS FAE	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	Enhanced D2FLS FAE	89.89	94.04	91.37	92.96	90.51	91.75	1.54
Epi Seizure	D2FLS FAE	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	Enhanced D2FLS FAE	91.48	90.59	89.65	91.16	90.24	90.63	0.65

Table 9-2: Comparison of the D2FLS with Enhanced D2FLS on the Regression dataset with Mean average error as the fitness function

Data Set	Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS FAE	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	Enhanced D2FLS FAE	0.1166	0.1054	0.1315	0.1167	0.1063	0.1153	0.0094
Swiss Premium Pred	D2FLS FAE	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	Enhanced D2FLS FAE	0.0709	0.0641	0.0601	0.0469	0.0552	0.0594	0.0081
CT Scan Region Pred	D2FLS FAE	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	Enhanced D2FLS FAE	0.1105	0.1039	0.0949	0.0938	0.0951	0.0997	0.0065
Predict Song Year	D2FLS FAE	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	Enhanced D2FLS FAE	0.0785	0.0822	0.1038	0.0804	0.0781	0.0846	0.0097
BT PWA	D2FLS FAE	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	Enhanced D2FLS FAE	0.0554	0.0541	0.0452	0.0589	0.0558	0.0539	0.0046

We tabulate the results of the D2FLS FAE, and Enhanced D2FLS FAE trained five times on the classification datasets using BB-BC in Table 9-1. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the AI Model with the best performance for the dataset

From Table 9-1, we can see that the D2FLS performs better than the Enhanced D2FLS with about 3.6% higher performance on average in the classification datasets. With the Enhanced D2FLS performing the worst in the BT Customer Data dataset with about 10% lower performance. If we ignore the performance in the BT Customer Data

Table 9-3: Rules of the Hidden Layer of a D2FLS with five outputs

ID	Antecedents			Consequents				
	1	2	3	H00	H01	H02	H03	H04
1	High MSLCL2 0	Low MSLCL1 3	Low TRAVEL 0	High	High	High	Low	High
2	Low CONT 2	High MSLCL1 1	Low MSLCL2 3	High	High	Low	Low	Low
3	Low TRAVEL 1	Low OT HOURS 3	High MSLCL1 0	High	High	Low	Low	High
4	Mid MSLCL2 0	High CALC PROD 3	Low RANK 0	Mid	Low	High	High	Low
5	High CONT 2	Mid OT HOURS 3	Low RANK 0	High	High	High	High	Low
6	High LOANS 2	High RANK 3	High ECO UTIL 0	High	High	High	High	Mid
7	Low LOANS 0	High RANK 0	High MSLCL1 0	High	Low	High	High	Low
8	High MISSAPP 0	Low ECO UTIL 1	High ECO UTIL 3	High	High	Mid	Mid	Low
9	Low ECO UTIL 3	Low CONTRACTOR 3	Low OT HOURS 3	High	Low	Low	Low	High
10	Low CALC PROD 0	Low OT HOURS 3	Low RANK 0	Mid	High	High	Low	Mid
11	Mid OT HOURS 0	High LOANS 1	High ON DAY UTIL 3	Low	Low	High	High	Mid
12	Mid MISSAPP 0	Low OT HOURS 3	High RANK 0	Mid	Mid	Mid	High	High
13	High MSLCL1 0	High OT HOURS 3	Low ECO UTIL 3	Low	Mid	High	Low	Low
14	High MISSAPP 0	High MSLCL2 0	Mid ON DAY UTIL 3	Mid	Low	High	High	High
15	High MSLCL1 3	Low RANK 0	Low TRAVEL 1	Mid	Low	High	Low	High

dataset, then the average performance loss when using the Enhanced D2FLS is only about 2%.

We tabulate the results of the D2FLS FAE, and Enhanced D2FLS FAE trained five times on the regression datasets using BB-BC is tabulated in Table 9-2. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the D2FLS training method with the best performance for the dataset

From Table 9-2, we can see that the D2FLS performs better than the Enhanced D2FLS in all the datasets with about 13% lower error on average across the regression datasets.

A snapshot of the rule base generated for Enhanced D2FLS trained using FAE on the BT PWA dataset is shown in Table 9-3 and Table 9-4. Table 9-3 contains a snapshot of the rules (15 out of 100 rules) of the Hidden Layer of the Enhanced D2FLS. Membership functions for the first two of the antecedents of the seventh rule in Table 9-4 are depicted in Figure 9-7 (a) and (b). Table 9-4 contains a snapshot of the rules of the output layer of the D2FLS.

If we observe the seventh rule in Table 9-3 and compare the consequents of this rule with the antecedents of the seventh rule in Table 9-4. We can see that the linguistic terms of the three antecedents, High H00, Low H01 and High H02, match the linguistic terms of the consequents. From this example, we can observe that when LOANS 0 is Low and RANK 0 is High and MSLCL1 0 is High, the output of the Enhanced D2FLS (PWA performance) will be High. To come to this conclusion, we only needed to observe the rules. If we do the same exercise for the rules of the D2FLS FAE in Table 6-4 and Table 6-5. We need to do an additional check to make sure that the membership functions of the linguistic terms in these tables are similar.

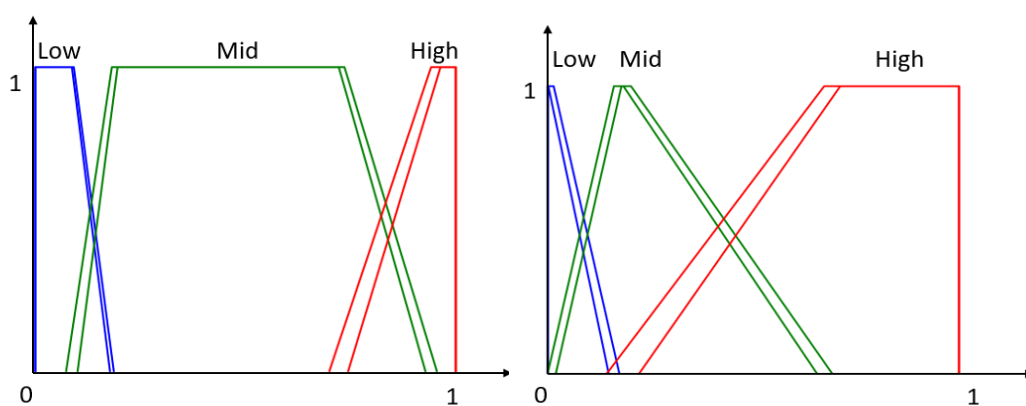


Figure 9-7: Fuzzy Sets Generated by Enhanced D2FLS FAE Training for (a) Loans 0 feature and (b) Rank 0 feature of the BT PWA dataset

Table 9-4: Rules of the Final Layer of a D2FLS

Rule No	Antecedents			Consequent
	1	2	3	0
1	High H03	Low H02	Mid H01	Very High
2	Mid H00	Mid H01	High H02	Mid
3	Low H00	Low H03	High H02	Very Low
4	Low H03	Mid H00	Mid H02	High
5	Mid H03	High H02	Low H00	Very High
6	Mid H03	High H02	High H00	High
7	High H02	High H00	Low H01	High
8	High H00	High H01	Low H03	Very High
9	Low H03	High H00	Mid H01	Low
10	High H03	Low H01	High H02	Very High
11	Low H00	Low H01	Low H02	Very High
12	High H00	Mid H02	High H03	Very High
13	High H03	High H02	Low H01	Very Low
14	Mid H01	Low H02	Low H00	Very Low
15	Low H00	Low H01	High H02	Very High

From these observations, we can see that the improved interpretability provided by the Enhanced D2FLS comes at the cost of about 3% loss in performance in the Categorical datasets and about 13 % higher error in regression dataset when compared to the D2FLS. The Enhanced D2FLS could be an option for situations in which the global interpretability of the Enhanced D2FLS is very important. And the Global Interpretability at the modular level provided by the D2FLS is insufficient.

9.3.2 Comparison between a D2FLS pre-trained as FAE and Enhanced D2FLS pre-trained using SAE

In this experiment, we compare the performance of the Enhanced D2FLS trained using SAE with the D2FLS pre-trained as an FAE. The goal of this experiment is to find if using SAE to pre-train the Enhanced D2FLS allows us to close the performance gap between the two models.

Table 9-5: Comparison of the D2FLS pre-trained using FAE with the Enhanced D2FLS pre-trained using SAE on Categorical Datasets with Average Recall as the fitness function

Data Set	Model	1	2	3	4	5	Average	Std
Santander CTP	D2FLS FAE	64.39	61.4	63.7	61.39	62.04	62.584	1.24
	Enhanced D2FLS FAE	60.93	57.92	60.51	60.35	63.62	60.67	1.82
BT Customer Data	D2FLS FAE	73.53	71.82	72.63	71.49	70.83	72.061	0.7
	Enhanced D2FLS FAE	61.29	62.01	61.46	61.39	60.27	61.28	0.36
PD Speech	D2FLS FAE	77.64	70.17	70.3	74.5	74.51	73.425	2.84
	Enhanced D2FLS FAE	74.61	77.29	70.46	71.11	68.14	72.32	3.24
IDA2016	D2FLS FAE	92.07	91.94	92.56	92.73	93.5	92.559	0.55
	Enhanced D2FLS FAE	91.64	94.18	94.65	93.44	93.28	93.44	1.02
Epi Seizure	D2FLS FAE	90.6	92.45	91.34	90.78	91.46	91.325	0.65
	Enhanced D2FLS FAE	91.23	90.96	91.83	91.01	91.48	91.3	0.32

We tabulate the results of the D2FLS FAE, and Enhanced D2FLS SAE trained five times on the classification datasets using BB-BC in Table 9-5. The results of the training runs are presented as Average Recall (equation (6.21)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the AI Model with the best performance for the dataset

From Table 9-5, we can see that the D2FLS performs better than the Enhanced D2FLS with about 2.5% higher performance on average in the classification datasets. With the Enhanced D2FLS performing the worst in the BT Customer Data dataset with about 10% lower performance. If we ignore the performance in the BT Customer Data dataset, then the average performance loss when using the Enhanced D2FLS is only about 0.5%. This shows that for Categorical datasets, the Enhanced D2FLS provides an alternative that performance as well as the D2FLS with slightly improved interpretability.

Table 9-6: Comparison of the D2FLS FAE with Enhanced D2FLS SAE on the Regression dataset with Mean average error as the fitness function

Data Set	Model	1	2	3	4	5	Average	Std
Wi-Fi Localization	D2FLS FAE	0.106	0.116	0.1049	0.1022	0.098	0.105	0.005897
	Enhanced D2FLS SAE	0.1538	0.1278	0.1373	0.1199	0.137	0.1352	0.0114
Swiss Premium Pred	D2FLS FAE	0.0534	0.0413	0.049	0.0459	0.0447	0.0469	0.004103
	Enhanced D2FLS SAE	0.0564	0.0501	0.056	0.0555	0.0524	0.0541	0.0025
CT Scan Region Pred	D2FLS FAE	0.0945	0.0911	0.0885	0.0843	0.0899	0.0897	0.003337
	Enhanced D2FLS SAE	0.1009	0.0944	0.0865	0.1071	0.0875	0.0953	0.0079
Predict Song Year	D2FLS FAE	0.076	0.0741	0.074	0.074	0.079	0.075	0.002247
	Enhanced D2FLS SAE	0.0785	0.0822	0.1038	0.0804	0.0781	0.0846	0.0097
BT PWA	D2FLS FAE	0.048	0.057	0.0511	0.045	0.057	0.0519	0.004737
	Enhanced D2FLS SAE	0.0581	0.0599	0.0577	0.0638	0.0592	0.0598	0.0022

We tabulate the results of the D2FLS FAE, and Enhanced D2FLS SAE trained five times on the regression datasets using BB-BC is tabulated in Table 9-6. The results of the training runs are presented as Mean Absolute Error (equation (6.24)) in columns (3-7). The mean and standard deviation of the five training runs is displayed in the eighth and ninth columns of the table, respectively. Where a result is in bold (column 8), it indicates that the row contains the AI Model with the best performance for the dataset

From Table 9-6, we can see that the D2FLS performs better than the Enhanced D2FLS with about 16% lower error on average in the regression datasets. With the Enhanced D2FLS performing the worst in the Wi-Fi Localisation dataset with about 28% higher error. If we ignore the performance in this dataset, then the D2FLS has about 12% lower error when compared to the Enhanced D2FLS.

From these observations, we can see that in categorical datasets at least the Enhanced D2FLS trained using SAE might be a viable alternative to the D2FLS FAE

even with the additional complexity of training an SAE due to its slightly higher interpretability.

9.4 Summary

This chapter discussed global interpretability and explained that it is challenging to achieve in practise when there are a large number of features in the input. It discussed alternatives such as SP-LIME, put forward to achieve global interpretability at the module level. Hence, the two local interpretability methods for extracting explanations from D2FLS proposed in the previous chapter were extended to provide Global interpretability at the module level. The modules are created by filtering the input or outputs using the linguistic variable that represents them to create a set of input-output pairs.

It also discussed another problem that might be encounter while trying to analyse the rules of the D2FLS. The problem is that the linguistic terms of outputs of one layer and the linguistic terms of the inputs of the next layer might not match. It proposed a method to enhance the D2FLS with the goal of simplifying the process of analysing the rules of the D2FLS. This is done by constraining the D2FLS during training to use the same membership functions for the consequents of a hidden FLS in the D2FLS and the antecedents of subsequent.

The D2FLS was compared against this enhanced version trained using FAE. The results show that the enhanced version has a 3% lower average recall in the categorical datasets. And a 13% higher error in the regression dataset. The D2FLS was then compared with the enhanced version trained using SAE. The results of this comparison show that the enhanced D2FLS SAE performs with 0.5% of the D2FLS in the

categorical datasets, with one exception. This shows that the Enhanced D2FLS SAE is a viable alternative to the D2FLS due to its superior readability of the rules even with the more complex training process.

Chapter 10. Conclusions and Future Work

In this thesis, we presented a novel Deep Type-2 Fuzzy logic system by combining the predictive accuracy and feature selection capabilities of Deep Learning with the interpretability of Interval Type-2 Fuzzy Logic System. The proposed model is built using easy to understand IF-Then rules that include linguistic labels similar to an Interval Type-2 FLS. This is one of the main benefits of the proposed model; that is, we could easily modify the system by changing the rules. Next, we presented two methods of training the model; the first training method uses greedy layer-wise training to train the model using both supervised and unsupervised data; the second training method uses a stacked autoencoder to pre-train the hidden layers of the model. Next, we proposed two methods for extracting local interpretable explanations from the model as the D2FLS rules might not be comprehensible to all audiences. Finally, we extended the local interpretability methods to provide global explanations at the modular level and enhanced the D2FLS by adding a constraint during training to improve the readability of the rules of the D2FLS.

10.1 Conclusions

The aims of the thesis were as follows

- To investigate and build an explainable AI model that is suitable for high dimensional datasets

This was achieved by building a deep algorithm using multiple layers of Interval type-2 fuzzy logic systems which are trained using the deep learning principle of greedy layer-wise learning. The model was then trained on eleven high dimensional datasets, and its performance was compared against several other AI models such

as Stacked Autoencoder (SAE), Convolutional neural network (CNN), Multi-Layer Perceptron (MLP) and Interval type-2 fuzzy logic system (IT2FLS).

In the experiments, it was found that the model achieved comparable performance to the deep models such as SAE and CNN. It outperformed the SAE by about 2% on average and performed within 2-3% of the CNN in the categorical datasets. It also outperformed both the MLP and IT2FLS by about 4% in the categorical datasets. In the regression datasets, the model performed slightly worse than the SAE, MLP and CNN models. It outperformed the IT2FLS with a 15% lower error in the regression datasets.

The interpretability/explainability of the model was evaluated by conducting a survey, where several subjects were asked to compare the explanations provided by the proposed model with the IT2FLS explanations, LIME and SHAP explanations for the SAE model. The results of the survey show that the explanations provided by the proposed model are highly interpretable, and the explanations are within 2% of the IT2FLS explanations. When compared to the LIME and SHAP model, the explanations were found to be about 20% and 17% better respectively.

The simple explanations provided by the model on the high dimensional datasets suggests that the model was able to inherently make feature selection thereby reducing the number of features in the explanations.

- To investigate the most suitable training method for the new explainable AI technique

This was achieved by examining two methods of training the D2FLS. The first one using a Fuzzy Autoencoder to pre-train the D2FLS using a Greedy Layer wise training

method. The second method used Stacked autoencoders to pre-train the hidden layers of the D2FLS. In the experiments, it was observed that the D2FLS FAE achieved 2% higher average recall when compared to the D2FLS SAE in classification dataset. In the regression datasets, the D2FLS FAE has about 6% lower MAE on average. The D2FLS SAE only performed better in the IDA 2016 dataset where it had 2% higher average recall.

Two types of training processes were also investigated; in the first training process, the D2FLS parameters for each of the layers were trained in three steps. In the first step, the D2FLS layer is trained as a Type-1 FLS, in the second step the FOU's are added to the MFs of the D2FLS layer and in the third and final step the rules of the D2FLS layer are retrained. In the second training process, all the parameters of the D2FLS layers are trained in a single step. In the experiments, it was observed that the three-step training process reduced the MAE by 47% on average across the regression datasets.

Hence, the D2FLS was trained as a Fuzzy Autoencoder in a three-step training process is the most suitable training method.

- To investigate the most suitable optimization algorithm for the new explainable AI technique

This was achieved by investigating meta-heuristic methods used to find near optimal solutions. The methods tested were Big Bang Big Crunch (BB-BC) and Genetic Algorithms (GA). A BB-BC and a GA were implemented to train the D2FLS. In the experiments, it was found that the D2FLS trained using BB-BC achieved 4.3% higher average recall in the categorical datasets and about 47% lower MAE in the regression

datasets. Further investigation revealed that GA requires a much larger number of generations to achieve similar fitness to the model trained using BB-BC. Hence, the BB-BC algorithm is more suitable for training the D2FLS.

- To investigate and develop local explanations that are understandable to all audience that might use these explanations

Two methods of extracting locally interpretable explanations from the D2FLS were presented. One method is built around the rules of the D2FLS; hence it is termed rule-based explanations. The method involves calculating the contributions of each of the rules in the D2FLS to the output. And based on these contributions, an explanation of how the output was predicted will be put forth. This provides a view into the inner workings of the D2FLS and might be suitable for experts. The second method is built around feature importance scores, where the explanation is a simple set of weighted values which show the relationship between the input features and the output, using linguistic variables. The simple formulas used to generate these explanations make it easy to verify the explanations provided by the model.

These explanations were evaluated by comparing them against popular Deep learning XAI tools LIME, SHAP and against explanations provided by an IT2FLS. This was done by conducting a survey in which three different audiences were asked to examine the explanations provided by these methods. The results of this survey show that the explanations provided by the D2FLS are comparable to the explanations provided by the IT2FLS (with only about 2% lower interpretability). It also shows that the explanations provided by D2FLS are better than the explanations provided by the tools, LIME and SHAP, about 20% and 17% better interpretability, respectively. At the same time, the performance of D2FLS model is about 2% higher than the SAE model

in categorical datasets and 40% higher error in the regression datasets. This is a small loss in performance for significantly higher interpretability making D2FLS a suitable replacement for the other AI models in applications where interpretability is paramount.

- To investigate and develop a global explanation that can provide a holistic understanding of the new Explainable AI Technique

The two local interpretability methods for extracting local explanations from D2FLS were extended to provide Global interpretability at the modular level. The modules are created by filtering the input or outputs using the linguistic variable that represents them to create a set of input-output pairs. These modules can be used to query the model and determine its behaviour based on the linguistic terms selected. This is a powerful tool for examining the behaviour of the model in a variety of situations. It is also helpful in analysing how changes in the input affect the output. Thus, giving a holistic understanding of the D2FLS model.

D2FLS was also enhanced to create combined IF-THEN rules to represent all the layers. This was done by constraining the D2FLS to use the same linguistic labels for the consequents or outputs of the hidden layer and the input features of the subsequent layer. The D2FLS was compared against this enhanced version. With the enhanced version having 3% lower average recall in the categorical datasets. And a higher error in regression dataset when trained as an FAE. And the enhanced version had about 2% lower average recall in the categorical dataset and higher error in regression datasets when the hidden layers are trained using an SAE.

10.2 Future work

In future work, I will explore extending the D2FLS to solve image, speech, and video classification problems. This could be achieved by combining convolutional neural networks with the D2FLS as CNNs are some of the best algorithms for image classification tasks. There is also the potential for extending the D2FLS for text classification using the method.

Next, I will explore methods to improve the D2FLS SAE training method as a similar training method used to train Fuzzy Stacked Autoencoders showed higher performance than the best D2FLS.

Next, I will explore the use of explanations provided by the D2FLS to determine how business functions can be improved/changed to get the desired outcomes. For example, if the model predicts customer satisfaction based on some business metrics. An analytical model can be built to analyse the global modular explanations to provide details about the input business metrics that can be improved/changed to get the desired level of customer satisfaction.

Additionally, there is a potential for examining single pass training methods to train the D2FLS to simplify the training process.

Bibliography

- [1] R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Explainable AI and Fuzzy Logic Systems," in *International Conference on Theory and Practice of Natural Computing*, 2018: Springer, pp. 3-20.
- [2] G. Nott, "Explainable Artificial Intelligence': Cracking open the black box of AI," *Computer World* <https://www.computerworld.com.au/article/617359>, 2017.
- [3] Z. C. Lipton, "The Mythos of Model Interpretability," *Queue*, vol. 16, no. 3, p. 30, 2018.
- [4] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA)*, nd Web, 2017.
- [5] "AI in the UK: ready, willing and able?," UK Parliament (House of Lords) Artificial Intelligence Committee,, 16 April 2017. [Online]. Available: <https://publications.parliament.uk/pa/ld201719/ldselect/ldai/100/100.pdf>
- [6] B. F. Goodman, Seth, "European Union regulations on algorithmic decision-making and a "right to explanation"," *2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*, New York, NY, 2016.
- [7] P. J. Phillips, C. A. Hahn, P. C. Fontana, D. A. Broniatowski, and M. A. Przybocki, "Four Principles of Explainable Artificial," 2020.
- [8] F. S. Board, "Artificial intelligence and machine learning in financial services," November, available at: <http://www.fsb.org/2017/11/artificialintelligence-and-machine-learning-in-financialservice/>(accessed 30th January, 2018), 2017.

- [9] R. Chatila and J. C. Havens, "The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems," in *Robotics and Well-Being*, M. I. Aldinhas Ferreira, J. Silva Sequeira, G. Singh Virk, M. O. Tokhi, and E. E. Kadar Eds. Cham: Springer International Publishing, 2019, pp. 11-16.
- [10] E. B. Authority, "EBA Report On Big Data And Advanced Analytics,"
- [11] F. Poursabzi-Sangdeh, D. G. Goldstein, J. M. Hofman, J. W. Vaughan, and H. Wallach, "Manipulating and measuring model interpretability," *arXiv preprint arXiv:1802.07810*, 2018.
- [12] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138-52160, 2018.
- [13] A. Preece, D. Harborne, D. Braines, R. Tomsett, and S. Chakraborty, "Stakeholders in explainable AI," *arXiv preprint arXiv:1810.00184*, 2018.
- [14] A. B. Arrieta *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *Information Fusion*, 2019.
- [15] H. Li, Y. Tian, K. Mueller, and X. Chen, "Beyond saliency: understanding convolutional neural networks from saliency prediction on layer-wise relevance propagation," *Image and Vision Computing*, vol. 83, pp. 70-86, 2019.
- [16] D. Doran, S. Schulz, and T. R. Besold, "What does explainable AI really mean? A new conceptualization of perspectives," *arXiv preprint arXiv:1710.00794*, 2017.
- [17] J. R. Zilke, E. L. Mencía, and F. Janssen, "Deepred–rule extraction from deep neural networks," in *International Conference on Discovery Science*, 2016: Springer, pp. 457-473.

- [18] M. Sato and H. Tsukimoto, "Rule extraction from neural networks via decision tree induction," in *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, 2001, vol. 3: IEEE, pp. 1870-1875.
- [19] M. G. Augasta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," *Neural processing letters*, vol. 35, no. 2, pp. 131-150, 2012.
- [20] A. D. Arbatli and H. L. Akin, "Rule extraction from trained neural networks using genetic algorithms," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 30, no. 3, pp. 1639-1648, 1997.
- [21] M. Affenzeller, S. Wagner, S. Winkler, and A. Beham, *Genetic algorithms and genetic programming: modern concepts and practical applications*. Crc Press, 2009.
- [22] M. W. Craven and J. W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *Machine learning proceedings 1994*: Elsevier, 1994, pp. 37-45.
- [23] Q. Zhang, Y. Yang, H. Ma, and Y. N. Wu, "Interpreting cnns via decision trees," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6261-6270.
- [24] M. W. Craven, "Extracting comprehensible models from trained neural networks," University of Wisconsin-Madison Department of Computer Sciences, 1996.
- [25] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez, "Beyond sparsity: Tree regularization of deep models for interpretability," *arXiv preprint arXiv:1711.06178*, 2017.

- [26] N. Frosst and G. Hinton, "Distilling a neural network into a soft decision tree," *arXiv preprint arXiv:1711.09784*, 2017.
- [27] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211-222, 2017.
- [28] Heatmap.org. "A Quick Introduction to Deep Taylor Decomposition." <http://www.heatmap.org/deeptaylor/> (accessed 18/08/2020, 2020).
- [29] M. Böhle, F. Eitel, M. Weygandt, and K. Ritter, "Layer-wise relevance propagation for explaining deep neural network decisions in MRI-based Alzheimer's disease classification," *Frontiers in aging neuroscience*, vol. 11, p. 194, 2019.
- [30] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.
- [31] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017: JMLR. org, pp. 3145-3153.
- [32] C. Molnar, *Interpretable Machine Learning*. Lulu. com, 2020.
- [33] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- [34] W. Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14-23, 2011.

- [35] L. Rokach and O. Z. Maimon, *Data mining with decision trees: theory and applications*. World scientific, 2008.
- [36] D. Heckerman, "A Tutorial on Learning With Bayesian Networks," *Studies in Computational Intelligence*, vol. 156, pp. 33-82, 2008, doi: 10.1007/978-3-540-85066-3_3.
- [37] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
- [38] J. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Springer, 2017.
- [39] Q. Liang and J. M. Mendel, "Interval type-2 fuzzy logic systems: theory and design," *IEEE Transactions on Fuzzy systems*, vol. 8, no. 5, pp. 535-550, 2000.
- [40] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," presented at the Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, California, 2016.
- [41] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-Precision Model-Agnostic Explanations," in *AAAI*, 2018, vol. 18, pp. 1527-1535.
- [42] M. T. Ribeiro, S. Singh, and C. Guestrin, "Nothing else matters: model-agnostic explanations by identifying prediction invariance," *arXiv preprint arXiv:1611.05817*, 2016.
- [43] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017, pp. 4765-4774.
- [44] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307-317, 1953.

- [45] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189-1232, 2001.
- [46] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation," *Journal of Computational and Graphical Statistics*, vol. 24, no. 1, pp. 44-65, 2015.
- [47] Z. Che, S. Purushotham, R. Khemani, and Y. Liu, "Interpretable deep models for ICU outcome prediction," in *AMIA Annual Symposium Proceedings*, 2016, vol. 2016: American Medical Informatics Association, p. 371.
- [48] R. Chimatapu, H. Hagraas, A. Starkey, and G. Owusu, "A Big-Bang Big-Crunch Type-2 Fuzzy Logic System for Generating Interpretable Models in Workforce Optimization," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2018: IEEE, pp. 1-8.
- [49] T. Garcia-Valverde, A. Garcia-Sola, H. Hagraas, J. Dooley, V. Callaghan, and J. Botia, "A fuzzy logic-based system for indoor localization using WiFi in ambient intelligent environments," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 4, pp. 702-718, 2013.
- [50] A. d. A. Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran, "Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning," *arXiv preprint arXiv:1905.06088*, 2019.
- [51] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," *Journal of Artificial Intelligence Research*, vol. 61, pp. 1-64, 2018.
- [52] A. Merentitis and C. Debes, "Automatic fusion and classification using random forests and features extracted with deep learning," 2015: IEEE, pp. 2943-2946.

- [53] Y. Yang, I. G. Morillo, and T. M. Hospedales, "Deep Neural Decision Trees," *arXiv preprint arXiv:1806.06988*, 2018.
- [54] P. Chen, C. Zhang, L. Chen, and M. Gan, "Fuzzy restricted Boltzmann machine for the enhancement of deep learning," *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 6, pp. 2163-2173, 2015.
- [55] M. Janmajaya, A. K. Shukla, T. Seth, and P. K. Muhuri, "Interval Type-2 Fuzzy Restricted Boltzmann Machine for the Enhancement of Deep Learning," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2019: IEEE, pp. 1-6.
- [56] M. S. Hosseini-Pozveh, M. Safayani, and A. Mirzaei, "Interval Type-2 Fuzzy Restricted Boltzmann Machine," *IEEE Transactions on Fuzzy Systems*, 2020.
- [57] Z. Zhang, M. W. Beck, D. A. Winkler, B. Huang, W. Sibanda, and H. Goyal, "Opening the black box of neural networks: methods for interpreting neural network models in clinical applications," *Annals of translational medicine*, vol. 6, no. 11, 2018.
- [58] Y. Deng, Z. Ren, Y. Kong, F. Bao, and Q. Dai, "A Hierarchical Fused Fuzzy Deep Neural Network for Data Classification," *IEEE Transactions on Fuzzy Systems*, vol. PP, no. 99, pp. 1-1, 2016, doi: 10.1109/TFUZZ.2016.2574915.
- [59] S. Park, S. J. Lee, E. Weiss, and Y. Motai, "Intra-and inter-fractional variation prediction of lung tumors using fuzzy deep learning," *IEEE journal of translational engineering in health and medicine*, vol. 4, pp. 1-12, 2016.
- [60] S. Rajurkar and N. K. Verma, "Developing deep fuzzy network with Takagi Sugeno fuzzy inference system," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017: IEEE, pp. 1-6.

- [61] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153-160.
- [62] S. Zhou, Q. Chen, and X. Wang, "Fuzzy deep belief networks for semi-supervised sentiment classification," *Neurocomputing*, vol. 131, pp. 312-322, 2014.
- [63] M. Wang and X.-S. Hua, "Active learning in multimedia annotation and retrieval: A survey," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 2, p. 10, 2011.
- [64] Y. Zheng, W. Sheng, X. Sun, and S. Chen, "Airline passenger profiling based on fuzzy deep machine learning," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 12, pp. 2911-2923, 2017.
- [65] R. R. Yager, "Pythagorean fuzzy subsets," 2013: IEEE, pp. 57-61.
- [66] H. Ishibuchi, K. Morioka, and I. Turksen, "Learning by fuzzified neural networks," *International Journal of Approximate Reasoning*, vol. 13, no. 4, pp. 327-358, 1995.
- [67] E. P. Klement, W. Schwyhla, and R. Lowen, "Fuzzy probability measures," *Fuzzy Sets and Systems*, vol. 5, no. 1, pp. 21-30, 1981.
- [68] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338-353, 1965.
- [69] A. J. Starkey, "Many-Objective Genetic Type-2 Fuzzy Logic Based Workforce Optimisation Strategies for Large Scale Organisational Design," University of Essex, 2018.
- [70] G. Klir and M. Wierman, "Uncertainty-Based Information," *Physica-Verlag, Heidelberg*, 1998.

- [71] J. M. Mendel, "Computing with words, when words can mean different things to different people," in *Proc. of Third International ICSC Symposium on Fuzzy Logic and Applications*, 1999, pp. 158-164.
- [72] N. R. Pal and J. C. Bezdek, "Measuring fuzzy uncertainty," *IEEE Transactions on Fuzzy Systems*, vol. 2, no. 2, pp. 107-118, 1994.
- [73] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Transactions on Fuzzy systems*, vol. 3, no. 3, pp. 370-379, 1995.
- [74] L.-X. Wang and J. M. Mendel, *Generating fuzzy rules from numerical data, with applications*. Signal and Image Processing Institute, University of Southern California ..., 1991.
- [75] J. M. Mendel, "Explaining the Performance Potential of Rule-Based Fuzzy Systems as a Greater Sculpting of the State Space," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 2362-2373, 2017.
- [76] C. Fu, A. Sarabakha, E. Kayacan, C. Wagner, R. John, and J. M. Garibaldi, "A comparative study on the control of quadcopter uavs by using singleton and non-singleton fuzzy logic controllers," in *2016 IEEE international conference on fuzzy systems (FUZZ-IEEE)*, 2016: IEEE, pp. 1023-1030.
- [77] E. H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE transactions on computers*, no. 12, pp. 1182-1191, 1977.
- [78] D. Driankov, H. Hellendoorn, and M. Reinfrank, *An introduction to fuzzy control*. Springer Science & Business Media, 2013.
- [79] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on fuzzy systems*, vol. 1, no. 1, pp. 7-31, 1993.

- [80] C. Wagner and H. Hagnas, "Toward general type-2 fuzzy logic systems based on zSlices," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 4, pp. 637-660, 2010.
- [81] J. M. Mendel and F. Liu, "On new quasi-type-2 fuzzy logic systems," in *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, 2008: IEEE, pp. 354-360.
- [82] S. Coupland and R. John, "Geometric type-1 and type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 1, pp. 3-15, 2007.
- [83] H. A. Hagnas, "A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots," *IEEE Transactions on fuzzy systems*, vol. 12, no. 4, pp. 524-539, 2004.
- [84] H. Hagnas, "Type-2 FLCs: A new generation of fuzzy controllers," *IEEE Computational Intelligence Magazine*, vol. 2, no. 1, pp. 30-43, 2007.
- [85] N. N. Karnik and J. M. Mendel, "Introduction to type-2 fuzzy logic systems," in *1998 IEEE international conference on fuzzy systems proceedings. IEEE world congress on computational intelligence (Cat. No. 98CH36228)*, 1998, vol. 2: IEEE, pp. 915-920.
- [86] D. Wu and W. W. Tan, "Type-2 FLS modeling capability analysis," in *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ'05.*, 2005: IEEE, pp. 242-247.
- [87] M. Nie and W. W. Tan, "Towards an efficient type-reduction method for interval type-2 fuzzy logic systems," in *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, 2008: IEEE, pp. 1425-1432.

- [88] J. M. Mendel and X. Liu, "Simplified interval type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 6, pp. 1056-1069, 2013.
- [89] D. Bernardo, H. Hagraš, and E. Tsang, "A genetic type-2 fuzzy logic based system for the generation of summarised linguistic predictive models for financial applications," *Soft Computing*, vol. 17, no. 12, pp. 2185-2201, 2013.
- [90] A. L. Buczak *et al.*, "Fuzzy association rule mining and classification for the prediction of malaria in South Korea," *BMC medical informatics and decision making*, vol. 15, no. 1, p. 47, 2015.
- [91] J. A. Sanz, D. Bernardo, F. Herrera, H. Bustince, and H. Hagraš, "A compact evolutionary interval-valued fuzzy rule-based classification system for the modeling and prediction of real-world financial applications with imbalanced data," *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 4, pp. 973-990, 2014.
- [92] T. Garcia-Valverde, A. Garcia-Sola, H. Hagraš, J. A. Dooley, V. Callaghan, and J. A. Botia, "A fuzzy logic-based system for indoor localization using WiFi in ambient intelligent environments," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 4, pp. 702-718, 2013.
- [93] L.-X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on systems, man, and cybernetics*, vol. 22, no. 6, pp. 1414-1427, 1992.
- [94] L.-X. Wang, "The WM method completed: a flexible fuzzy system approach to data mining," *IEEE Transactions on fuzzy systems*, vol. 11, no. 6, pp. 768-782, 2003.
- [95] O. Erol and I. Eksin, "A new optimization method: big bang–big crunch," *Advances in Engineering Software*, vol. 37, no. 2, pp. 106-111, 2006.

- [96] R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Interval type-2 fuzzy logic based stacked autoencoder deep neural network for generating explainable ai models in workforce optimization," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2018: IEEE, pp. 1-8.
- [97] K. Y. Lee and P. S. Mohamed, "A real-coded genetic algorithm involving a hybrid crossover method for power plant control system design," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, 2002, vol. 2: IEEE, pp. 1069-1074.
- [98] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of parallel and distributed computing*, vol. 47, no. 1, pp. 8-22, 1997.
- [99] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
- [100] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [101] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [102] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [103] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26-31, 2012.
- [104] "UCI Machine Learning Repository," ed, 2017.

- [105] *Santander Customer Transaction Prediction* (<https://www.kaggle.com/c/santander-customer-transaction-prediction>), Banco Santander,
- [106] *Swiss healthcare premium prediction* (<https://www.kaggle.com/comparisdata/premium-prediction>),
- [107] J. P. Mallm *et al.*, "Linking aberrant chromatin features in chronic lymphocytic leukemia to transcription factor networks," *Molecular systems biology*, vol. 15, no. 5, 2019.
- [108] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.
- [109] C. O. Sakar *et al.*, "A comparative analysis of speech signal processing algorithms for Parkinson's disease classification and the use of the tunable Q-factor wavelet transform," *Applied Soft Computing*, vol. 74, pp. 255-263, 2019.
- [110] J. Torres-Sospedra *et al.*, "UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems," in *2014 international conference on indoor positioning and indoor navigation (IPIN)*, 2014: IEEE, pp. 261-270.
- [111] F. Graf, H.-P. Kriegel, S. Pölsterl, M. Schubert, and A. Cavallaro, "Position prediction in ct volume scans," in *Proceedings of the 28th International Conference on Machine Learning (ICML) Workshop on Learning for Global Challenges, Bellevue, Washington, WA*, 2011.
- [112] F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, and A. Cavallaro, "2d image registration in ct images using radial image descriptors," in *International*

- Conference on Medical Image Computing and Computer-Assisted Intervention*, 2011: Springer, pp. 607-614.
- [113] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," 2011.
- [114] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [115] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929-1958, 2014.
- [116] R. Chimatapu, H. Hagra, A. Starkey, and G. Owusu, "Interval Type-2 Fuzzy Logic Based Stacked Autoencoder Deep Neural Network For Generating Explainable AI Models in Workforce Optimization," presented at the 2018 IEEE International Conference on Fuzzy Systems (FUZZ), in press.
- [117] L. Magdalena, "Semantic interpretability in hierarchical fuzzy systems: Creating semantically decouplable hierarchies," *Information Sciences*, vol. 496, pp. 109-123, 2019.
- [118] L. A. Zadeh, "Is there a need for fuzzy logic?," *Information sciences*, vol. 178, no. 13, pp. 2751-2779, 2008.
- [119] C. Mencar, C. Castiello, R. Cannone, and A. M. Fanelli, "Design of fuzzy rule-based classifiers with semantic cointension," *Information Sciences*, vol. 181, no. 20, pp. 4361-4377, 2011.