Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Extracting logic rules from neural networks with discrete weights

Armin Hadžić

Supervisor: doc. Ing. Tomáš Pevný, Ph.D.
Field of study: Open Informatics
Subfield: Cyber Security
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Hadžić Armin**	Personal ID number:	**516468**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Cyber Security**		

II. Master’s thesis details

Master’s thesis title in English:

Extracting logic rules from neural networks with discrete weights

Master’s thesis title in Czech:

Extrakce logických pravidel z neuronových sítí s diskrétními váhami

Guidelines:

One of directions to decrease the computational complexity of neural networks is to decrease precision of weights and activation to as few as one bit. This opens door to represent the neural network as a very large set of logical rules. This work will explore strategies to prune the ruleset to keep the number of rules small.
Instructions.
1. Read the prior art on extraction logic rules from neural networks.
2. Study, how to convert discrete neural networks to logic rules.
3. Implement selected strategies to prune the number of logic rules.
4. Measure the quality of ruleset obtained by different pruning strategies.

Bibliography / sources:

Fu, LiMin. "Rule generation from neural networks." IEEE Transactions on Systems, Man, and Cybernetics 24, no. 8 (1994): 1114-1124.
Towell, Geoffrey G., and Jude W. Shavlik. "Extracting refined rules from knowledge-based neural networks." Machine learning 13 (1993): 71-101.
Burkhardt, Sophie, Jannis Brugger, Nicolas Wagner, Zahra Ahmadi, Kristian Kersting, and Stefan Kramer. "Rule extraction from binary neural networks with convolutional rules for model validation." Frontiers in artificial intelligence 4 (2021): 642263.

Name and workplace of master's thesis supervisor:

doc. Ing. Tomáš Pevný, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **09.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Ing. Tomáš Pevný, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I want to thank my supervisor, Associate Professor Doc. Ing. Tomáš Pevný, PhD, for the constant help and support throughout my work on this thesis.

Declaration

I declare that I have prepared this thesis independently and that I have cited all sources and literature used.

In Prague, 24. May 2024

Abstract

Given a discrete deep neural network with binary activations and ternary weights, each neuron can be expressed as a logic expression, either an M-of-N rule or a predicate. This logical form is suitable for the abductive explanation of neural networks, where the explanation is a classification rule. Since abductive rule extraction is at least NP-hard, this work focuses on approximate methods of varying complexity. The thesis presents a method which uses activation statistics of individual neurons on class and non-class data determined by the quantised network to estimate neuron importance and extract sufficient rules by network pruning. The methods are experimentally compared.

Keywords: quantised neural networks, interpretability, rule extraction

Supervisor: doc. Ing. Tomáš Pevný, Ph.D.

Abstrakt

V diskrétních neuronových sítich s binárními aktivacemi a ternárními váhami lze každý neuron vyjádřit jako logický výrok ve formě M-of-N pravidla nebo predikátu. Tato logická forma je vhodná pro abduktivní vysvětlování neuronových sítí, kdy vysvětlení je klasifikační pravidlo. Vzhledem k tomu, ze extrakce abduktivních pravidel je nejméně NP-těžká, tato práce se zaměřuje na aproximativní metody s různou složitostí. Metody jsou experimentálně porovnány.

Klíčová slova: kvantované neuronové sítě, interpretovatelnost, extrakce pravidel

Překlad názvu: Extrahování logických pravidel z neuronových sítí s diskrétními vahami

Contents

Figures

Tables

May 24, 2024

Chapter 1

Introduction

The thesis proposes and presents a method for providing classification explanations for deep neural networks with discrete weights. Deep neural networks are widely popular today, offering immense predictive power, but they suffer from being black boxes. The thesis aims to develop a method that uses a deep neural network expressed as logic rules and extracts sets of rules that provide classification explanations.

The second chapter introduces prior arts in the field of AI, specialising in providing explanations for predictions made by different AI models. Numerous different techniques and creative ideas have been devised to gain insight into the decision mechanism of different AI models. The solution proposed in subsequent chapters uses logical rules' expressive and inference power to deliver the explanation. It has to compete with conventional and contemporary approaches in rule learning, like a decision tree. Hence, a detailed overview of methods of directly learning rules from data is provided.

The third chapter introduces the NuLog method, which requires a trained binarised neural network. The chapter presents quantised neural networks in a general manner and the different techniques used to train them. It goes over the specific implementation of quantised neural networks and their special binarised version, with binary activations and ternary weights. The chapter discusses feature quantisation as an integral part required by discrete neural networks. After the binarised networks are introduced, expressing such network's neurons as logic, specifically predicates and M-of-N rules, and converting it into a logical neural network is covered. A theoretical introduction and complexity analysis of the problem being dealt with to extract the valid explanatory rule. Defines the general problem of extracting rules from a neural network and the necessary conditions and implications which must hold for such extracted rules. Everything is accompanied by simple examples illustrating the procedures.

The fourth chapter describes how the proposed rule extraction is implemented. It clarifies each phase of the devised solution in the NuLog method with implementation details and time complexity analysis. Pseudo-codes of the procedures are provided for easier understanding.

The fifth chapter provides the results of experiments conducted on two classification problems, with outcomes and effects of the developed solution. The NuLog method is evaluated and compared to two contemporary methods, CART and RIPPER. Generated rules by all strategies are evaluated by qualitative and quantitative metrics, like total number of rules, rule length on average and average rule scope. Evaluation is done on two interesting datasets, MNIST and the generated flower dataset, both of which are used for classification problems. Nice visual representations of the extracted rules are provided for the used data sets.

The sixth and final chapter contains the conclusions drawn during the development and evaluations of the datasets. Lists the benefits and drawbacks of the developed methods and where areas for improvement lie. The chapter provides ideas for further exploration and research.

1.1 Motivation

AI is becoming an integral part of our daily lives and exerts a great influence on us as ever greater reliance is being placed on AI, from education and entertainment industries to essential and critical sectors like healthcare, security, defence and finance [46]. Decision-making processes in these fields have a profound and enduring impact; therefore, everybody's greatest interest is to have the best possible judgement [65]. AI systems are expected to make all decisions and resolve deadlocks on our behalf but to be able to achieve the level of trustworthiness in humans to delegate decisions, it necessitates explaining its decision-making process [8]. For example, what type of medical treatment is best or whether a program is malicious or not? For medical purposes, clinical decision rules are more followed if the decision has an explanation with an inherent logic that can be interpreted and deduced to make medical sense, and such decisions are more likely to be followed and implemented by the staff [49]. It holds for any use case in which AI models could be employed, but as humans will be reaping the benefits or suffering the consequences of those decisions, it is only natural and human to ask, "Why?" "Why is this condition diagnosed as benign?" "Why is this transaction labelled as fraudulent?" [65] [15].

Taking a detailed look into the world of cybersecurity, AI is being used to address many varying problems and threats, spanning from anomaly and intrusion detection over zero-day attacks to deepfakes [6]. Zero trust is a cybersecurity standard with the premise that trust is never granted by default, but it must be constantly validated and certificated [76]. Developing any method without understanding how it works or the ability to formally verify the results and decision-making processes opens it up to potential vulnerabilities and malicious use cases [35]. Complete or even high testing coverage of an AI model is not efficient or even realistically achievable, which is a consequence of high complexity. Security provided by an AI solution cannot be depended on and used, assuming zero trust principle, because it cannot be guaranteed without formal evaluation by knowing the reasoning behind the decisions [35]. If the adversary discovers a potential way to fool and bypass the protection, like an AI-based firewall, he can disguise malicious traffic as benign and avoid detection [39]. Possibly, even in this case, he has no access to the AI model itself, just its inference results [63]. The same issues exist as with any non-AI-based solution with a bug or feature opening a vulnerability for zero-day attacks.

The best examples to highlight this importance are adversarial examples, with detailed definitions provided later in the thesis, here 2.1 in chapter 2. Examples of stickers targeting image classifiers processed by models, influencing their decisions, but which are not noticeable to humans [13] [77]. An attack would be to add a sticker with specifically designed noise and place it on or next to the road such that an autonomous vehicle would detect it and process it. The sticker could then influence the decision-making mechanism to cause the car to swerve to the side on a straight road or speed up and cause a disaster [83]. That is exactly why AI detectors and decision makers require to be debugged and formally verified, which is something you cannot do with models that provide no direct interpretability, like black-box oracles [56]. By providing explanations for decision-making and a way to decide if the logic behind it is sound, justified and reasonable or not, debugging is possible [56].

Discrete weighted networks are extensively investigated as they offer a smaller memory footprint and reduced inference time compared to ones with continuous weights [75] [31]. Discrete neural networks are developed with edge devices in mind, which cannot afford huge amounts of hardware resources to store the classifier locally, and any decision must be delegated to a central server over the Internet, which takes time [62]. If a network intrusion detector has to consult the servers to determine whether some suspicious behaviour is justified for raising the alarm, precious time is lost on getting the answer [62]. Furthermore, such communication could be targeted and sabotaged. The ideal scenario would be to have a classifier on the edge locally that can be interpreted and its decisions explained.

The motivation behind the decision to use discrete weighted neural networks is these weights are ideal for expressing neuron activation functions as logic rules, specifically as M-of-N rules and predicates. Networks with continuous weights can also be expressed as M-of-N rules [78], but discrete weights offer numerous advantages, as stated above. The conversion process from discrete weights to M-of-N rules is straightforward and described in chapter 2 here 3.2. These rules are the basis of the rule extraction method proposed in the thesis.

Chapter 2

Prior works

The chapter briefly overviews and introduces prior arts in Explainable Artificial intelligence. It gives information regarding different definitions of explanations, interpretations, and metrics for comparison and evaluation. The chapter dwells on different methods for gaining an explanation related to various machine learning models, whether they are interpretable or black boxes, with particular interest in neural networks. The main emphasis is on analysing rules as prediction explanations and methods for their learning and extraction to compare contemporary approaches to the solution proposed in this thesis, which is based on extracting rules from a neural network.

2.1 Explainable Artificial Intelligence

Explainable Artificial Intelligence (XAI) is a field of artificial intelligence that develops techniques which provide users with explanations for decisions made by AI models. An explanation is preferably simple and understandable, while the performance and predictive power of explanation-based models are not greatly diminished compared to the AI model being explained [1]. The first introduction of the term XAI occurred in [47], depicting the system's capacity to generate AI-controlled entity behaviour explanations within the context of simulation game applications [1]. Another term used alongside explainable is interpretable. There exist multiple different definitions of these terms. The interpretable method is "if a user can correctly and efficiently predict the method's results" [44]. These are the definitions set by the International Standards Organisation [40]: **explainability** refers to the "level of understanding of how the AI-based system came up with a given result", **interpretability** is the "level of understanding how the underlying (AI) technology works". Other valid terms are understandability and comprehensibility. In essence, the point is to be able to "take a look under the hood", recognise and fix an issue, as well as have a formal validation and verification for attained results.

An abductive explanation is a reasoning form in philosophy and science, along with deductive and inductive, inferring the most likely hypothesis to explain the attained result [38]. Given an ML model $\mathcal{M}$ and prediction π, an **prediction explanation** is a subset-minimal set of literals $\mathcal{E}$ representing distinct, influential features and their values such that $\mathcal{E} \models (\mathcal{M} \rightarrow \pi)$ [39] [38]. **Example-based** explanations are simple and straightforward; they select the best representatives of examined data, who capture most of the features of a black box model [56]. Types of examples [56]: **counterfactual** (how a sample has to change to alter its prediction, explains how the model predicts), **influential** (exert the biggest influence on the prediction model's parameters), **prototypes** (representative samples of a population subset), **criticisms** (samples with poor representation by prototypes). **Counterexample** is a subset-minimal set $\mathcal{C}$ of literals to a prediction π if $\mathcal{C} \models (\mathcal{M} \rightarrow \pi)$ where p is a prediction and $p \neq \pi$ [39]. **Adversarial examples** are counterexamples with minimal modifications to feature values w.r.t to a predefined distance measure, such that misclassification is caused [39]. Adversarials are particularly interesting from the cybersecurity point of view regarding attacks on black-box models by minuscule alterations to samples causing a different classification, like single-pixel attacks and adversarial patches [33] [63] [77] [13]. Case-based reasoning uses experiences based on examples and similar cases to adapt existing solutions instead of building them from the ground up based on general knowledge and predefined rules[79].

Explanations can be modelled in different forms requiring defining metrics for explanation comparison to differentiate a lousy explanation from a good one and achieve a proper level of interpretability and explainability. Any explanation should correlate feature values of an instance to its model prediction in a natural and intuitive fashion [56]. Properties by which explanations and their methods can be compared are defined in [70]. **Expressive power** is the explanation structure (natural language, IF-THEN-ELSE rules, value ranges). **Portability** is the set of models the method applies to (surrogate models high, neural networks low). **Translucency** is the method's reliance on the inner workings of a system, a tradeoff between transparency and portability (linear regression high, data perturbation none). **Algorithmic complexity** is the computational complexity of the explanation generation. Qualities of individual explanations are defined in [70], as the following:

- **Accuracy**: performance predicting unseen data
- **Fidelity**: approximation of black box model's predictions
- **Consistency**: differences between explanations of models trained for the same task, with similar predictions
- **Stability**: similarity for similar instances
- **Comprehensibility**: human understanding of explanations
- **Certainty**: reflecting the certainty of the ML model
- **Degree of importance**: evaluating feature importance
- **Novelty**: reflecting if a sample is from the same distribution as training data
- **Representativeness**: number of instances explanation explains

Explanations frequently take the form of **IF-THEN-ELSE** logic structures representing the reasoning behind a decision; a formal definition of a rule is given later in chapter 2.2. Influential examples are also often used and chosen by methods performing feature importance, which indicates features greatly impact decisions, but such examples tend to suffer from the Rashomon effect [56]. The Rashomon effect is a storytelling term that depicts multiple contradictory explanations of the same outcome, which can happen if a method providing explanations is non-deterministic and highly unstable due to randomisation [48]. It can be stated that not all machine learning models are created equal. Models can be separated into two distinct groups: the interpretable group, which by design or as a side effect offers the user an explanation for their predictions, and the uninterpretable group, black-box models, which do not provide any interpretation. Explanations derived from interpretable models are modelled directly based on their parameters and structure, whereas highly specialised techniques are necessary to identify them from black-box models.

Interpretable models. Machine learning models with "out of the box" interpretability are linear and logistic regression models, decision rules, decision trees, naive-Bayes and k-NN [74]. **Linear regression** is a weighted sum of features, and linearity makes interpretation straightforward, weights corresponding to prediction change for a unit change of a feature, with many techniques for feature importance estimation [36]. **Logistic regression** is an adaptation of linear regression for classification problems [36]. **k-NN** algorithm bases its decision-making on majority voting or the average outcome of class samples on a predefined neighbourhood [36]. **Decision trees** are hierarchical structures with a series of decision nodes, explanations provided by traversing the tree and modelling a rule from nodes. Random decision forests are sets of decision trees. Many models are considered interpretable , but provided explanations are weighted, probabilistic or rely on fuzzy logic, which is not ideal since the explanations are vague, less understandable and intuitive. They can be expressed as fuzzy logic rules, which are difficult to understand [3]. **Naive-Bayes** [2] is a linear classifier based on Bayes's theorem, providing a probabilistic estimation of feature importance. **Rule-Fit** [25] algorithm combines linear regression with decision trees to overcome correlated features, while **SIRUS** [10] technique operates on random forests, both of which result in generating weighted decision rules.

Interpreting black-box models. Black-box models are models without the benefit of direct interpretability and can be understood indirectly. Many different approaches currently exist, with varying degrees of success, usually performing model sensitivity, component analysis or using surrogate models [57]. Other methods focus on getting a local explanation by targeting a single instance and inspecting how well the explanation describes it and how well it generalises for its neighbours [57].

The model-agnostic approach separates processed data from the model processing it, explaining the underlying nature of the data by simpler interpretable models, referred to as surrogates, using the results to explain more complex black-box models [56]. The main advantage is the flexibility of models, explanations, and representations, which is not achievable with a model-specific approach that provides highly specialised methods [67]. Model agnosticism creates a pipeline, starting with a real-world process, captures the data, predicts targets with a robust black box model, interprets the same data with a less powerful interpretable model and ends with humans using the black box results with the explanations provided by a less powerful tool [56]. Global model agnostic methods explain average behaviours and help understand the bigger picture of how a model operates [56]. Local interpretable model-agnostic explanations [68] (**LIME**) is a strategy for local explanations, working on a subset of samples with a specific prediction and training a surrogate to explain it. Applicable for different data formats, tabular, text and images [56].

As mentioned earlier, many methods for result interpretation perform feature importance estimation to identify which features of the model are sensitive enough to change a decision and identify influential examples. **Functional decomposition** is a procedure decomposing complex functions into simpler parts, a sum of individual and interaction effects interpreted individually by different strategies [56]. **Dependence plots** and **local effects** are methods indicating marginal feature impacts on the targeted outcome, discovering input-output relationship nature [24] [7]. Dependency plots are a valuable tool for measuring and representing feature importance [34]; the higher the variance in the plot, the higher the feature's importance and influential examples are easier to isolate. **Permutation feature importance** is the idea of permuting the feature's values between samples, measuring and testing the sensitivity to error change [23]. Processing data, in this fashion, breaks relationships between features; if the error then increases, the feature is deemed important. A direct parallel to determining feature importance is the concept of coalitional games from the field of game theory. Coalitional games model the problem of fair division of the prize among winning coalition players[71], with features as players and prediction being the prize, providing axiomatical foundations behind the generated explanation[56]. **Shapley values** specify the fair payout distribution or, in the context of interpretability, feature importance for a given sample[52].

Interpreting neural networks. Neural networks offer great prediction performances and are a prevalent approach for flexibility and scalability, but they are black boxes. Model-agnostic methods can be utilised for neural network interpretation, but far better results are achieved if the network and neuron specifics are examined and used for interpretation [56]. Most methods are related to image detection and classification problems, where convolutional neural networks (CNN) are used. Explanations are commonly found by some form of feature importance analysis and presented as images or heat maps indicating important segments of the image that influence a decision. Provided explanations give an insight into the network's "thinking" process but can be suffering from the Rashomon effect, defined above 2.1 and could provide an illusion or false impression of interpretability [56]. The illusion of interpretability stems from the fact that neural networks are immensely complex and incredibly difficult to draw conclusions based on small samples. **Feature visualisation** [60] marks significant features by finding inputs maximising different unit (neuron, layer, channel) activation functions, uncovering what image parts the network is sensitive to. **Network dissection** [9] locates highly activated areas of CNN channels and links them with human concepts visible and labelled in processed images, like a flower, tree, dog, or cat. **Saliency maps** [73] are constructed by importance evaluation of individual pixels for classification and, depending on estimation strategy, can be perturbation-based or gradient-based. The gradient approach uses the gradient of the class score function with respect to input pixels, while perturbations introduce small input changes to images and observe error changes [32]. Feature importance for image classification heavily relies on individual pixels, which, by themselves, are meaningless. **Concept detecting** discovers larger concepts based on colours or shapes and measures their influence on the prediction, indirectly explaining it [56]. A popular method is **TCAV** [45], which measures the conceptual sensitivity of a single sample or the entire class.

The approach examined in the thesis relies on exploiting the specific structure of neurons with discrete weights to express them as rules and ultimately extracting decision rules from a neural network employing an importance estimation strategy for individual units. Emphasises the significant importance placed on rule learning overview.

2.2 Rule learning

Rule-based models are a subset of machine learning models distinguished by their notable ease of interpretability [28]. Rules are expressed as IF-THEN-ELSE statements, and their interpretability stems from pattern and irregularity search in the processed data [16].

Rule is a predefined statement dictating how a system processes input data to deliver the output and consists of a head and a body [29]. The head and body naming convention is used in rule systems based in PROLOG [51]. The head is the left-hand side or IF part of a rule, indicating conditions that all must be satisfied for the rule to be triggered or activated, as per the Domination law 2 in boolean algebra. The head is represented as a set of premises or independent boolean predicates which form terms in conjunction with one another. However, in the case of M-of-N rules, at least M out of all N terms must be true to evaluate true. The body is the right-hand side, the consequence or the THEN part of the rule, specifying outcomes of inference and activation. In the case of classification, it holds the predicting class label. Predicates specify a precise value within a specific categorical feature or its membership in a discrete set of values. For continuous feature analysis, predicates evaluate the inclusion of said features within predefined value intervals. **Anchor** of a prediction is a scoped rule for which any change in the sample's feature values within the scope does not change the original prediction [69]. Coverage, scoping or rule support is the percentage of the population for which the relationship is relevant, and the rule's head and body are satisfied [29]. Rule confidence measures the relationship between the head and body; it is the percentage of the population that satisfies the rule's body, satisfying the rule's head simultaneously [29]. The advantage of rules is the ease of interpretation and direct conversion to natural language [58]. Unfortunately, rule-based classifiers, employed as surrogates, suffer from numerous hyperparameters and inefficiency as they require many black box model calls. Rule learning techniques have to deal with the problem of quantising real-valued features needed to model predicates for a rule head; significant ranges are needed as strict equality for real-valued features limits their predictive power [31]. A number of rule-learning techniques only work with discrete values like CORELS 2.2, so data must be transformed before training by a quantisation strategy, which defines the number of bins and bin ranges for data quantisation.

As the classifying rules are scoped, a single rule cannot be expected to cover all of the class data; frequently, a rule can cover only a single sample, like extreme edge cases or outliers [69]. In other words, a single rule is often insufficient to provide a complete explanation of class data for complex problems since rule learning can easily overfit, and multiple rules must be combined [29]. Rule sets, and decision lists are quintessential for a complete rules-based classifier, which must fully understand the entire population. **Rule set** is a list of rules without any hierarchy or order, meaning everybody has the same power and importance; every rule must be evaluated to obtain an answer and follow the consequences [29]. In using rule sets, two issues arise: activation of multiple rules, necessitating additional rules to solve deadlocks for classification, and no rule activation, again demanding additional default rules [29]. These cases describe ambiguous classifications which can occur and reduce the classifier's accuracy. **Decision list** has a predefined hierarchy and levels of importance, so classification is done such that the first rule that evaluates true indicates the class [29]. Decision lists remove the necessity for additional default and tie-breaking rules but can be biased and depend on the correct ordering of rules. Two major approaches to rule learning exist: descriptive and predictive rule learning.

Descriptive rule learning is an unsupervised machine learning approach to rule learning, working towards describing important patterns and critical features in the examined dataset [29]. The idea behind them is to find rules describing the actual data's nature as best as possible, not focusing on predictive performance as a classifier. There are two major approaches: subgroup discovery and associative rules [80] [37]. **Apriori** is the representative algorithm, which finds associative rules by searching for the most frequent item sets and then extracts a rule which covers all items of a set [30]. While the approach is generally intriguing, the primary focus is on the extraction of rules for predictions, relying on supervised learning and labelled data. One idea for a part of the solution presented in this thesis is adapted and modified from the **Top-Down Hill-Climbing** algorithm from descriptive rule learning. It is a greedy algorithm finding a single rule, starting from an empty one and building it up until finding the best rule by a specific heuristic, generally maximising class and minimising non-class coverage. The pitfall of this approach is that it can get stuck in local optimums [27].

Predictive rule learning is an approach for dealing with supervised machine learning, predicting labelled data [29]. Emphasis is placed not on discovering the nature of data being processed but on exact predictions of discrete outcomes. The goal is to learn as few rules as possible from training data, correctly discriminate between classes, and perform on unseen data as best as possible. Predictive rules are generally learned as descriptive rules, but with additional constraints like rules must not misclassify on the dataset. **Classification by association** operates on learning associative rules, finding frequent patterns and selecting those indicating the target class [29]. Popular algorithms in this category are **CBA**, **CMAR** and **FARC-HD** algorithms based on the Apriori algorithm 2.2, introduced above, but vary in how the identification of frequent sets work. The methods are only applicable to features with discrete values; real-valued features must be quantised as described above 2.2. FARC-HD solves the problem using fuzzy logic to alleviate this issue [29], but as discussed earlier 2.1, fuzzy logic reduces the rule's interpretability. **Covering algorithms** are a divide-and-conquer or divide-and-explain strategy, and in a nutshell, consist of learning a single rule, removing all the samples it explains or covers and repeating the process until all data is explained or stopping criteria is met[29]. The solution examined in the thesis falls under this category. **AQ** is the oldest and original covering algorithm, adopting a top-down beam search, finding only the seed example's specialisation, **CN2** is a beam search-guided overfitting reduction estimate and **Opus** is the first to learn rules that maximise a certain quality or heuristic [29]. The solution proposed in this thesis belongs to this category of algorithms; the only difference is that the rules will not be learned and extracted from the data, but the neural network is the one learning, and rules will be extracted as prediction explanations for the said network.

Decision trees are interpretable models as defined above 2.1. Methods to learn trees employ different strategies to fit the data by forming a hypothesis growing the tree, possibly overfitting. Certain methods prune and simplify the tree to improve predictive power and reduce the query costs [16]. A popular decision tree method is Classification and Regression Tree [49] (**CART**), which works based on binary recursive partitioning, repeatedly dividing the population into two groups by alternating feature predicates.

Reduced Error Pruning (**REP**) is a rule-learning algorithm where, in training, data is split into a growing and pruning set, and an overfitting rule is learned from a growing set, which is then simplified by pruning such that the error is maximally reduced [16]. Incremental REP (**IREP**) modification integrates it with a separate and conquer technique [16]. Repeated Incremental Reduced Error Pruning [16] (**RIPPER**) is a state-of-the-art rule learning algorithm, a modification of the REP and rectifies its main weaknesses [29], and improvement of IREP with a heuristics for evaluation of rules during pruning and greedy stoppage of adding rules, as well as an additional post-pass phase for rule optimisation. The optimisation is done by re-learning already generated rules under a different context, as other rules have already been generated and the dataset has been changed [29].

Certifiably Optimal Rule Lists [5] (**CORELS**) is currently the best algorithm for solving optimal decision tree problems. Corels strategy is to generate rule candidates and select ones that maximally reduce the classification error based on splitting data or misclassified instances coverage to improve overall accuracy. Corels terminates on a predefined stopping criterion, like maximum number of rules, minimum accuracy improvement or after an exhaustive search of the rule space [5]. A key feature of CORELS is the ability to provide theoretical guarantees of optimality for the generated rule list, ensuring the rule list is certifiably optimal within a certain error bound.

2.3 Rule extraction from neural networks

Extraction of rules is based on connection weight analysis and network node sensitivity, translated into symbolic rules using quantisation, mirroring the underlying knowledge of the network [4]. Rule extraction from neural networks is not an unexplored idea and dates back to the 1990s.

KT [26] algorithm is one of the first methods for extracting symbolic rules from trained neural networks. The paper [26] formally established an association between neural networks and rule-based classifiers, along with verifying that neural network-generated rules outperform decision trees on datasets with noisy data. The KT algorithm, in essence, performs activation analysis of hidden layers to search for patterns of influential activations and features for the final decision. Pattern search is done by combining positive and negative attributes, exploring and refining positives with negated negative attributes, and vice versa for negative attributes, finding patterns for attribute-concept relationships. A pattern of features is ultimately expressed as a symbolic rule, which is generated by performing a systematic tree search led by heuristic pruning to reduce the exponential search space.

Knowledge-Based Artificial Neural Networks [78] (**KBANN**) is an approach which merges symbolic knowledge, often problem or domain-specific, with connectionist AI paradigm employing deep neural networks to further process the knowledge with backpropagation training, creating a hybrid system. The basic principle is that the neural network is initialised with preexisting knowledge already expressed as rules. The network is then trained, which is sped up, thanks to being led by the initialisation knowledge, and after training, the rules are extracted again back into symbolic form using some form of pattern analysis, like the KT algorithm 2.3.

The paper [14] introduced rule extraction for binary convolutional neural networks with stochastic local search. The method successfully creates a rule-based system, which can model a neural network while at the same time producing interpretable logical rules. It showed that the extracted rules are usually longer, but they explain the decision for the input and also have the benefit of easy interpretation by visualisations, which is extensively used in interpretations of experimental results of the thesis. The rules themselves are extracted with activation analysis, which is an approach explored and utilised by the solution presented in the thesis in subsequent chapters.

Chapter 3

NuLog method introduction

The chapter details training and utilising quantised neural networks, emphasising the neural networks with ternary weights and binary activations and exploring different techniques to train and achieve such a network. The chapter also briefly discusses the network elements used for binarisation, such as feature quantisers and quantised dense layers. The solution presented in the thesis exploits the neuron's discrete weight property to convert such neurons to logical expressions. This property is the target of the proposed solution, introduced in this chapter, and the implementation details are explained in the next chapter. The method is essentially a rule extraction technique based on abductive reasoning and explanations introduced later to produce classification rule lists.

3.1 Quantised Neural Networks

Quantised neural network (QNN) is a network which has activations restricted to a discrete set of values represented as integers [75]. **Binarised neural network** (BNN) is a special case of a QNN, which has activations restricted to two discrete values $+1$ and -1; the weights can be restricted the same as well, but depending on the implementation, weights could also be ternary, with a third value of 0 [72]. The goal of the use of ternary weighted networks is the pruning of densely connected layers for redundancy reduction [27] [53] [55].

The best results are, by far, achieved with classic deep neural networks. Still, a significant restriction for their mass deployment is their requirement for massive resources to operate correctly and cannot be deployed on edge devices. Different techniques in the field of Tiny ML (ML for low-power devices) are being developed, like topological and hardware optimisations [75]. Quantised networks use reduced precision and significantly faster integer operations over floating point ones to minimise memory footprint and inference time on devices with limited hardware resources [75]. The solution proposed in this thesis is interested in the different aspects QNNs have to offer. Specifically, neurons of BNNs can be easily transformed and expressed as logical expressions, ultimately extracting classification decision rules from such a network.

Discrete weights are incompatible with classic backpropagation and the gradient descent approach for training, as the gradient is non-existent everywhere and offers no information [18]. Networks with continuous weights are transformed into quantised weighted networks with either quantised aware training or post-training quantisation [31].

Post-training quantisation is done without any fine-tuning, meaning no re-training, benefits of no overhead and limited training data requirements, but worse performance compared to quantised training [11]. Zero-shot quantisation is an approach to quantising without any training data [31] [18]. Numerous approaches exist for post-quantisation, like deterministic (sign), stochastic (probabilistic decision using hard sigmoid), ACIQ (finding optimal clipping range), AdaRound, simulated, and mixed-precision (different parameters use different levels of quantisation).

Quantised aware training is an adaptation of training to produce discrete neural networks. **Straight-through estimator** (**STE**), which is a simple and often the best approach for quantising stochastic discrete neurons [11]. STE originated from the 1950s perceptron training algorithm, where the gradient is not computed via the standard chain rule. The gradient is estimated via modifications where the identity function's derivative is a proxy for the derivative of the binary function [81]. The choice of the proxy function is not limited to just the identity; different functions offer different features. Examples of used proxy functions are clipped ReLU, vanilla ReLu [81], piecewise polynomial [50], hard tanh and SwishSign [19]. The intent is to minimise the discrepancy between forward and backward passes, as they are done, in essence, with different activation functions. For that reason, a network trained with the STE technique suffers enormously from instability, the introduction of gradient bias, discretisation errors and vanishing gradient, compared to full-precision neural network [19]. **BinaryConnect** is a process to train a network with regularisation to force binary weights, essentially a binary version of the DropConnect algorithm with custom noise distributions [17]. A different approach to ternary weight networks, which trains weights by first parametrising them with $\tanh(\Theta)$ imposing the weights in a range $[-1, +1]$ and a weight discretisation regularisation forces the weights to be ternary [21]. The approach mentioned above is being investigated and developed in a student Bachelor's thesis [59], to be used instead of the STE approach. **XNOR-Net** is an adaptation of BNNs for convolutional neural networks, optimising the convolutional layer by implementing the XNOR operation and dot product as bit counting over binary vectors [66].

In all our experiments, the network is trained using gradient descent estimation done with STE. The STE used in the BNN is an identity function estimator with a fixed window width of 1. The activation function is a hard hyperbolic tangens, with the deterministic binarisation of output with sign function. For weights, a ternary function is used, which essentially works as a sign function, except for a neighbourhood around 0 for which the output is exactly 0. The neighbourhood is defined by a window of fixed width 0.05. Weights and biases are initialised with uniform distribution, batch normalisation is used, and no weight sparsification is performed during training. BNNs rely on binarised features, and different binarisation techniques exist [31]. The training process in our experiments uses a logit cross-entropy as an error function and an AdaBelief optimiser.

3.2 Logical Neural Networks

The method proposed in this thesis for extracting logical rules from neural networks necessitates a discrete weights network, specifically a network with ternary weights $\{-1, 0, 1\}$ and binary activations $\{-1, 1\}$, as described in the paper [18]. The choice of binarisation outputs as $\{-1, 1\}$ over $\{0, 1\}$ representation does not matter as both encodings are linear transformations of the other [41]. Most implementations opt for $\{-1, 1\}$ for the symmetry and simplicity of implementation of operations like AND, XOR and multiplication (implemented as simple sign-changing). The reason for specific weight and activation requirements is that the neuron of such a BNN can be expressed as a logic expression with equivalent performance [78]. A neural network where all neuron activation functions are expressed as logic expressions in the form of M-of-N rules is referred to as **Logical Neural Network** (LNN). LNN is the basis for extracting rules based on the training data for the BNN, which is described later in the chapter.

LNN architecture consists of two types of layers. An important distinction is that the first layer is special since it is a feature quantiser for features with real values and training the quantisation strategy. The feature quantiser is integral for extracting interpretable rules, as it is transformed into a predicate which defines which feature values are important. The quantisation level is the number of threshold values that perform the quantisation of real-valued features, which is predetermined by the network design. Features are quantised into quantisation level $+1$ number of bins. Each feature has the same level, with a specific number varying from problem to problem. For instance, for MNIST data, a single level proved sufficient as the feature values are in a relatively narrow interval $[0, 1]$, and a single predicate splitting the interval into two bins is more than sufficient 5.3. In contrast, for the classification problem on the flower dataset, level numbers can vary between 4 and 20 5.2. Theoretically, there can be infinitely many quantisations, according to Cantor's diagonal argument for the cardinality of the set $\mathbb{R}$ [22], but limited by the floating-point representation of real numbers in computer memory. If the data processed is binary, with correct encoding $\{-1, +1\}$, feature quantisation is unnecessary and should be omitted from the model.

For $x \in \mathbb{R}^n$, **feature quantiser neuron** used is formulated as

$$y = \text{sign}(\mathbf{W}x + \mathbf{b})$$

Here is a general expression for converting a binarised neuron for quantisation into logical expressions with predicates.

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} \end{bmatrix} \mathbf{b} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{bmatrix} \mathbf{P} = \begin{bmatrix} x_1 \leq -\frac{b_{11}}{w_{11}} \\ x_1 \leq -\frac{b_{12}}{w_{12}} \\ \vdots \\ x_1 \leq -\frac{b_{1k}}{w_{1k}} \\ x_2 \leq -\frac{b_{21}}{w_{21}} \\ x_2 \leq -\frac{b_{22}}{w_{22}} \\ \vdots \\ x_n \leq -\frac{b_{nk}}{w_{nk}} \end{bmatrix}$$

Example. Converting a feature quantiser for binary outputs into an equivalent predicate is shown below. It shows a two-element real-valued input sample x after a forward pass through a layer that binarises input into two quantities for each feature, resulting in a four-element output vector.

$$x = \begin{bmatrix} 0.75 \\ -1.23 \end{bmatrix} \mathbf{W} = \begin{bmatrix} 1.52 & -1.18 \\ 0.12 & -1.05 \end{bmatrix} \mathbf{b} = \begin{bmatrix} -0.03 & -1.21 \\ -0.23 & -1.03 \end{bmatrix}$$

$$\begin{bmatrix} 0.75 \leq 50.67 \\ 0.75 \leq -1.03 \\ -1.23 \leq 1.92 \\ -1.23 \leq 0.98 \end{bmatrix} \Rightarrow y = \begin{bmatrix} \mathbf{T} \\ \mathbf{F} \\ \mathbf{T} \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

Predicates are only compatible with real-valued features, as discrete features require exact equality. Still, they could be simulated if integer features are transformed into real values and with multiple inequalities in conjunction to uniquely determine the integer value in the range specified by the predicates. For example $x \in \{1, 2, 3\}$ and a predicate $x = 2$ can be expressed as $(x > 1.5) \wedge (x < 2.5)$.

Quantised dense layer is a linear-BatchNorm-binarise module mapping binary input to output vectors with trainable weight parameters [41]. Let bin_w be a function defined as $\text{bin}_w : \mathbb{R}^{m \times n} \to \{-1, 0, 1\}^{m \times n}$. For $x \in \{1, -1\}^n$, it is formulated as

$$y = \text{sign}(\text{BatchNorm}(\text{bin}_w(\mathbf{W})x) + b)$$

Here is a general way of how a binarised dense layer produces outputs and the strategy for converting a densely connected binarised neuron into logical expressions with M-of-N rules, where lit is a function defined as returning a logical literal, which references the antecedent layer's neuron. In case the $\text{bin}_w(w)$ is -1, the literal is negated, while if it is 0, the literal is discarded from the M-of-N terms. The algorithm for converting a BNN neuron's weights and biases into M-of-N rules is shown here 1, with the formula for determining the correct M value.

$$\mathbf{W} : \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} \quad \mathbf{b} : \begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} \begin{bmatrix} m_1\text{of}(\text{lit}(w_{11}, 1), \cdots, \text{lit}(w_{1k}, k)) \\ m_2\text{of}(\text{lit}(w_{21}, 1), \cdots, \text{lit}(w_{2k}, k)) \\ \vdots \\ m_n\text{of}(\text{lit}(w_{m1}, 1), \cdots, \text{lit}(w_{mk}, k)) \end{bmatrix}$$

Example. Conversion of weight and bias matrices of a quantised dense layer into M-of-N rules of a logical layer, with equivalent performance, are shown. It shows the forward propagation result of the previous example's output through the network.

$$x = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} 0.68 & 0.37 & -0.23 & 0.07 \\ -0.72 & 0.26 & 0 & 0.92 \\ -0.16 & -0.39 & -0.33 & 0.74 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.5 \\ 0.5 \\ -1.5 \end{bmatrix}$$

$$\text{M-of-N rules:} \begin{bmatrix} 2 \text{ of } (x_1, x_2, \bar{x}_3, x_4) \\ 2 \text{ of } (\bar{x}_1, x_2, x_4) \\ 3 \text{ of } (\bar{x}_1, \bar{x}_2, \bar{x}_3, x_4) \end{bmatrix} \Rightarrow y = \begin{bmatrix} \mathbf{T} \\ \mathbf{F} \\ \mathbf{F} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Algorithm 1 Conversion of ternary weights into M-of-N rules

```
Require: $w$-ternary weights, $b$-bias, $j^{\text{th}}$ layer of the NN
  function TERNARY WEIGHTS TO MOFN RULE($w^j, b^j$)
    if any $w^j \notin \{-1, 0, 1\}$ then
      Error: conversion to logic requires weights to be $\{-1, 0, +1\}$
    end if
    $\mathcal{T} \leftarrow \emptyset$
    for each $w_i^j \in w$ do
      if $\text{bin}_w(w_i^j) > 0$ then
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{f_i^{j-1}\}$
      else if $\text{bin}_w(w_i^j) < 0$ then
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\bar{f}_i^{j-1}\}$
      end if
    end for
    $M \leftarrow \frac{|\mathcal{T}|-b}{2}$
    $\bar{M} \leftarrow \left\lceil \frac{|\mathcal{T}|-b}{2} \right\rceil$
    if $M = \bar{M}$ then
      $M \leftarrow \bar{M} + 1$
    else
      $M \leftarrow \bar{M}$
    end if
    return MofN($M, \mathcal{T}$)
  end function
```

3.3 Rule extraction

Let $\mathcal{X}$ and $\mathcal{Y}$ denote the sets of samples and labels, respectively and the LNN denoted as classification function $f = f^n \circ f^{n-1} \circ \cdots \circ f^1$, where f^i is the i^{th} layer with each neuron expressed as an M-of-N rule. Function f takes as input a sample $x \in \mathcal{X}$ and outputs a prediction π as a label $c \in \mathcal{Y}$.

Decision rule extracted from the structure of the LNN provides an abductive explanation of the made decision 2.1 [38]. It is defined as a function taking a sample $x \in \mathcal{X}$ as input and output a binary value, representing whether the network decided it belongs to class $c \in \mathcal{Y}$ for which $f(x) = c$ or not, formulated here

$$r^c : \mathcal{X} \rightarrow \{\mathbf{F}, \mathbf{T}\}$$

$$\exists c \in \mathcal{Y}, \exists x \in \mathcal{X} \quad \text{s.t.} \quad f(x) = c$$

$$\forall x' \in \mathcal{X}, r^c(x) = \mathbf{T} \Rightarrow f(x') = f(x) \quad \wedge \quad \forall z \in \mathcal{X}, f(z) \neq f(x) \Rightarrow r^c(z) = \mathbf{F}$$

The above implication that defines the rule must hold for the entire problem space, meaning for both training and testing sets of samples. Let $\mathcal{X}_{\text{tr}}$ and $\mathcal{X}_{\text{te}}$ denote the sets of training and testing samples, $\mathcal{X}_{\text{tr}} = \mathcal{X} \setminus \mathcal{X}_{\text{te}}$. A less strict criterion is defined and used such that the above implication holds only for the training set $\mathcal{X}_{\text{tr}}$ which was used to train the BNN used in the extraction, while at the same time, no guarantees are given for the set of samples defined as the testing set $\mathcal{X}_{\text{te}}$. The decision rule is now formulated as

$$\exists c \in \mathcal{Y}, \exists x \in \mathcal{X}_{\text{tr}} \quad \text{s.t.} \quad f(x) = c$$

$$\forall x' \in \mathcal{X}_{\text{tr}}, r^c(x) = \mathbf{T} \Rightarrow f(x') = f(x) \quad \wedge \quad \forall z \in \mathcal{X}_{\text{tr}}, f(z) \neq f(x) \Rightarrow r^c(z) = \mathbf{F}$$

This criterion, which defines rules based on the training set alone, gives no guarantees for the performance of said rule on the testing set regarding precision and recall.

Let $\mathcal{X}^c_{\text{tr}} \subseteq \mathcal{X}_{\text{tr}}$ be a set of samples from $\mathcal{X}_{\text{tr}}$, strictly and unambiguously classified as a class c by the LNN, referred to as set of class data. Set $\mathcal{X}^{\bar{c}}_{\text{tr}}$ is the complement of the of class data set $\mathcal{X}^{\bar{c}}_{\text{tr}} = \mathcal{X} \setminus \mathcal{X}^c_{\text{tr}}$, referred to as non-class data later. $\mathcal{R}_c$ is a set of decision rules, defined above 3.3, then rs^c is a function that can be defined as

$$\mathcal{R}_c = \{r^c \mid \exists x \in \mathcal{X}^c_{\text{tr}}, r^c(x) = \mathbf{T} \wedge \forall z \in \mathcal{X}^{\bar{c}}_{\text{tr}}, r^c(z) = \mathbf{F}\}$$

$$\text{rs}^c : \mathcal{X} \to \{\mathbf{F}, \mathbf{T}\}$$

$$\text{rs}^c(x) = \begin{cases} \mathbf{T} & \exists r^c \in \mathcal{R}_c \text{ s.t. } \mathrm{r}^c(x) = \mathbf{T} \\ \mathbf{F} & \forall r^c \in \mathcal{R}_c \text{ s.t. } \mathrm{r}^c(x) = \mathbf{F} \end{cases}$$

Using this rule set extraction function is defined as

$$\text{EX}_{rs} : (\mathcal{X}_{\text{tr}}, f) \to \mathcal{R}_c$$

$$\forall x \in \mathcal{X}_{\text{tr}} \text{ s.t. } \text{rs}^c \in \mathcal{R}_c, \text{rs}^c(x) = \mathbf{T} \Rightarrow f(x) = c$$

$$\forall x \in \mathcal{X}_{\text{tr}} \text{ s.t. } \text{rs}^c \in \mathcal{R}_c, \text{rs}^c(x) = \mathbf{F} \Rightarrow f(x) \neq c$$

The rule set extraction function takes as input the training set and the function f, representing the LNN, and produces $\mathcal{R}_c$ as a set of rules explaining the classification decisions of the network.

Extracted rules will provide understandable abductive explanations for classifications, be used to form a rule-based classifier, evaluate and compare its performance to the original network it was extracted from, as well as some contemporary rule learning methods. The problem of abductive explanation extraction is proven to be NP-hard; more specifically, to highlight the level of computational complexity, it is even NP^{NP} hard [38].

The complexity of the described problem justifies the use of heuristic approaches to find a solution, which is suboptimal but sufficient. For that reason, the method introduced in the thesis, named **NuLog** short for Neural Logic, implements the rule set extraction function EX_{rs} as defined above led by heuristics of the activation values for each neuron based on the $\mathcal{X}_{\mathrm{tr}}$ training set.

The NuLog logical rules extraction method, based upon abductive reasoning and explanation as defined here 2.1, extracts the explanation rule for the classification for a single sample x and its prediction π, which is then tested for how well it explains other samples.

3.4 Redundancy assumption and heuristic for network pruning

Artificial neural networks often exhibit redundancy, which can be utilised and reduced in multiple diverse approaches. The redundant networks have better predictive performances and exhibit more robustness but suffer from a high memory footprint and slower inference times [62]. Techniques for removing redundant elements of a network are precision reduction, weight sparsification and pruning dense and convolutional layers [62]. BNNs are used to reduce redundancy by discretising weights and outputs without significant performance loss, forcing the weights to belong to the set $\{1, 0, -1\}$ [18].

One example of redundancy in the BNNs designed with the feature quantiser layer, specifically the threshold values trained to binarise real-valued features, is that they fall out of the value ranges specified by the nature of the problem or the training data. A detailed explanation of redundancy reduction by identifying contestants and propagating them through the network is explained in the appendix B. The mentioned constants propagation is a preprocessing of the LNN after conversion from a BNN and does not affect the accuracy performance, it only simplifies the network structure to improve inference performance.

Papers [62] [12] discuss different pruning techniques and approaches for complete precision networks. Generally, the problem of removing redundancy via network pruning is expressed as training the weights with sparsification terms in the cost function to minimise the model size, represented by an l_0 norm minimisation [55]. Unfortunately, the l_0 norm minimisation is a known NP-complete problem [61], as it is non-convex, requiring combinatorial search [55].

The network pruning problem can be formulated as a binary or 0-1 integer program, finding the optimal selection of neurons to keep [82]. The goal is to keep the most important connections to have the simplest possible network with minimal loss in precision and recall properties. The pruning can be done individually for each neuron or as a whole for the entire layer. As the M-of-N rule, used in LNNs, equates to having a $\binom{N}{M}$ of rules to choose from, selecting the best ones by exhaustive search has a factorial time complexity for a single neuron. Furthermore, selection can be formulated as a 0-1 integer program, where the goal is to find at least M terms which, in conjunction, form a rule which maximises the coverage function over a set of samples $\mathcal{X}$.

$$\begin{aligned}
\mathbf{max} \sum_{j=1}^{|\mathcal{X}|} y_j & \\
\mathbf{s.t.} \sum_{i=1}^{n} x_i \geq M & \\
y_j \leq 1 - \frac{1}{N}\sum_{i=1}^{N}(x_i - s_{ij}) & \qquad \forall j \in \{1, \cdots, |\mathcal{X}|\} \\
x_i \in \{0,1\} & \qquad \forall i \in \{1, \cdots, N\} \\
s_{ij} \in \{0,1\} & \qquad \forall i \in \{1, \cdots, N\}, \forall j \in \{1, \cdots, |\mathcal{X}|\} \\
y_j \in \{0,1\} & \qquad \forall j \in \{1, \cdots, |\mathcal{X}|\}
\end{aligned}$$

Where:

- $\mathcal{X}$ set of samples
- N number of neurons in the M-of-N rule
- M number of neurons to be selected from the M-of-N rule
- $\mathbf{S}$ matrix of $N \times |\mathcal{X}|$ binary constants modelling whether the i^{th} term evaluates true on the j^{th} sample of the sample set, if the term is not selected the value must be 0
- x vector of N binary decision variables modelling whether an antecedent layer's neuron is included in the final selection, or in other words, the decision is the link between the layers pruned
- y vector of $|\mathcal{X}|$ binary variables modelling whether the conjunction of selected terms evaluates true on the j^{th} sample of the sample set

0-1 integer programming is a known NP-complete problem [42] and is listed as one of Karp's 21 NP-complete problems [43].

Papers [82], [55] introduce various approaches to estimating the importance of neurons for pruning and redundancy elimination. Their idea works with convolutional neural networks, estimating the importance of final layer neurons, then propagating it backwards through the network by weights and removing connections if the estimation is below a predetermined threshold value.

NuLog's rule extraction idea is to have a suboptimal greedy heuristic-guided search and selection of the best M subset from N terms, essentially performing pruning. The heuristic is based on the activation statistics collected independently for each neuron or logical expression. It is defined as the absolute difference in the percentage of positive activations between class and non-class data that identifies neuron importance for best class discrimination. The heuristic estimates neuron importance in each layer independently of preceding and succeeding layers. The methodology is founded upon the premise that the redundant neurons, if such are present, will be indicated by the importance metric and eliminated from the selection, which would result in extracting highly general rules with relatively high coverage. The NuLog extraction method does not presuppose the existence of redundancy in hidden layers of the network for the rule to be extracted but bases the number of rules which can be extracted and their generalisation on the redundancy. The NuLog's approach differs from the papers referenced above, as it independently estimates neuron importance for each layer and does not propagate it from back to front. Also, the NuLog is designed around LNNs and BNNs instead of convolutional networks with continuous weights.

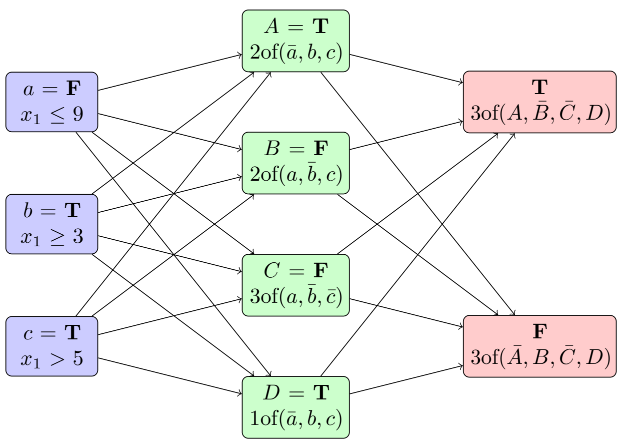
Chapter 4

NuLog method implementation

This chapter introduces the NuLog method for extracting rules as prediction explanations. The chapter is divided into sections by describing specific phases of the NuLog method. The first section presents NuLog's main method, network pruning, which deals with extracting an abductive explanation. The second part extensively details the implementation of extracting sufficient rules with a minimal number of redundant terms. Explicit pseudo-codes describing the algorithms solving the tasks at hand are provided along with the time complexity analysis. Adequate token examples accompany the introduction for better understanding.

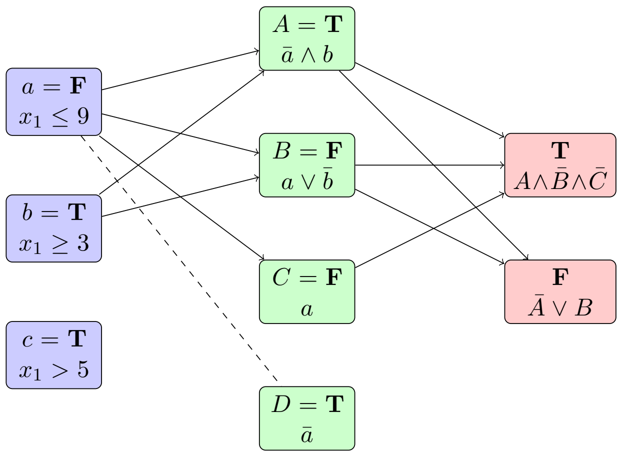
4.1 Method overview and implementation

This section is the general outline of the idea behind using LNNs and obtaining a reasonable explanation for the classifications made by the same network. The goal of the procedure is to determine the most significant terms related to features responsible for the procured classification. An ideal result would be to have a conjunction of terms related to significant value ranges. The procedure is done in multiple steps and works by greedily selecting a sample, extracting a rule from its classification and checking how well it explains other samples with identical classification. A pedagogical example is given here with a simplified overview of the procedure works and will be used as a reference point in the overview. The figure below 4.1 is a simple representation of an LNN required by this method, which consists of three layers, the input layer being a feature quantiser and the rest being quantised dense layers with neurons as logical expressions in the form of M-of-N rules. The presented network deals with a single feature vector and classifies it into class $c \in \{1, 2\}$. For this trivial example, the term importance estimated by activation statistics will be identical to the lexicographic order of the neuron labels. The first layer's expressions have been marked by lower case letters and the second by upper case letters for easier distinction between expressions. The sample to gain an explanation for is a vector $x = \begin{bmatrix} 10 \end{bmatrix}$.



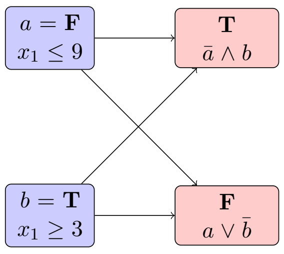
$a = \mathbf{F}$ $x_1 \leq 9$
$b = \mathbf{T}$ $x_1 \geq 3$
$c = \mathbf{T}$ $x_1 > 5$
$A = \mathbf{T}$ $2\text{of}(\bar{a}, b, c)$
$B = \mathbf{F}$ $2\text{of}(a, \bar{b}, c)$
$C = \mathbf{F}$ $3\text{of}(a, \bar{b}, \bar{c})$
$D = \mathbf{T}$ $1\text{of}(\bar{a}, b, c)$
$\mathbf{T}$ $3\text{of}(A, \bar{B}, \bar{C}, D)$
$\mathbf{F}$ $3\text{of}(\bar{A}, B, \bar{C}, D)$

(a) : Initial LNN state



$a = \mathbf{F}$ $x_1 \leq 9$
$b = \mathbf{T}$ $x_1 \geq 3$
$c = \mathbf{T}$ $x_1 > 5$
$A = \mathbf{T}$ $\bar{a} \wedge b$
$B = \mathbf{F}$ $a \vee \bar{b}$
$C = \mathbf{F}$ a
$D = \mathbf{T}$ $\bar{a}$
$\mathbf{T}$ $A \wedge \bar{B} \wedge \bar{C}$
$\mathbf{F}$ $\bar{A} \vee B$

(b) : State of the LNN after pruning



$a = \mathbf{F}$ $x_1 \leq 9$
$b = \mathbf{T}$ $x_1 \geq 3$
$\mathbf{T}$ $\bar{a} \wedge b$
$\mathbf{F}$ $a \vee \bar{b}$

(c) : Rule extraction

Figure 4.1: NuLog - heuristic generalisation

4.2 Statistics collection

A prerequisite of the method is the collection of activation statistics of each neuron, which means tracking whether it is true or false. The intention is to take the training data, split it based on a class determined by the network for which an explanation is desired, and, for each sample, infer the network while collecting statistics on each layer. Statistics collection is done by first evaluating all samples labelled as a certain class, followed by evaluating all other samples not belonging to said class. The process is performed by layer encapsulation, which acts like regular layers but with the additional task of tracking which neuron logical expression is evaluated true in each pass. Activation statistics are later used as a metric to prioritise neurons and estimate the importance of their respective rules. Statistics collection is done on the entire training data but can be done on a subset as well.

4.3 Extracting abductive rule by network pruning

After all the activation statistics are collected, the process of extracting an abductive explanation 2.1 for a single sample begins. The image 4.1a visualises the evaluation of each neuron after a forward pass on the sample in question. The network is pruned such that all connections not contributing to the classification are removed in densely connected layers. The significance of the collected statistics is that they serve as a guide for deciding which element of the input vector is worth keeping for each neuron based on the activations of the previous layer. The pruning procedure starts from the output layer and proceeds through hidden layers, converting M-of-N rules into simple conjunctions or disjunctions until reaching the input layer. The input layer has nothing to prune, being only the quantiser layer and having simple predicates as expressions. For a single layer, each neuron is examined and converted independently, meaning the pruning process can be done in parallel.

Each neuron in LNN is represented as an M-of-N rule, and pruning transforms it into a conjunction of the best M selected terms from the N given terms. Conversion filters the terms not matching the previous layer's output, sorting the remaining terms by importance estimation, and finally selecting the best M terms from the matching terms and forming a rule. The first neuron of the final layer, in 4.1a, will be used as an example to demonstrate and clarify the pruning procedure. The neuron evaluates true, meaning its expression is taken as is, which states that three of the four terms must match.

Going through the terms individually and looking at their evaluations from the antecedent layer, it is seen that A is a match, as it expects the A neuron to be true, and it is evaluated to be true. $\bar{B}$ is a match since it negates the B's evaluation, which is false, meaning it is true and a match; C and $\bar{D}$ are also matching by analogy. As the number of matching terms here is more than the minimum of three, sorting of terms is required. Terms are greedily selected by importance order, meaning A, $\bar{B}$, $\bar{C}$ are chosen and put in conjunction. The rule defined here 2.2 as a conjunction is interpreted as a list of neurons from the previous layer that must be evaluated as true to have this neuron's output be true.

The process is identical whether the M of N rule's output should be true or false; the only difference is that if the output is false, the M-of-N rule is first negated, then pruned and finally, the conjunction is negated again, resulting in the disjunction of terms. Negation of M-of-N terms is defined by De Morgan's law defined later in the chapter, here 8. The second neuron, in 4.1a, to be pruned in the final layer, evaluates false, and its expression must be negated, so 2 of $(A, \bar{B}, C, \bar{D})$ is the rule to be pruned. Repeating the same procedure, only A and $\bar{B}$ match the antecedent outputs and are put in conjunction, but the new expression must be negated so the final result is a disjunction $\bar{A} \vee B$. Disjunction is interpreted as a list of neurons from the previous layer, which must be false to have this neuron evaluated as false.

The pruning procedure results in sparse connections between layers, shown in the image here 4.1b, and the neurons not referenced by their subsequent layer should be completely removed for optimisation as they are redundant. The pruned network must be evaluated correctly for all samples the extracted rule covers, but the size of this set is not guaranteed, only guarantee is it must be non-empty if pruning is done correctly. Generally, for all samples other than the one being explained, correct predictions of a pruned network are not guaranteed. In case the set of covered samples by a rule is empty, it means the pruning was not done correctly, which happens if, for some M-of-N rule, less than M terms remain for selection, resulting in a contradictory rule with zero coverage being extracted.

H function. The most important function is the H function, which controls the pruning phase and decides ahead of time for the entire layer which terms are to be preserved and which are to be discarded. Assuming a neural network in the form of $f^n \circ f^{n-1} \circ \cdots \circ f^1$, where $f^i \in \mathcal{F}$ is the logical layer with neurons expressed as M-of-N rules and collected statistics from the initial phase. For a given input x, the output of layers (activation vector) is denoted as $h^i = f^i(h^{i-1})$ with $h^0 = x$. The H function can be defined as $\mathrm{H} : \mathcal{F} \to \{(d, s)\}^{m^i}$ where $d \in \{+, -\}$ and $s \in \mathbb{R}$ and serves to decide for each neuron in the layer what literal is matching and calculates the importance score estimation. The results are once used for pruning the current layer, then saved and reused for the following layer, as pruning the subsequent layer requires decisions made for the current layer. For a single neuron, the function is expressed as

$$\mathrm{H}_j(f_j^i, (a_c, n_c), (a_{\bar{c}}, n_{\bar{c}})) \begin{cases} (+, |\frac{a_c}{n_c} - \frac{a_{\bar{c}}}{n_{\bar{c}}}|) & f_j^i(h_j^{i-1}) = 1 \\ (-, |\frac{a_c}{n_c} - \frac{a_{\bar{c}}}{n_{\bar{c}}}|) & f_j^i(h_j^{i-1}) = -1 \end{cases}$$

Where a_c is the total number of positive or true activations of the j^{th} neuron, n_c is the total number of inferences of the neuron on class data, with $\bar{c}$ denoting non-class data, which is stored in the layer.

Prune Literals. The procedure to prune literals, as input, takes a literal, which refers to the output of a specific neuron of the preceding layer and a list containing tuples with all data needed for pruning. The literal can be an affirmation expressed as x_j^{i-1} or negation $\bar{x}_j^{i-1}$. The procedure, as shown in the expression below, checks the corresponding decision made by the H function relative to the supplied literal being pruned.

$$\mathrm{Prune}(t_j, h_j^{i-1}) = \begin{cases} t_j & t_j \equiv x_j \wedge \mathrm{decision}(h_j^{i-1}) = + \\ t_j & t_j \equiv \bar{x}_j \wedge \mathrm{decision}(h_j^{i-1}) = - \\ \mathbf{F} & \text{othervise} \end{cases}$$

If the evaluation decision of a literal matches the decision value, the same literal is returned as a result. The matching cases are if the literal is affirmative and the output of the neuron in the previous layer is $+1$ or if the literal is the negation of its respective value and the output of the neuron is -1, for all other cases mismatch is the default result. If there is a mismatch, a constant value false is returned. Indicating the term is to be pruned or discarded, because the literal does not contribute to the activation of the neuron.

Prune M-of-N rule. This procedure deals with the pruning of M-of-N rules, and as input, it takes the M-of-N rule mentioned above and a list of tuples necessary for the pruning itself. The algorithm is shown here 0.

It filters the M-of-N rule terms by the result of pruning functions applied to those terms. For each term, the according pruning function is called, and in the case the result of pruning a particular term is a constant, the term is discarded. Three situations are possible after filtering the terms of the M-of-N rules. In the case that too many terms had to be pruned. Now, the number of remaining terms is less than M, which means the condition for the M-of-N rule to evaluate true can never possibly be met and is always going to evaluate false, so the procedure returns a constant false value. In the other case, the number of filtered terms equals the number M, and the condition to evaluate to true is met if all terms are evaluated to true. Therefore, the result of the procedure is a conjunction of all remaining terms. The final case is if more than M terms remain, the best M terms must be selected. A greedy approach is used; the list of terms is sorted by the scores saved in the list of tuples calculated ahead of time by the H function; first, M terms, after sorting, are selected and placed in conjunction with the final result of the procedure. The time complexity is $\mathcal{O}(n \log n)$, where n is the number of terms in M-of-N, and p represents the pruning complexity for the terms.

Algorithm 2 Prune M of N rule

```
Require: $r_{M,N}$- M-of-N rule, $\mathcal{H}$-previous layers's decision list
 1: function PRUNE($r_{M,N}, \mathcal{H}^{i-1}$)
 2:     $\mathcal{T} \leftarrow \emptyset$
 3:     for each $t_j \in r_{M,N}$ do
 4:         $t'_j \leftarrow \text{Prune}(t_j, h_j^{i-1})$
 5:         if $t' \neq \mathbf{F}$ then
 6:             $\mathcal{T} \leftarrow \mathcal{T} \cup \{t'_j\}$
 7:         end if
 8:     end for
 9:     if $|\mathcal{T}| < M$ then
10:         return F
11:     else if $|\mathcal{T}| = M$ then
12:         return Rule($\mathcal{T}$)
13:     end if
14:     $\mathcal{T}' \leftarrow \text{Sort}(\mathcal{T}, \text{by} = \text{score}(\mathcal{H}), \text{order} = \text{desc})$
15:     return Rule($\{t'_1, t'_2, \cdots, t'_m\}$)
16: end function
```

Special cases. For special cases, if the logical expression is in the form of a rule or a rule set, which represents conjunction and disjunction, respectively, the pruning process is delegated to the pruning M-of-N rule with special properties. For the case of the rule, it is equivalent to pruning an N-of-N rule, where N is the number of terms in the rule.

$$x_1 \wedge x_2 \wedge \cdots \wedge x_n \equiv n \text{ of } (x_1, x_2, \cdots, x_n)$$

On the other hand, the case of the rule set is equivalent to pruning a 1-of-N rule.

$$x_1 \vee x_2 \vee \cdots \vee x_n \equiv 1 \text{ of } (x_1, x_2, \cdots, x_n)$$

Prune layer. The prune layer procedure takes as input an LNN layer which needs to be pruned and two lists which store important information needed for pruning. The lists hold tuples with evaluation decisions and scoring for their neurons.

The procedure is mapping each neuron's logical expression into its pruned version, importantly, taking the values of the second list, which is relevant to the current layer. For each neuron, its respective tuple is read for the decision value. If the decision is $+$, the neuron is supposed to evaluate as true, and it is pruned as is; on the other hand, if the decision is $-$, the neuron's logical expression must be negated and pruned as its negated form. After the pruning of the negated expression, the result must again be negated to cancel out the first negation. The procedure results in a logical layer with all of its neurons pruned, and the algorithm is shown here 0. Additionally, neurons which are not referenced by the subsequent layer should be replaced by a constant value, either true or false, or eliminated completely to reduce the structure complexity, without affecting the accuracy of the pruned network.

The time complexity is $\mathcal{O}(mn \log n)$, where m is the number of neurons in the current layer being pruned, while m is the number of neurons of the antecedent layer.

Algorithm 3 Prune Logical Layer

```
Require: $f^i$-Logical layer,$\mathcal{H}^i$-decision lists, $j$th neuron in $i$th layer in LNN
 1: function PRUNE($f^i, \mathcal{H}^{i-1}, \mathcal{H}^i$)
 2:     $\mathcal{E} \leftarrow \emptyset$
 3:     for $j \in |f^i|$ do
 4:         if decision$(h^i_j) = +$ then
 5:             $e \leftarrow$ Prune$(f^i_j, \mathcal{H}^{i-1})$
 6:         else
 7:             $e \leftarrow \neg$Prune$(\neg f^i_j, \mathcal{H}^{i-1})$
 8:         end if
 9:         $\mathcal{E} = \mathcal{E} \cup \{p\}$
10:     end for
11:     return LogicalLayer$(\mathcal{E})$
12: end function
```

4.3.1 Extracting sufficient rule

The rule extracted from the previous phase is guaranteed to cover the sample used to generate it, but no guarantees have been made about it covering other samples from the same class. The only guarantee is that it will not be evaluated as true for any sample of other classes; in other words, it will not be misclassified. It is important to state that the sample's class means the class determined by the LNN and not the actual class defined by the dataset generator, which means if the network incorrectly decides, so will the extracted rule. The generality of rules extracted by the previous phase can not be guaranteed, which greatly depends on the nature of the dataset being explained and the complex inference process of the neural network from which it is extracted. Terms selection is a post-processing procedure after NuLog's extraction to provide sufficient rules. Sufficient rules have improved coverage of extracted rules, ultimately improving the recall metric of a rule-based classifier without sacrificing precision.

Essentially, term selection is a performing feature selection by eliminating terms that are not essential for a correct decision on classifications of other samples of the same class and any sample of other classes. It can be formulated as a 0-1 integer program in the same fashion as the pruning of the M-of-N rules, defined here 3.4, but it is an NP-complete problem. Many different strategies exist for feature selection; all-subset selection is an exhaustive search that guarantees an optimal solution but works in $\mathcal{O}(2^{|\mathcal{T}|}m)$, where $\mathcal{T}$ is the set of terms of the extracted rule, and m relates to checking for misclassification and stands for total number of non-class samples.

A token example is presented here of the given problem: let $x, y \in \mathcal{X}^c$ be class data samples and $z \in \mathcal{X} \setminus \mathcal{X}^c$ a sample of non-class data decided by an LNN. r^c is a rule extracted from the LNN, as mentioned earlier by NuLog, and it covers only the sample x.

$$x = \begin{bmatrix} 0.25 \\ 13.1 \\ 2 \end{bmatrix} \quad y = \begin{bmatrix} 0.33 \\ 10.9 \\ 3.1 \end{bmatrix} \quad z = \begin{bmatrix} -0.1 \\ 14.2 \\ 5.5 \end{bmatrix}$$

$$r^c(x) = (x_1 \geq 0)_1 \wedge (x_2 > 11)_2 \wedge (x_2 \leq 15)_3 \wedge (x_3 \geq 1)_4$$

$$r^c(x) = 1 \quad r^c(y) = 0 \quad r^c(z) = 0$$

NuLog's method utilises the collected activation statistics from the initial phase explained here 4.2 as a heuristic to guide the selection. This metric defines the best terms as those that evaluate the same value for most class samples and, for most non-class samples, evaluate the opposite, while the worst terms are those that equally evaluate the class and non-class data. In a nutshell, the difference between activations as an indicator of discrimination power between classes of a single predicate is taken. The set of sorted terms for the token example is $\mathcal{T} = \{t_3^{(0.75)}, t_1^{(0.6)}, t_4^{(0.56)}, t_2^{(0)}\}$, along with importance scores.

The selection process to reduce the rule length is done by sorting all terms by the importance score. After sorting, terms are greedily added to a rule until it stops misclassifying; at this point, the reduced rule is found, and the final rule is generated. For the token example, the first term to be added is the fourth in the original rule, and the modified rule currently is

$$r^c(x) = (x_2 \leq 15)_4 \quad \rightarrow \quad r^c(x) = 1 \quad r^c(y) = 1 \quad r^c(z) = 1$$

As it is misclassifying, the second best term is added, which happens to be the first term in the original rule.

$$r^c(x) = (x_2 \leq 15)_4 \wedge (x_1 \geq 0)_1 \quad \rightarrow \quad r^c(x) = 1 \quad r^c(y) = 1 \quad r^c(z) = 0$$

The new rule does not misclassify sample z and any other sample in non-class data, so the search is finished. The final rule is shown above, which covers both x and y samples and the generalisation is improved without sacrificing precision. The final explanation provided is that x and y are classified as a certain class because $x_1 \in [0, +\infty)$ and $x_2 \in (-\infty, 15]$, while $x_3 \in (-\infty, +\infty)$ and does not influence the decision.

The order in which terms are selected to be removed from the full conjunction (backwards selection) matters as the issue is that by removing one term, the coverage of a rule is changed, which might not be possible to further expand by additional removals, starting to misclassify, but with a different removal order more terms might be removed to gain a similar coverage. Still, a shorter rule is achieved, which is also a goal, as shorter rules are easier for humans to understand and interpret 5.1. The same reasoning holds for the inclusion of terms in forward selection.

The heuristic-guided search is an adaptation of the Top-Down Hill-Climbing algorithm first mentioned here 2.2. It is a suboptimal solution regarding rule length because, just like the hill-climb algorithm, it easily gets stuck in local minimums. However, the greedy selection is optimised for time complexity and runs in $\mathcal{O}(n \log n + nm)$, where n is the number of terms in a rule and m is the total number of non-class samples required to be consulted to avoid misclassification. Rule coverage of class samples should be improved after selecting the best terms, although this is still not a guarantee, as it again greatly depends on the nature of the training set. The pseudo-code for the procedure is shown below 0.

Algorithm 4 Best terms selection

```
Require: $r^c$- rule, $\mathcal{X}$-training data, $\mathcal{S}$-predicate scores, $c$-class
 1: function TERMS SELECTION($r^c, \mathcal{X}, \mathcal{S}, c$)
 2:     $\mathcal{T}_s \leftarrow \mathrm{Sort}(\mathrm{r}^c, \mathrm{by} = \mathcal{S}, \mathrm{order} = \mathrm{desc})$
 3:     $\mathcal{T} \leftarrow \emptyset$
 4:     for each $t \in \mathcal{T}_s$ do
 5:         $\mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
 6:         if $\neg\mathrm{Misclassify}(\mathcal{T}, \mathcal{X}, \mathrm{sf})$ then
 7:             return $\mathrm{Rule}(\mathcal{T})$
 8:         end if
 9:     end for
10:     return $\mathrm{r}^c$
11: end function
```

The only exception is that the order in which terms are to be selected in advance is determined by the heuristic score of each term. Inclusion and elimination implemented over a heuristic-determined order will finish with the same selection of terms, as both of them will run until the conjunction of selected terms stops misclassifying in such a predetermined order. The reason behind this is that the order is known ahead and just reversed. The process is implemented in the opposite direction by sorting in ascending order and removing terms from the full rule until a misclassifying rule is obtained. The order of sorting the terms should depend on whether the sufficient rules are expected to be longer than half of the length of the extracted rule; if that is the case, it makes more sense to remove terms from the full conjunction. On the other hand, if sufficient rules are expected to be shorter than half the length of the extracted, then it makes more sense to build the conjunction by including terms. Different directions of term selection, which use a heuristic guide like in NuLog, provide the same final rule. It must be noted that in the general case, if the ordering for forward selection is not the reverse of the backward selection, then the two approaches do not guarantee the same selection, for instance, with randomised ordering.

4.4 Extracting classification rule set

Rule extracting is accomplished by converting the pruned network to a single-layer network, which must provide identical predictions as the original input network, $f^n \circ f^{n-1} \circ \cdots \circ f^1 \equiv f^{n,n-1,\cdots,1}$, and the rule is read from the c^{th} neuron in the single-layer network, $r^c \equiv f_c^{n,n-1,\cdots,1}$. The extraction procedure lacks novelty, and it is skipped over in this chapter, but it is based on merging and simplifying logic expressions and is exhaustively detailed in the appendix; refer to the appendix here A for explicit clarification.

The procedure extracts rules and explains the entire class data as determined by the trained neural network. It takes as input an LNN, training data, and the class for which an explanation is wanted. The result returned is a set of rules explaining the class of interest. In short, it performs NuLog pruning extraction and post-extraction processing iteratively to generate all possible rules and provide explanations of classifications for the provided data.

Extraction of rules starts by taking the LNN, wrapping it in a statistics collection layer, and executing a forward pass on all training data to collect the activation statistics. The statistics are collected ahead of any rule extractions.

The first sample of unexplained class data is taken from the set and evaluated using the wrapped network. If the class output does not predict the class of interest, the sample is removed from the list and discarded, and the loop starts from the beginning; this is done as it makes no sense to explain why something is classified as a certain class if it is not classified as such.

If the output matches the class of interest, the wrapped network is pruned. After pruning, a rule is extracted from the network. The initial rule is then subjected to terms selection to obtain a sufficient rule without redundant terms.

The extracted sufficient rule filters class data to remove all samples that the rule covers. In case the rule fails to explain anything, in other words, it evaluates false on all samples, including the sample used to generate it, an exception is raised, and generation is terminated. If that case occurs, it indicates that the pruning phase was not done properly and resulted in a contradiction inside the pruned network. It means a contradictory rule or a constant false value was extracted from the network, which fails to explain anything. The issue stems from the criteria defined in the H function, which controls pruning. On the other hand, if one or more samples are covered, they are removed from the class data, and one iteration is finished. The loop runs until class data has been explained, for which the examined network can answer. The pseudo-code shown is shown here 0.

Algorithm 5 Generate rules

```
Require: $f$- Logical Neural Network, $\mathcal{X}$-training data, $c$-class
 1: function GENERATE RULES($f, \mathcal{X}, c$)
 2:     $\mathcal{R}_c \leftarrow \emptyset$
 3:     $f_S, \mathcal{S} \leftarrow$ CollectStats($f, \mathcal{X}, c$)
 4:     while $|\mathcal{X}| \neq 0$ do
 5:         $\pi \leftarrow f(x_1)$
 6:         if $\pi \neq c$ then
 7:             $\mathcal{X} \leftarrow \mathcal{X} \setminus \{x_1\}$
 8:             continue
 9:         end if
10:         $f_P \leftarrow$ Prune($f_S$)
11:         $r^c \leftarrow$ Extract($f_P, c$)
12:         $r^c \leftarrow$ TermsSelection($r^c, \mathcal{X}, \mathcal{S}, c$)
13:         $\mathcal{U} \leftarrow \emptyset$
14:         for each $x \in \mathcal{X}$ do
15:             if $\neg r^c(x)$ then
16:                 $\mathcal{U} \leftarrow \mathcal{U} \cup \{x\}$
17:             end if
18:         end for
19:         if $|\mathcal{U}| = |\mathcal{X}|$ then
20:             Error: Rule is contradictory
21:         end if
22:         $\mathcal{X} \leftarrow \mathcal{U}$
23:         $\mathcal{R}_c \leftarrow \mathcal{R}_c \cup \{r^c\}$
24:     end while
25:     return $\mathcal{R}_c$
26: end function
```

The order of selecting samples for rule extraction and abductive explanation generation does matter here, as the order will determine the number of extracted rules. Let r_1 be a rule explaining only the sample x. In the second iteration of the algorithm 0, for sample y, rule r_2 is extracted, which covers the y sample but also evaluates true on the x sample. If the sample y is used first to extract a rule, r_2 would be extracted and set $\{x, y\}$ would be covered, sample x would be removed from class samples, and r_1 would not be extracted. The reason why the sample x did not provide the rule r_2 immediately is that the extraction is based on the structure of the LNN used and the heuristic used to guide the search, which does not offer an optimal solution and can lead to local optimums. The LNN can conceivably have multiple different rules for explaining a single sample, and NuLog, being a heuristic approach, extracts a sufficient but suboptimal rule. Ideally, it would be best to select the sample whose rule, when extracted, has the highest possible coverage out of all possible rules that could be extracted to gain the minimal possible rule set that explains the class data. Finding such samples is not a trivial problem, but luckily, it can be avoided by generating all rules using any strategy in selecting initial samples, which can be greedy or randomised. Evaluate all coverage sets for class samples and check if there are any pairs of sets where one is a subset of the other. If there exist such two sets, the rule, which is the subset of the larger set, is removed, in the case of the last example with r_1 and r_2, $\mathcal{E}_1^c \subset \mathcal{E}_2^c$, so r_1 is to be removed, where $\mathcal{E}$ is the set of samples explained by a rule.

Chapter 5

Experimental results

The chapter presents the results of evaluating the introduced NuLog rule extraction method on two classification problems. It briefly introduces metrics by which methods will be compared and a basic overview of the datasets used. The methods are measured by qualitative and quantitative characteristics of yielded rules and their respective scopes, along with neat visual representations of the achieved results.

5.1 Rule comparisons

The evaluation and comparison of extracted rules are based on the quality and quantity of generated rules for each class and compared to contemporary methods for learning rules, like CART and RIPPER. Evaluations compare how many rules are generated for each classification being explained. The fewer and shorter rules extracted, the more desirable the technique is, as fewer rules are more interpretable and easier for humans to understand. Miller's law [54], from the field of psychology, states that humans can, at most, hold 7 ± 2 items in short-term memory at once, meaning any rule which is longer than 7 terms significantly lose on direct comprehensibility. Additional forms of presentations for rules and rule sets must be used to get a clear picture of the reasoning behind a decision, which is related to understanding higher dimensional spaces, as each term is a predicate associated with one dimension. The best case scenario is to have a rule set of cardinality 1, or simply a single rule, to explain the entirety of the class data training set. The opposite is to have a one-to-one match of rules to samples; each rule is only good for a single sample and fails to generalise.

Besides the quantity, the length of individual rules is an important factor. The shorter the rule, the simpler it is, and the important features and their respective values are clearer. Rule length is expressed as the number of terms in conjunction. A rule of length n can reference anywhere between $\lceil \frac{n}{2} \rceil$ and n dimensions because for one dimension at most 2 predicates can be present in conjunction, as described in the appendix here A.1. The ideal scenario is to generate rules with as small a number of terms as possible, highlighting the critical features for classification. An optimal number of terms is problem and data-dependent. The opposite is to have two terms for each feature and for all features to be significant, which would create the longest possible rules.

Another interesting metric is the coverage or the rule's scope. Coverage is the percentage of the training and testing set explained by the rules. Besides the quality of the rules, the accuracy of classifiers created from generated rules is of the utmost importance, indicated by precision and recall metrics. Therefore, the trained LNN, NuLog, CART, and RIPPER rule-based classifiers are compared using training and testing sets.

5.2 Flower dataset

5.2.1 Dataset introduction

The flower dataset is a toy dataset with symmetrically spaced non-linearly transformed normal distributions. First introduced in [64] for density estimation, but here it is used for a classification problem, with each leaf being one class. The dataset has two real features in the range $[-5.25, 5.25]$, ideal for representing explanations on a 2D plot, and the target value is the class of the flower, defined as the colour. The size of the generated dataset for method evaluation is 2000 samples for training and 1000 samples for testing purposes, approximately equally split between 8 classes. The training dataset is shown here 5.1, and classes are colour-coded.

The neural network's architecture used for classification consists of four layers with 50, 20, 20, and 8 neurons, respectively. The first is a feature quantiser layer, with a quantisation level of 25, followed by three quantised dense layers. Each layer incorporates a hard tanh as an activation function. Additionally, the sign function is used as a quantiser for each layer with a

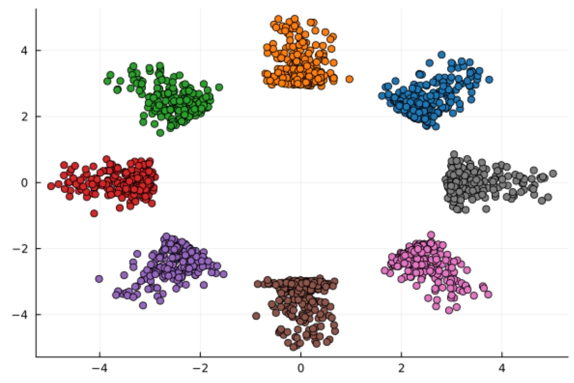


Figure 5.1: Flower training dataset representation

linear proxy STE 3.1, with the window width of 1, and each layer employs batch normalisation. The parameters used in training were AdaBelief as an optimiser and logit cross entropy as the loss function. The training of the BNN was done in 100 epochs.

5.2.2 Results

Interpretability

The following results were achieved and shown in the table 5.1. It compares the total number of rules, mean rule length of individual rules and mean rule coverage of the training data set.

The table contains evaluations for the NuLog extraction method and compares two different post-extraction processing strategies for extracting sufficient rules. More precisely, the two different best-term selection approaches, along with CART and RIPPER methods for reference. The first NuLog method, labelled NL, performs sufficient rule extraction using a heuristic-guided selection, as described in 4.3.1. The other NuLog method, the column marked as eNL, performs sufficient rule extraction employing an exhaustive search for the minimal length rule by finding the shortest rule that does not misclassify and maximises class coverage simultaneously. It must be stated that to avoid any confusion, the initial extraction of both NuLog approaches is identical; the only difference for comparison is the post-extraction phase. These two strategies are selected to be compared on performance qualities. Comparisons are shown for each class, respectively, and for the overall problem for each rule-based classifier.

	Number of rules				Mean rule len				Mean coverage in %			
Class	**NL**	**eNL**	**CA**	**RI**	**NL**	**eNL**	**CA**	**RI**	**NL**	**eNL**	**CA**	**RI**
Blue	1	1	1	13	2.0	2.0	4.0	3.5	100.0	100.0	100.0	7.5
Orange	5	2	1	7	3.2	3.0	3.0	3.4	20.6	49.9	100.0	13.8
Green	3	3	1	13	2.0	2.0	3.0	3.6	67.3	67.3	100.0	7.7
Red	9	6	1	7	3.4	2.7	3.0	3.4	11.1	17.4	100.0	13.8
Violet	3	2	1	14	2.0	2.0	2.0	3.6	52.8	79.2	100.0	7.1
Brown	4	3	1	8	2.8	2.7	3.0	3.5	25.0	33.3	100.0	12.5
Pink	3	2	1	14	3.0	3.0	3.0	3.6	30.7	46.0	100.0	7.1
Gray	17	6	1	8	3.6	2.8	4.0	3.5	6.4	16.6	100.0	12.5
All	**45**	**25**	**8**	**84**	**2.8**	**2.5**	**3.1**	**3.5**	**39.9**	**50.7**	**100.0**	**10.2**

Table 5.1: Interpretability metrics of generated rules for the flower dataset
NL: NuLog - sufficient heuristic ex.,
eNL: NuLog - sufficient exhaustive ex.,
CA: CART, **RI**: RIPPER

All methods indicate the generated rules are understandable based on the number and length of rules, which is greatly helped by the small dimensionality of the problem space. The results are interesting, and by far, the best method regarding the quality of rules is CART, as it has only one rule for each class, the best-case scenario for interpretability. The two NuLog methods are significantly less intuitive for interpreting overall as the total rule number is nearly 6 and 4 times the CART's rule count, respectively, but the number of rules, in this case, greatly varies across classes. The blue class only has one rule of two terms, but on the other hand, the grey class is far more complex as it has 17 rules for heuristic and 6 for exhaustive terms selection, using NuLog.

Comparing the NuLog method with RIPPER indicates that NuLog methods give far better results than RIPPER, which has nearly double the number of rules extracted by heuristic and more than triple the exhaustive approach with NuLog. Interestingly, some classes are easily explained by NuLog and are more complex with RIPPER and vice versa, like blue and grey classes, to highlight a few cases.

Regarding the average length of rules, the best results are given by NuLog methods as it has the minimal average length, closely followed by CART, and the worst again is RIPPER. As expected, the exhaustive selection in NuLog has, on average, the shortest rules, but those results are paid for by slower extraction time compared to the heuristic-guided selection.

The coverage of rules, in table 5.1, shows consistent results, as it is negatively correlated with the number of rules and their length. Simpler and shorter rules result in having fewer rules and higher coverage. Again, after looking at the coverage metric, CART is the best as its result is the ideal case, the single rule for the entire class data set. NuLog's methods show worse coverage than CART, while RIPPER is the worst. Comparing two NuLog methods with different approaches again indicates that the exhaustive search offers better coverage.

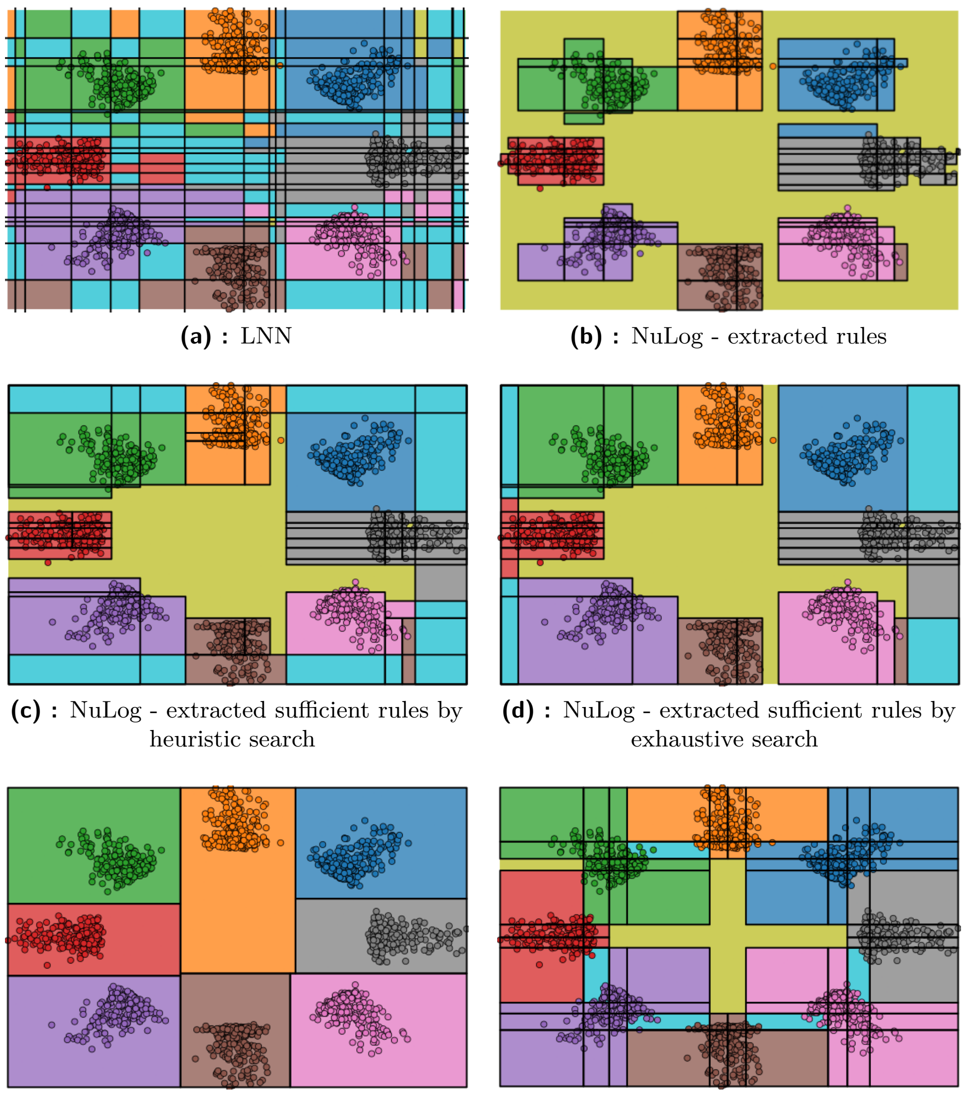
Visualisation

The generated rules and statistics are best presented with visualisations in figure 5.2, with 2D plots. The figure showcases the training data and colour-coded classes, with rectangles as visual representations of classification areas defined by rules. Each area is coloured as the class the method unambiguously determines. The yellow-coloured areas mark values the classifier cannot decide; it gives no answers. On the other hand, teal-coloured areas mark values for which the classifier makes an ambiguous decision; it answers that the area belongs to two or more classes.

The first image 5.3a portrays the decision-making logic of the LNN, indicating classifications for areas with training data and decisions for areas with unseen data where it generalises and hallucinates. The second image 5.3b pictures the result of the network pruning, as the rules shown here are extracted but not subjected to any post-extraction processing, like best terms selection. A nice comparison between 5.3b and 5.3a images. Even though more rules exist in the network, only the ones which are activated and inferred on the training set are extracted. It indicates the reason why extracted rules and networks will not perform identically for the entire problem space and why recall can be significantly worse for rule-based classifiers compared to the original LNN.

The third 5.3c and fourth 5.3d images are the final results of NuLog methods and display how NuLog-extracted rule-based classifiers make decisions. A really interesting comparison is how different best terms selection strategies end up extracting sufficient rules from the same starting rule in various ways and terminating with different rule interpretability metrics, as discussed earlier 5.1. The third image 5.3c is by rules generated with NuLog using a heuristic-led best terms selection, while the fourth 5.3d is by rules generated with NuLog using exhaustive search.

Fifth 5.3e and sixth 5.3f images present the CART's and RIPPER's rule classifiers, respectively. They are serving to compare direct training and learning of rules to extracting from structures like neural networks.



(a) : LNN

(b) : NuLog - extracted rules

(c) : NuLog - extracted sufficient rules by heuristic search

(d) : NuLog - extracted sufficient rules by exhaustive search

(e) : CART

(f) : RIPPER

🟦 Ambiguous answer 🟨 No answer

Figure 5.2: Comparisons of classification boundary plots.

Taking a look at the final NuLog results in images 5.3c and 5.3d, an interesting and rather strange phenomenon sticks out in the form of oddly shaped rectangles. For example, in the image 5.3c, the grey class area is strictly marked for one part while for the rest above and below overlaps with blue and brown areas. That kind of decision classification directly results from post-extraction processing, and at first, the reason for such rules and areas is not intuitive, so a detailed explanation is provided.

Image 5.3b demonstrates how the original extracted rule looked like. The rule at hand is in the form $r^v = (x \geq x_{\text{lo}}) \wedge (x \leq x_{\text{hi}}) \wedge (y \geq y_{\text{lo}}) \wedge (y \leq y_{\text{hi}})$. The area determined by the rule immediately after extraction is $x \in [x_{\text{lo}}, x_{\text{hi}}]$ and $y \in [y_{\text{lo}}, y_{\text{hi}}]$. By removing the term $y \leq y_{\text{hi}}$, the area becomes $y \in [y_{\text{lo}}, \infty)$, while the x axis remains unchanged. The new shorter rule does not misclassify training data, and it is deemed sufficient; no sample exists in the training data to contradict such an expansion.

The example provides a great comparison of how two different best terms selection techniques work. Results in the image 5.3d indicate that the extraction of sufficient rules takes a slightly different path as the overall rules with the exhaustive search are more limited but provide a lot less ambiguous answers.

As the selection of terms is led by selection such that no misclassification occurs, the rectangle results from the fact that there are no samples in the training set to contradict the removal of terms limiting the range for the x axis on a 2D plot. The previous examples show how the extraction of sufficient rules works, headed by consulting the training set. Still, a pitfall of such consultation is that no guarantees can be given on the accuracy of unseen data. The rules might produce rules that are more general and have great coverage; for example, the blue class rules defined by the NuLog method are visually similar to CART's result, shown in image 5.3e. It could also just as easily cause misclassification and errors because of rules that create these bizarre decision areas, like for orange and grey classes. Extracting sufficient rules is highly dependent on how quantisation thresholds, i.e. the weights, are trained in the first layer.

The quality of extracted rules is highly dependent on the inner structure of the LNN and best observable in the image 5.3a. For red and grey classes can be viewed that the trained quantisation thresholds are extremely dense, while for other classes like blue and green, which are explained with a few rules, the thresholds are more spaced apart, as well as that such a high quantisation level is redundant.

It must be stated that visualisations nicely show for which samples the network is not working properly, like the fact that pink class samples are classified as brown. The network's outputs must be taken at face value without explanations like these. It is exactly where the NuLog method shines and proves its worth as a method to make neural networks interpretable by showing for which results the network should not be trusted and must be debugged and rectified.

Regarding the quality of rules and the explanations that follow from the rules, CART shows the nicest, most intuitive and coherent areas, which are easily explainable. RIPPER's rules are slightly worse in comparison, as the shapes are rather unexpected and far from the actual division; they are still easy to understand. NuLog's method produces odd shapes for classification explanations and is the worst of the three methods.

A curious comparison is how each method decides on the area in the centre of the plot. The NuLog method cannot give any answer, as it is limited to training data, and no samples exist to be used to generate a rule, depicted in the second image 5.3b Only a sample existing outside of the central area can provide a rule which and its restrictions loosened by term selection can cover such areas, but as long as it does not cause a different prediction than an LNN. RIPPER created such rules that the area is symmetrically divided between classes. It is biased towards some classes, specifically the four corner classes (blue, red, purple, pink), by giving them larger areas, while it has ambiguous classifications for areas between them. Effectively overlapping and reducing the precision of the extracted rule sets for each class. Interestingly, NuLog methods overlap on the edges and corners, caused by the lack of samples to provide rules and prevent expansion of class areas, but not between class areas. CART marked the central area as one of the classes, similar to the selection of the terms in NuLog, because no contradiction could be found, and the original assumption persisted.

Accuracy

Comparison of classifiers' accuracy in the table 5.2. The comparison is made between the LNN, the classifiers based on the rules extracted from the same network, and the two rule learning methods for reference.

	Train					**Test**				
Class	**NN**	**NL**	**eNL**	**CA**	**RI**	**NN**	**NL**	**eNL**	**CA**	**RI**
Blue	1.00	1.00	1.00	1.00	0.97	1.00	1.00	1.00	1.00	0.98
Orange	1.00	1.00	1.00	1.00	0.97	1.00	1.00	1.00	1.00	0.97
Green	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Red	1.00	1.00	1.00	1.00	0.97	1.00	1.00	1.00	1.00	0.95
Violet	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Brown	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Pink	0.99	0.99	0.99	1.00	1.00	0.99	0.99	0.99	1.00	1.00
Gray	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
All	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**

Table 5.2: Accuracy metrics for the LNN and rule-based classifiers
NL: NuLog - sufficient heuristic ex.,
eNL: NuLog - sufficient exhaustive ex.,
NN: LNN, **CA**: CART, **RI**: RIPPER

The results show the quality of the rule extraction method and prove its worth. Results closely follow the accuracy of the original LNN and CART while being better than RIPPER. The flower problem at hand is relatively simple and does not punish overfitting the data on the training set. Out of the interpretable methods, CART is the best, with the best accuracy accompanied by the simplest possible explanation for all classes. It nicely indicates how a model-agnostic method using a surrogate model works, introduced here 2.1 to explain how the black-box model like LNN works. CART is closely followed by NuLog as the accuracy and the quality of generated rules and related explanations are slightly worse compared to it. Out of the two post-extraction approaches, accuracy is nearly identical, but in favour of the exhaustive search, regarding interpretability, the exhaustive search takes the win as expected. Heuristic-led search provides worse rules, but extraction and processing are significantly faster.

5.3 MNIST dataset

5.3.1 Dataset introduction

The MNIST [20] dataset, a standardised and widely used dataset in the machine learning world, has been used to evaluate and measure the performance of the NuLog method. MNIST stands for modified National Institute of Standards and Technology database and is a collection of handwritten digits, from 0 to 9, each digit represented as a grayscale image of size 28×28 pixels. It is an ideal dataset for testing the explanations generation method as it poses several key challenges to overcome the variance of handwriting styles, noise, and ambiguities in the images. Here are two representative samples of digit zero 5.3; the dataset supplied the first, and the second is transformed into two discrete values.

The same parameters for training neural networks on MNIST are used for the flower problem LNN, defined here 5.2.1. The only difference is in the layers, as it uses four layers with 784, 784, 392, and 10 neurons, respectively, and the quantisation level is now 1 creating 2 separate bins for each pixel. The network will train threshold values for binarising pixel intensity values.

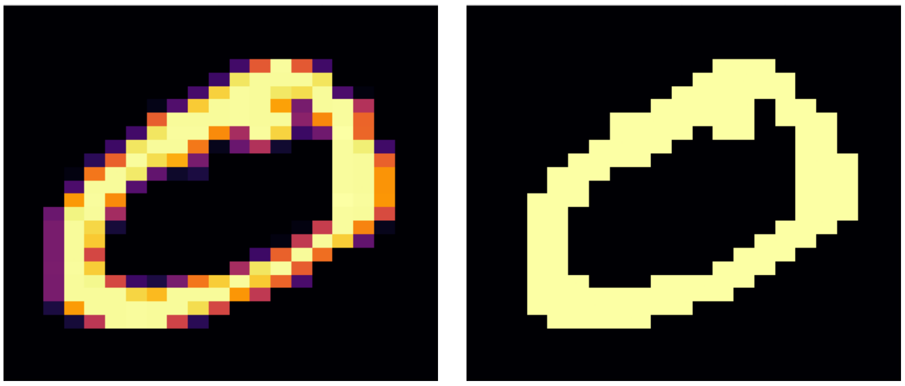


Figure 5.3: Examples of MNIST samples of digit zero

5.3.2 Results

Interpretability

Results of the rule-generating methods achieved are shown in the table 5.3 below, along with contemporary rule learning methods' results. The table compares techniques by the number of rules generated, the average rule length, and coverage expressed as the average percentage of the training set covered by generated rules. It is shown for each class respectively and for the overall rule-based classifiers.

The table contains evaluations for two NuLog extractions, CART and RIPPER methods. The first NuLog method uses heuristic-guided best terms selection, denoted as NL. The second NuLog evaluation, here denoted as gNL, employs a greedy terms selection, which takes a full rule and then greedily selects the term that can be deleted whose removal will not cause the remaining rule to misclassify and will maximise coverage; this is repeated until all terms which fit the condition above are removed.

The difference between the techniques is staggering when comparing the number of generated rules. The NuLog method with heuristic selection extracts around three times the number of rules generated by CART and just shy of eleven times more than the RIPPER method. The greedy best terms selection approach offers a significantly better solution regarding interpretability, as the number of extracted rules is nearly halved. Still, this approach suffers from significantly slower extraction time compared to heuristic-guided best terms selection but is faster than a full, exhaustive one, which guarantees

	Number of rules				Mean rule len				Mean coverage in %			
C	**NL**	**gNL**	**CA**	**RI**	**NL**	**gNL**	**CA**	**RI**	**NL**	**gNL**	**CA**	**RI**
0	222	142	58	23	16.5	9.3	9.6	5.7	1.0	2.7	1.7	11.6
1	127	33	42	16	30.0	14.3	10.6	6.2	1.6	15.6	2.4	13.0
2	369	224	104	35	17.5	9.9	10.2	5.5	0.4	1.7	0.9	5.8
3	364	208	103	27	17.1	9.5	10.4	6.5	0.4	1.6	0.9	7.5
4	298	197	96	21	19.0	10.9	10.6	6.1	0.5	2.4	1.0	12.4
5	379	306	89	22	17.4	9.9	10.3	6.3	0.3	0.7	1.1	8.0
6	183	125	69	24	16.0	10.1	10.0	6.1	1.0	4.5	1.4	9.3
7	211	129	68	17	22.0	10.0	11.0	6.4	1.2	3.4	1.4	16.6
8	447	222	116	60	17.2	10.2	10.0	5.9	0.3	1.6	0.8	2.1
9	421	206	86	30	24.6	10.4	10.9	7.5	0.3	1.7	1.1	4.1
A	**3021**	**1792**	**831**	**275**	**19.7**	**10.5**	**10.4**	**6.2**	**0.6**	**3.0**	**1.1**	**7.9**

Table 5.3: Interpretability metrics of generated rules for MNIST
NL: NuLog - sufficient heuristic ex.,
gNL: NuLog - sufficient greedy ex.,
CA: CART, **RI**: RIPPER

the optimal solution.

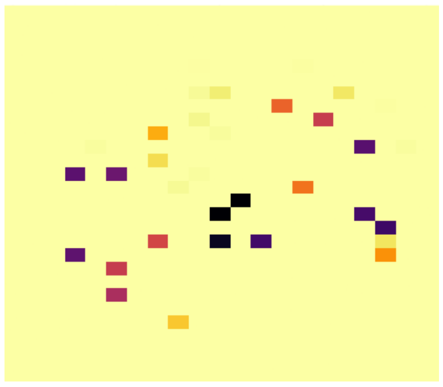
The NuLog heuristic selection method yields longer rules, nearly two times longer on average than CART and even more than three times longer rules than RIPPER. The greedy term selecting NuLog generates nearly identical rule length to CART on average, but RIPPER generates the shortest rules over all.

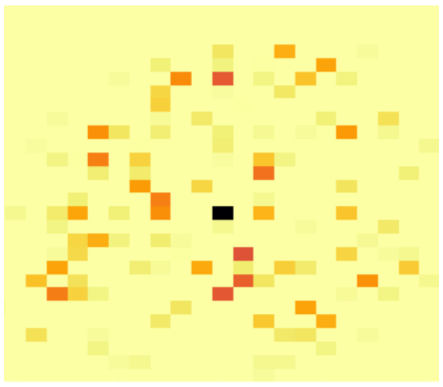
The coverage statistics are in table 5.3. Understandably, longer rules are less general. Subsequently, more rules are required to cover the same sample size. However, an interesting result is gained for class 1, which shows that the greedy NuLog version extracted 21% less rules than CART, and the average coverage is significantly higher than CART's for the class. Achieved by extracting rules where all rules have similar coverage percentages and explain approximately equal portions of the training set. For the above reason, the overall result breaks the negative correlation between the number of rules and rule set coverage. NuLog's version with greedy selection has a higher number of rules and higher coverage than CART.

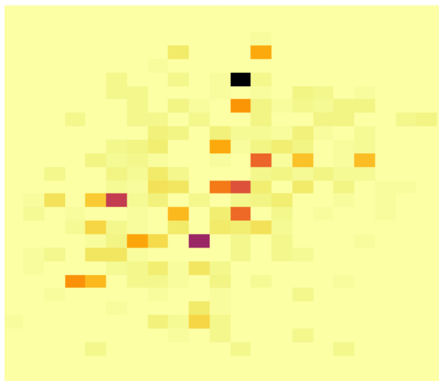
Visualisation

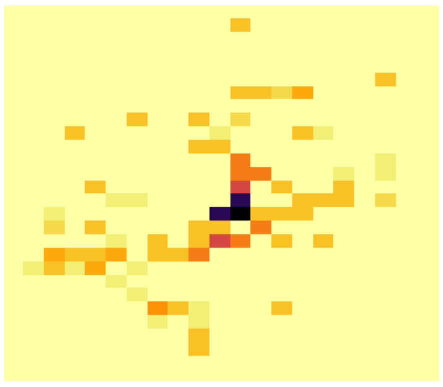
Interpreting these rules at face value, RIPPER stems as the best method regarding interpretability, as fewer simpler rules are preferable to more convoluted ones. The issue that arises here is that looking at class zero, for example, is impractical and nearly impossible to take all 23 rules into account, at the same time, as just a list of predicates. Visual representation of said rules is required to compare interpretability effectively.

Representations are done in such a way that they indicate the relevant pixels and give information about the data in the form of a heatmap. The heatmap has three main colour shades; the yellow pixels indicate that areas of the full 28×28 image are not referenced by any predicate in the extracted rules. Dark and light pixels are portrayed as variations of black and purple, as well as various shades of orange. The extracted rules reference these pixels and are important for decision-making. The pixels marked as dark mark areas are expected to be dark, meaning they have an intensity of 0 or close to 0, while light expects their intensity to be as high as possible, as 1 or as close to 1. The following images are visualisations of decision rule sets for the same class of digit zero, achieved by different techniques 5.4.


(a) : Important pixels by ex. suff. rules with heuristic elimination


(b) : Important pixels by ex. suff. rules with greedy backwards elimination


(c) : Important pixels by CART method


(d) : Important pixels by RIPPER method

🟧Light ⬛Dark ⬜Not checked

Figure 5.4: Class 0 visualisations of important pixels for classification

The first 5.5a and second 5.5b images are visual representations of NuLog extracted rules. Image 5.5a is the heuristic guided search, while 5.5b shows the greedy best terms selection, which generates shorter rules. Third 5.5c and fourth images 5.5d are for CART and RIPPER generated rule sets. It is interesting to notice that the approach which generated longer rules requires significantly fewer terms overall, specifically for this problem pixels, out of the full 28×28 image to explain the training set. The result is expected as the pixels are selected as part of the post-extraction processing led by heuristic sort, which will always generate the same order, but the misclassification will determine the length, resulting in always the same terms being selected for the sufficient rule. Overall, the number of critical features is considerably less than other methods, also supported by the numbers in the table 5.4.

	Num. of features				**Percentage of full image**			
C	**NL**	**gNL**	**CA**	**RI**	**NL**	**gNL**	**CA**	**RI**
0	35	116	165	72	4.46	14.8	21.05	9.18
1	43	108	144	54	5.48	13.8	18.37	6.89
2	49	126	216	106	6.25	16.1	27.55	13.52
3	42	134	217	95	5.36	17.1	27.68	12.12
4	40	129	206	84	5.10	16.5	26.28	10.71
5	45	127	202	83	5.74	16.2	25.77	10.59
6	45	117	183	76	5.74	14.9	23.34	9.69
7	43	131	176	67	5.48	16.7	22.45	8.55
8	47	127	214	142	5.99	16.2	27.30	18.11
9	60	116	197	87	7.65	16.4	25.13	11.10
A	**60**	**167**	**308**	**328**	**7.65**	**21.3**	**39.29**	**41.84**

Table 5.4: Key features metrics for the MNIST
NL: NuLog - sufficient heuristic ex.,
gNL: NuLog - sufficient greedy ex.,
NN: LNN, **CA**: CART, **RI**: RIPPER

Here, a trade-off in interpretability quality is presented between having more numerous rule sets with longer rules but fewer important features versus having fewer and shorter rules with significantly more important features. If fewer important features are deemed a higher priority, the NuLog method is the best, and RIPPER is the worst. Still, if the number and length of rules take higher precedence, the roles are reversed, RIPPER being the best and NuLog the worst method. CART is found to be the middle ground in this case. The heuristic strategy result is consistent with the nature of the NuLog method, as it explains a neural network for which the first layer's quantifiers during training can have threshold values outside the expected value ranges. Therefore, these predicates are meaningless as part of the explanation and should not be considered. The procedure for identification and propagation of constants is explained in detail in the appendix here B. However, the greedy selection strategy in NuLog indicates properties similar to CART and RIPPER with more relevant pixels than the heuristic approach in NuLog.

It is interesting to see and interpret the "neural network's decision process on differentiating digits, as shown in the images 5.4. For the class of digit zero, it is clear that it looks for dark blobs in the middle of the image, with a patch of lighter pixels on all sides surrounding the dark blob and, again, a dark patch on both sides of the lighter patches. An interpretation like this indicates that it is memorising the parts characteristic of a particular digit enough to distinguish it from other digits. A similar result can be seen using the RIPPER method, while nothing analogous can be done to interpret the CART's visualisation of zero's rule set.

Accuracy

The most important comparison is the comparison of accuracies and how the methods behave on unseen data, with precision and recall metrics shown below 5.5. The comparison between the rule-based classifiers and the neural network explained by the NuLog method is also shown.

	Train					**Test**				
Metric	**NN**	**NL**	**gNL**	**CA**	**RI**	**NN**	**NL**	**gNL**	**CA**	**RI**
Precision	1.00	1.00	1.00	1.00	1.00	0.98	0.95	0.95	0.95	0.96
Recall	0.99	0.99	0.99	1.00	0.98	0.93	0.87	0.82	0.77	0.91

Table 5.5: Accuracy metrics for the neural network and rule-based classifiers
NL: NuLog - sufficient heuristic ex.,
gNL: NuLog - sufficient greedy backwards ex.,
NN: LNN, **CA**: CART, **RI**: RIPPER

The results here show what was already known: neural networks are the best classifiers overall. CART is the best for training data. Interestingly, RIPPER shows rather poor accuracy in both training and testing data. NuLog's rules have significant performance on the training set, which is expected as the training set is consulted in extraction, which explains the difference in training performances. For the training data, the NuLog's rules follow the network's performance, but as the NuLog method offers no guarantees for unseen data, accuracies differ significantly on testing. All methods are precise, with precision on a testing set of at least 95%, but recall stats for NuLog methods are worse as they can only identify around 82% to 87% of relevant results. CART has a slightly worse recall than heuristic selection NuLog, and RIPPER has the best recall out of the rule-based classifiers but is worse than the LNN.

The reason for bad recall metrics for NuLog methods compared to the neural network's metric used to extract said rules stems from the fact that only the training set is used in the extraction, each sample serving as a "seed" to extract an explaining rule. The extraction would have to rely only on the neural network's elements and extract all possible rules that could be inferred in the network to gain a better recall metric. An alternative would be to perform an exhaustive extraction of rules based on the entire possible problem space, effectively brute-forcing all significant samples of the problem.

These results indicate that the best results are achieved by separating concerns, which means having the best tool for each part of the job. Have the best classifier, a neural network, accompanied by rule sets and a method to generate the same rules, which is NuLog's rules extraction technique. The selection of the best terms and techniques to use depends on which aspect is prioritised. If interpretability is required, then some form of greedy or randomised criteria for selection is best, of course, if the problem size does not allow an exhaustive search. On the other hand, if faster sufficient rule extraction is prioritised, a heuristic-guided terms selection is the best with a worse but satisfiable level of interpretability.

5.3.3 Comparing the effect of training set size

An interesting result is shown when different sizes of the training are used to train the QNN, convert it to LNN and extract rules, as shown in the table below 5.6.

	Number of rules			**R-S ratio in %**		
Tr. set size	**NL**	**CA**	**RI**	**NL**	**CA**	**RI**
100% - 60000	15556	3330	885	25.9	5.6	1.5
50% - 30000	3021	831	275	10.1	2.8	0.9
25% - 15000	4466	1204	360	29.8	8.0	2.4

Table 5.6: Number of extracted sufficient rules for different sizes of training sets
R-S: Ratio of the number of rules to training set size
NL: NuLog, **CA**: CART, **RI**: RIPPER

These results indicate that the number and quality of extracted rules via training set greatly depends on its size. The network does not generalise, but it tries to memorise training set samples as much as possible, supporting the previous conclusion for bad recall performance, as the rule extraction requires all possible samples in order to extract complete knowledge from the network in this fashion. An interesting indicator is that both conventional rule-learning methods behave similarly to the neural network.

5.3.4 Results on binarised MNIST dataset

The binarised MNIST dataset is used to evaluate and compare two differently trained networks, one classically and the other more robustly trained, on an augmented training set. Additionally, with the binarisation of the data set, the training of feature quantisation is no longer necessary. A pre-binarised data set with a threshold of 0.5 means all three methods work with the same initial number of features. In other words, the binarisation of data initially makes the NuLog's extraction work with all possible pixels equally relevant. The data binarised is the same as the previous MNIST evaluation; the only difference in training is the missing first layer used for feature quantisation. The robust network training is performed such that 10% of pixels are selected randomly, and their respective values are flipped from light to dark and vice versa. Note that the augmentation was done solely during neural network training and not for the NuLog's extraction. The following metrics describing the generated rules are collected 5.7. The comparison shows more or less the

Number of rules				Mean rule len				Mean coverage in %			
NL	**aNL**	**CA**	**RI**	**NL**	**aNL**	**CA**	**RI**	**NL**	**aNL**	**CA**	**RI**
3825	3357	831	249	28.34	27.64	10.35	6.68	0.47	0.52	0.85	9.92

Table 5.7: Interpretability metrics of generated rules for the binarised MNIST
NL: NuLog, **CA**: CART, **RI**: RIPPER,
aNL: NuLog using an LNN trained with an augmented set,

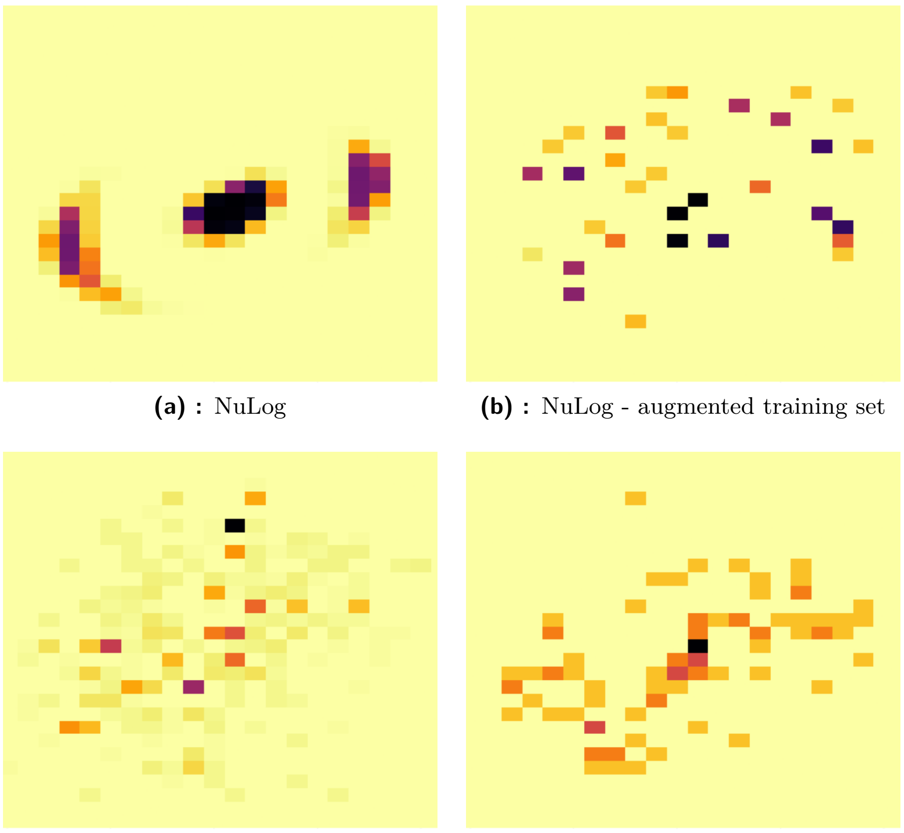
same relationships and results as the regular MNIST evaluation. A rather interesting comparison is that the more robustly trained network indicates improvements regarding the number of rules and their length. The overall rule set size is decreased by around 12% on average. However, this is not uniform across all classes but greatly differs, and for some classes, there is a negligible increase. The rule length is also slightly reduced. The coverage indicates again the same negative correlation between the number of rules required and the coverage, which is indicated by the table 5.7

The key features and their respective numbers and comparisons are shown below 5.8. Again, the NuLog is the best when selecting fewer key features, and the training method significantly affects rule generation, which is expected as the rules are extracted from the trained network.

Number of features				Image percentage			
NL	**aNL**	**CA**	**RI**	**NL**	**aNL**	**CA**	**RI**
265	65	308	346	33.8	8.3	39.3	44.1

Table 5.8: Key features statistics for the binarised MNIST
NL: NuLog, **CA**: CART, **RI**: RIPPER,
aNL: NuLog using an LNN trained with an augmented set,

Visualisations in 5.5 illustrate a noticeable difference in the extracted rules between robust 5.5b and non-robustly trained neural networks 5.5a. The classically trained network neatly concentrates on particular features and parts of the image. CART 5.5c and RIPPER 5.5d methods are consistent and do not show any significant differences between the two datasets, which is expected of such methods. In contrast, the more robust network has far sparser key pixels that are more spread out, indicating more options and combinations when selecting features, which results in nicer and shorter rules with greater coverage.



(a) : NuLog **(b) :** NuLog - augmented training set

(c) : CART **(d) :** RIPPER

🟧Light ■Dark □Not checked

Figure 5.5: Binariesd class zero visualisations

The results support the conclusions made for the classic MNIST dataset: the rule extraction method generated more rules but fewer significant features. In comparison, rule learning methods generated fewer rules but with more key features. Also, the more robust network showcases how it learned to find the 8.3% key pixels to correctly classify to a significant accuracy, with accuracy comparisons shown in the next table 5.9.

	Train						Test					
Metric	**NN**	**NL**	**aNN**	**aNL**	**CA**	**RI**	**NN**	**NL**	**aNN**	**aNL**	**CA**	**RI**
Precision	1.00	1.00	0.99	0.99	1.00	0.88	0.98	0.95	0.98	0.93	0.97	0.82
Recall	1.00	1.00	0.99	0.99	0.99	0.72	0.95	0.85	0.94	0.84	0.91	0.61

Table 5.9: Accuracy metrics for neural networks and rule-based classifiers
NN: LNN, **NL**: NuLog, **CA**: CART, **RI**: RIPPER,
aNL: NuLog using an LNN trained with an augmented set,

5.4 Experiment result conclusions

From the conducted experiments, conclusions on the provided explanations can be drawn and referencing the metrics introduced here 2.1.

- **Fidelity**: Explanations are truthful to the LNNs as they are directly extracted based on their structure but guaranteed only for the training data.
- **Consistency**: Exact rules extracted from two LNNs differently trained will be different but provide similar explanations, i.e. they will cover similar areas.
- **Stability**: NuLog is deterministic, not randomised, meaning the same rules will be extracted every time.
- **Comprehensibility**: Explanations provided are easy to interpret and convert to natural language [58] but are related to the complexity of the problem and may require various visualisation techniques
- **Certainty**: NuLog reflects the certainty of the machine learning model, as it is not model agnostic and is based on the inner structure of the LNN.
- **Degree of importance**: The extracted sufficient rules indicate the importance of each feature.
- **Novelty**: If test data comes from different distributions from training, the NuLog method does not guarantee a valid or any explanation at all, and rule-based systems differ from the network's output.
- **Representativeness**: Extracted rules are scoped and often cannot provide high coverage; at least it is worse than CART and RIPPER, but this depends on the data.

Chapter 6

Conclusion

This chapter concludes the thesis and introduction of the NuLog method for extracting rules from neural networks with discrete weights. More specifically, converting networks with discrete weights into neural networks with logical expressions for activation functions and extracting an explanatory rule. The chapter lists the advantages and disadvantages of the method. Conclusions are drawn from the experimental results on two classification problems based on flower and MNIST datasets.

The conclusion is that the extraction of decision rules for a prediction of a single sample, in other words, abductive explanation, can be done relatively easily when converting M-of-N into conjunction and disjunctions by pruning the network by heuristic estimation of neuron importance. The pruning process led by activation statistic heuristics does not provide an optimal solution, and often extracted rule contains all possible features. The main reason the post-extraction phase is necessary is to perform feature selection and provide a sufficient rule without redundant terms, which is more general with ideally more coverage.

The advantage of the explanations provided by the NuLog method is that it utilises discrete neural networks, which have great advantages over full precision ones. They are perfect for low-power and low-hardware edge devices and ideal for many security purposes [75]. Reduced precision networks are being developed to become as good as full precision networks regarding predictive power [31]. Additionally, discrete networks have the advantage of fairly straightforward expressions as logical M-of-N rules. Another advantage is that the quantisation of data is learned and not determined ahead and biased; essentially, it is done automatically except for the estimation of a sufficient number of bins to be quantised in. Regarding the extraction of exact rules with network pruning, it provides valid explanations consistent with the network's predictions. After post-extraction processing, the sufficient rules indicate important features necessary for correct classification.

While discrete neural networks have benefits, their massive disadvantage is that they are far behind full-precision deep neural networks in predictive performance [75]. NuLog requires discrete neural networks with binary activations and ternary weights, which limits the scope of possible models and neural networks to be explained. The NuLog pruning procedure uses an activation statistic of individual neurons to provide neuron importance estimation and prune less important neurons which provides a suboptimal solution. To find an optimal solution some form of exhaustive search and pruning would have to be employed. The pruning process does not perform feature selection well, and post-extraction processing with best terms selection of the extracted rule is necessary.

Further research will investigate the robustness of networks and how many different rules could be extracted for a single sample, as well as different points of view on network pruning and importance estimation, including MILP solvers, randomised selection and data perturbation. Research could be expanded as well with different training implementations for BNNs, like the approach in [21], and implemented in the student Bachelor's thesis [59], to compare activation statistic analysis and extracted rules of two differently trained BNNs.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[2] Khadija Mohammad Al-Aidaroos, Azuraliza Abu Bakar, and Zalinda Othman. Naïve bayes variants in classification learning. In *2010 International Conference on Information Retrieval Knowledge Management (CAMP)*, pages 276–281, 2010.

[3] José Maria Alonso, Ciro Castiello, and Corrado Mencar. Interpretability of fuzzy systems: Current research trends and prospects. In *Handbook of Computational Intelligence*, 2015.

[4] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995. Knowledge-based neural networks.

[5] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data, 2018.

[6] Meraj Farheen Ansari, Bibhu Dash, Pawankumar Sharma, and Nikhitha Yathiraju. The impact and limitations of artificial intelligence in cybersecurity: A literature review. *IJARCCE*, 11:81–90, 10 2022.

[7] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models, 2019.

[8] Theo Araujo, Natali Helberger, Sanne Kruikemeier, and Claes de Vreese. In ai we trust? perceptions about automated decision-making by artificial intelligence. *AI SOCIETY*, 35, 09 2020.

[9] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3319–3327, 2017.

[10] Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. SIRUS: Stable and Interpretable RUle Set for classification. *Electronic Journal of Statistics*, 15(1):427 – 505, 2021.

[11] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons, 2013.

[12] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

[13] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch, 2018.

[14] Sophie Burkhardt, Jannis Brugger, Nicolas Wagner, Zahra Ahmadi, Kristian Kersting, and Stefan Kramer. Rule extraction from binary neural networks with convolutional rules for model validation. *Frontiers in artificial intelligence*, 4:642263, 2021.

[15] John M Carroll. Why should humans trust ai? *Interactions*, 29(4):73–77, 2022.

[16] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 115–123. Morgan Kaufmann, San Francisco (CA), 1995.

[17] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2016.

[18] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016.

[19] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Regularized binary network training, 2020.

[20] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[21] Xiang Deng and Zhongfei Zhang. An embarrassingly simple approach to training ternary weight networks, 11 2020.

[22] Jailton C. Ferreira. The cardinality of the set of real numbers, 2013.

[23] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously, 2019.

[24] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[25] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3), September 2008.

[26] LiMin Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8):1114–1124, 1994.

[27] Johannes Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27:139–172, 1997.

[28] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. *Foundations of rule learning*. Cognitive Technologies. Springer, Berlin, Germany, 2012 edition, November 2012.

[29] Johannes Fürnkranz and Tomáš Kliegr. A brief overview of rule learning. In *International symposium on rules and rule markup languages for the semantic web*, pages 54–69. Springer, 2015.

[30] Ekta Garg and Meenakshi Bansal. A survey on improved apriori algorithm. *International Journal of Engineering Research and Technology*, 2, 2013.

[31] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[32] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile, 2018.

[33] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[34] Brandon M. Greenwell, Bradley C. Boehmke, and Andrew J. McCarthy. A simple and effective model-based variable importance measure, 2018.

[35] Britta Hale, Douglas L Van Bossuyt, Nikolaos Papakonstantinou, and Bryan O'Halloran. A zero-trust methodology for security of complex systems with machine learning components. In *International design engineering technical conferences and computers and information in engineering conference*, volume 85376, page V002T02A067. American Society of Mechanical Engineers, 2021.

[36] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Introduction. In *The Elements of Statistical Learning*, Springer series in statistics, pages 1–8. Springer New York, New York, NY, 2009.

[37] Petr Hájek, Martin Holeňa, and Jan Rauch. The guha method and its meaning for data mining. *Journal of Computer and System Sciences*, 76(1):34–48, 2010. Special Issue on Intelligent Data Analysis.

[38] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models, 2018.

[39] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On relating explanations and adversarial examples. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[40] International Organization for Standardization. Iso/iec/ieee 26531:2023 – systems and software engineering — content management for product life cycle, user and service management information for users, 2023. Geneva, Switzerland.

[41] Kai Jia and Martin Rinard. Efficient exact verification of binarized neural networks, 2020.

[42] Ravi Kannan and Clyde L. Monma. On the computational complexity of integer programming problems. 1978.

[43] Richard Karp. Reducibility among combinatorial problems. volume 40, pages 85–103, 01 1972.

[44] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[45] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR, 10–15 Jul 2018.

[46] Raymond ST Lee. *Artificial intelligence in daily life.* Springer, 2020.

[47] Michael Lent, William Fisher, and Michael Mancuso. An explainable artificial intelligence system for small-unit tactical behavior. pages 900–907, 01 2004.

[48] Anastasia-M. Leventi-Peetz and Kai Weber. Rashomon effect and consistency in explainable artificial intelligence (xai). In Kohei Arai, editor, *Proceedings of the Future Technologies Conference (FTC) 2022, Volume 1*, pages 796–808, Cham, 2023. Springer International Publishing.

[49] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer, 2000.

[50] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm, 2018.

[51] J.W. Lloyd and R.W. Topor. Making prolog more expressive. *The Journal of Logic Programming*, 1(3):225–240, 1984.

[52] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[53] David A. Medler and Michael R.W. Dawson. Using redundancy to improve the performance of artificial neural networks. 1999.

[54] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[55] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning, 2019.

[56] Christoph Molnar. *Interpretable Machine Learning.* 2 edition, 2022.

[57] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. *Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges*, page 417–431. Springer International Publishing, 2020.

[58] Aikaterini Mpagouli and Ioannis Hatzilygeroudis. Converting first order logic into natural language: A first level approach. In *Current Trends in Informatics: 11th Panhellenic Conference on Informatics, PCI*, pages 517–526, 2007.

[59] Viktor Nezveda. Training of binary-ternary neural networks, 2024.

[60] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks, 2016.

[61] Thanh T. Nguyen, Charles Soussen, Jérôome Idier, and El-Hadi Djermoune. Np-hardness of 0 minimization problems: revision and extension to the non-negative setting. In *2019 13th International conference on Sampling Theory and Applications (SampTA)*, pages 1–4, 2019.

[62] Dimitris Papadimitriou and Swayambhoo Jain. Data-driven low-rank neural network compression, 2021.

[63] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017.

[64] Tomas Pevny, Vasek Smidl, Martin Trapp, Ondrej Polacek, and Tomas Oberhuber. Sum-product-transform networks: Exploiting symmetries using invertible transformations, 2020.

[65] Alun Preece. Asking 'why'in ai: Explainability of intelligent systems–perspectives and challenges. *Intelligent Systems in Accounting, Finance and Management*, 25(2):63–72, 2018.

[66] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016.

[67] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning, 2016.

[68] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[69] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[70] M. Robnik-Sikonja and Marko Bohanec. Perturbation-based explanations of prediction models. In *Human and Machine Learning*, 2018.

[71] Lloyd S. Shapley. *A Value for N-Person Games.* RAND Corporation, Santa Monica, CA, 1952.

[72] Taylor Simons and Dah-Jye Lee. A review of binarized neural networks. *Electronics*, 8(6), 2019.

[73] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[74] Dylan Slack, Sorelle A. Friedler, Carlos Scheidegger, and Chitradeep Dutta Roy. Assessing the local interpretability of machine learning models, 2019.

[75] Matteo Spallanzani, Gian Paolo Leonardi, and Luca Benini. Training quantised neural networks with ste variants: the additive noise annealing algorithm, 2022.

[76] VA Stafford. Zero trust architecture. *NIST special publication*, 800:207, 2020.

[77] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019.

[78] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[79] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354, 1994.

[80] Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In Jan Komorowski and Jan Zytkow, editors, *Principles of Data Mining and Knowledge Discovery*, pages 78–87, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[81] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets, 2019.

[82] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. Nisp: Pruning networks using neuron importance score propagation, 2018.

[83] Jindi Zhang, Yang Lou, Jianping Wang, Kui Wu, Kejie Lu, and Xiaohua Jia. Evaluating adversarial attacks on driving safety in vision-based autonomous vehicles. *IEEE Internet of Things Journal*, 9(5):3443–3456, 2022.

Appendix A

Extract the rule

Rule extracting is accomplished by converting the pruned network to a single-layer network, which must provide identical predictions as the original input network, $f^n \circ f^{n-1} \circ \cdots \circ f^1 = f^{n,n-1,\cdots,1}$. The conversion of a deep into a single-layer neural network is designed such that the resulting layer holds the entirety of the classification decision logic from the layers that come before it. For that reason, pruning the original network and customising it for a particular sample, selected ahead of time for abductive explanation extraction, is done. The conversion is done layer by layer, from the final layer, taking the preceding layer and merging it with the final into a new final layer, repeated until reaching the initial layer. At this point, only a single layer remains, which holds the entire logic of the initial network. The conversion process can be done on the full LNN but is extremely computationally inefficient, and the resulting rule is in the form of nested M-of-N rules, which are completely incomprehensible and uninterpretable, so a pruned network 4.3 is used.

The first neuron of the final layer, in 4.1b, will be used as an example to demonstrate and clarify the conversion procedure. The expression $A \wedge \bar{B} \wedge \bar{C}$ is examined, and all literals, which directly reference neurons of the preceding layer, are replaced with the logical expressions from their respective neurons. The expression now becomes $(\bar{a} \wedge b) \wedge (\bar{a} \wedge b) \wedge (\bar{a})$. With the replacement of literals, a new logical expression is created through simplification procedures to attain the minimum or, as small as possible, logical expression regarding the number of terms and operations. The expression above, after simplifications, is equivalent to $\bar{a} \wedge b$. The simplifications, in short, are done by filtering the terms being put into conjunction or disjunction depending on the original expression into which the layer is merged. They are explained later in the appendix in greater detail here A.1.

An identical procedure is performed for all the neurons in the final layer, creating a new final layer, with the result of the first iteration shown here 4.1c, and iteratively repeated until only a single layer remains. When a single layer is reached, the rule explaining a certain classification is simply read from the neuron representing the class of interest. Other neurons hold negation of the classification rule, interpreted as what feature values do not cause a certain classification The rule for the sample $x = \begin{bmatrix}10\end{bmatrix}, f(x) = 1$ is $\bar{a} \wedge b$, which is $\neg(x_1 \leq 9) \wedge (x_1 \geq 3)$, simplified to just $x_1 > 9$. The provided explanation states that the network is classifying the sample as class 1 because the value of the first and only feature is greater than nine.

Convert rule set. Convert rule set procedure accepts a rule set or list of rules in disjunction and the layer which precedes the one to which the supplied rule set belongs. The conversion is done so that each rule is mapped and stored as a new rule set. Converted rules are simplified such that the resulting expression returned is equivalent to the original but as small as possible. Simplifications are done on the fly; as each rule is converted, it is added to the set and the intermediate set is simplified. The specifics of the exact implementation of these simplifications are explained later in the appendix here A.1. The procedure for converting conjunctions is analogous to the conversion procedure for disjunctions; the only difference is the simplifications, which are related to conjunctions.

Algorithm 6 Convert a set of rules - disjunction

```
Require: $\mathcal{R}$-ruleset logical expression, $l$-previous layer
 1: function CONVERT($\mathcal{R}, l$)
 2:     $\mathcal{P} \leftarrow \emptyset$
 3:     for each r $\in \mathcal{R}$ do
 4:         $r' \leftarrow$ Convert(r, $l$)
 5:         $e \leftarrow$ SimplifyOR($\mathcal{P}, r'$)
 6:         if $e = \mathbf{T}$ then
 7:             return T
 8:         end if
 9:     end for
10:     return Ruleset($\mathcal{P}$)
11: end function
```

A.1 Simplifications

Simplifications are operations performed over logical expressions by applying the following set of replacement rules in logic.

1. **Identity Law:**

   $A \wedge \top = A$, $A \cap U = A$
   $A \vee \bot = A$, $A \cup \varnothing = A$

2. **Domination Law:**

   $A \wedge \bot = \bot$, $A \cap \varnothing = \varnothing$
   $A \vee \top = \top$, $A \cup U = U$

3. **Idempotent Law:** $A \wedge A = A$, $A \cap A = A$

4. **Commutative Law:** $A \wedge B = B \wedge A$, $A \cap B = B \cap A$

5. **Associative Law:** $A \wedge (B \wedge C) = (A \wedge B) \wedge C$

6. **Distributive Law:** $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

7. **Absorption Law:** $A \wedge (A \vee B) = A$, $A \cap (A \cup B) = A$

8. **De Morgan's Laws:**

   $\neg(A \wedge B) = \neg A \vee \neg B$, $(U \setminus (A \cap B)) = (U \setminus A) \cup (U \setminus B)$
   $\neg(A \vee B) = \neg A \wedge \neg B$, $(U \setminus (A \cup B)) = (U \setminus A) \cap (U \setminus B)$
   $\neg(m \text{ of } (A_1, A_2 \cdots, A_n)) = (n - m + 1) \text{ of } (\neg A_1, \neg A_2 \cdots, \neg A_n)$

The principle on which the simplifications of conjunctions and disjunctions work is analogous to one another, and one can be performed by the other, as defined by DeMorgan's laws 8. For that reason, only conjunction simplifications are examined.

Simplifications work by taking all the terms meant to be placed in conjunction with one another, going through them individually, and using them to filter out other terms by a set of laws, stated above. The ordering of terms in conjunction is irrelevant as a conjunction is a commutative operation 4. Simplifications could take all the terms and then perform filtering, but a more efficient way of implementation is to simplify terms one by one as they are generated. This way, the whole process can be terminated if a contradiction is reached at some point in the generation. The only issue is time efficiency, which depends on the ordering of the terms to be processed and put into conjunction. An example of using the simplifications is best seen in the token example shown here A, and visually in 4.1 transforming a pruned network into a single layer network.

The simplification procedures have been implemented such that they separate the terms by their type into literals, predicates, rule sets, and M-of-N rules, and the function for each type carries out the simplification rules to discard the new term to be added, find a contradiction or remove already added terms. In case a constant value is meant to be added as a term, if it is true, it is discarded, per Domination law 2, and if it is false, it is a contradiction by Identity law 1. Any term which has already been processed is discarded by the Idempotent law 3. New literals are used to filter existing disjunctions by the distribution law of conjunction over disjunction 6.

Predicates are pandans to literals, the only exception being that they represent value ranges instead of referencing preceding layer outputs. LNNs use predicates relative to pre-specified elements of a vector or a dimension, and only ones related to the same feature can be compared. The conjunction of predicates is the cross-section of sets they define; if a cross-section cannot be found, they contradict.

As M-of-N rules are a shorter notation for disjunction of $\binom{N}{M}$ conjunctions, which are M terms long, the same simplifications apply, and M-of-Ns are filtered similarly. The advantage of the condensed representation of M-of-N rules allows for the filtering of terms to be applied directly; in other words, the list of N terms is filtered, and the number M is reduced accordingly. If the number of terms falls under the minimum required number M, it is a contradiction, while if the minimum required number reaches 0, the rule is equivalent to the constant value true. M-of-N rule can be expanded and simplified as such, but because it represents $\binom{N}{M}$ rules of length M, it pays off only if M is a small number like 1 or 2, or close to N, like $N-1$. Suppose any term in the form of a rule set or disjunction is generated. In that case, it is simplified so that the existing terms are checked against the new rule set, according to the logic laws, by distributing its elements and looking for identities and contradictions 7.

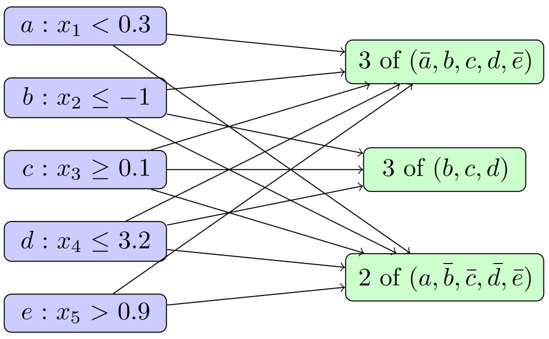
Appendix B

Model constants propagation

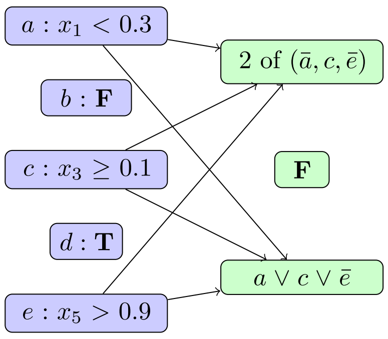
A scenario with training threshold values for quantisation, which can happen, is that the threshold value falls out of the value ranges for a feature, and the converted predicate effectively acts as a constant value. For example, in the MNIST problem, pixels have values in the range $x \in [0, 1]$ but a predicate $x < 2.3$ is equivalent to a constant true. The constant value can be propagated through the layers to simplify neuron expressions without losing generality. Improving inference time of a sample as useless predicates are removed and neuron logical expressions are simplified. It indicates that, the number of relevant pixels needed for the network to decide correctly, meaning, for MNIST, it does not need all 784 pixels to make an inference, but significantly less. As often is the case that the architecture of the network is bigger than is needed, some parts of it can become redundant and removing them can simplify the network and improve its performance. One simple example of the propagation of constants is shown in the following figure B.1.

In the figure B.1a is displayed an LNN which works with the first five features for the sake of simplicity of the example. The second and fourth neurons have predicates which have been trained with such thresholds that they can be replaced with a constant false and true, respectively. After propagating those constants, the M-of-N rules are simplified by distribution 6 and domination 2 laws.

Constant propagation has no effect if the problem's nature determines the feature ranges. If the training data determines the ranges, an issue might arise if testing data comes from a different distribution or with higher extreme values, where a complete network might still perform sufficiently,



(a) : Initial LNN



(b) : LNN state after propagation

Figure B.1: Propagation of constants in LNN

but a simplified one will not. For example using MNIST, the corner and edge pixels of the image are always dark (intensity always 0), even though their respective term's threshold falls within the $[0, 1]$ range. Such a predicate also acts as a constant on the training set and it could be eliminated, but with the possibility of affecting testing data accuracy.

The process is not essential for extracting rules and explanations but is useful from the performance and time complexity point of view, as it creates a simpler structure. These terms will, in any case, be removed in the post-extraction processing phase when sufficient rules are extracted (best terms selection) as their scores would be equal on both class and non-class data; this propagation removes truly unnecessary terms for training data. It can indicate feature importance if, after propagation, all terms referring to a feature are constants, then the network does not require it to make a decision. These predicates will be eliminated in the post-extraction processing phase when sufficient rules are extracted, as they are redundant.