VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Logical Neural Networks: Opening the black box

Daniel Thomas Braithwaite

Supervisor: Marcus Frean

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Artificial Neural Networks are growing in popularity and can generalise well to unseen examples. However it can be difficult to interpret the learned models. A problem which is driving the development of systems that are not only accurate but have a decision-making process that can be defended. This report develops a novel neural network architecture which builds in an interpretable structure. Experiments on the MNIST dataset showed these models had a statistically equivalent performance to Multi-Layer Perceptron Networks and provided evidence that they have a more interpretable learned model. The networks developed here were also shown to perform well when compared with recently developed methods aimed at solving the same problem.

Acknowledgements

I would particularly like to thank Marcus Frean; this report would not have been possible without his constant support, encouragement, and knowledge. I could not have asked for a more helpful supervisor.

I would also like to express my gratitude towards my parents, Anna Debnam and Eric Braithwaite, without their unwavering financial and spiritual support I would not be in a position to be at university or peruse research.

Thank you to my partner Scarlett Aroha O'Callaghan for supporting me through this and putting up with all the late nights and absent weekends.

Last but not least thank you to everyone I have gotten to know and work alongside in the Honors Labs, it has been a pleasure.

Contents

Chapter 1

Introduction

Artificial Neural Networks (ANN's) are commonly used to model supervised learning problems. A well trained ANN can generalize well, but it is difficult to interpret how the network is operating. This issue with interpretation makes ANNs like a black-box. This report aims to alleviate this problem by formalizing and developing a novel neural network architecture that builds interpretability into its structure.

1.1 Motivation

The number of situations in which ANN systems are used is growing, leading to an increasing interest in systems which are not only accurate but also provide a means to understand the logic used to derive their answers [1]. Interest in interpretable systems is driven by the variety of situations that utilize ANNs where incorrect or biased answers can have significant effects on users.

Ensuring a system is Safe and Ethical are two concepts which depending on the application are important to verify. If an ANN was able to provide its reasoning for giving a specific output then defending actions made by the system would be a more feasible task [1].

In the context of safety, a Machine Learning (ML) system often cannot be tested against all possible situations, as it is computationally infeasible to do so. If accessible, the logic contained inside the model could be used to verify that the system will not take any potentially dangerous actions.

It is also important to consider the implications of an ANN which is biased against a protected class of people. A paper published in 2017 demonstrated that standard machine learning algorithms trained using text data learn stereotyped biases [2]. An ANN designed with the intent to be fair could result in a system which discriminates because of implicit biases in the data used for training.

Another pressure causing the development of interpretable ML systems is changing laws. In 2018 the European Union (EU) General Data Protection Regulation [3] (GDPR) will come into effect. The GDPR will require algorithms which profile users based on their attributes be able to provide *"Meaningful information about the logic involved"*. This law would affect many situations where ML systems are used. For example, banks, which use ML systems to make loan application decisions [4]. Using some simulated data sets, researchers trained an ANN to compute the probability of loan repayment [4]. The simulated data consisted of white and non-white individuals, both groups with the same proportion of the population that made repayments. As the proportion of white to non-white individuals in the data increased, the ANN became less likely to grant loans to non-white individuals. This artificial situation demonstrates the effect biased data could have on an ANN.

The argument thus far has established that being able to defend or verify the decisions made by ANNs is not just an interesting academic question. It would allow for the creation of potentially safer and fairer ML systems. They provide a means to verify that not only correct decisions are made, but they are made for justifiable reasons. The GDPR gives a monetary motivation to use interpretable systems as breaches of the regulation will incur fines [4].

1.2 Solution

To address the problems laid out thus far we improved upon a probability based network architecture called Logical Neural Networks (LNNs) which yield a simpler trained model [5].

Existing methods for interpreting ANNs are post-hoc algorithms to extract some meaning from standard ANN's. The approach presented in this report builds interpretability into the ANN through a structure based off logical functions. This report introduces a formal foundation for LNNs through the following stages.

1. Motivate the concept that Logical Neural Networks are built from (Chapter 2).
2. Motivate and derive at a particular case of Logical Neural Networks called Logical Normal Form Networks (Chapter 3).
3. Discuss the performance, generalization and interpretation capabilities of Logical Normal Form Networks (Chapter 3).
4. Discuss the situations where Logical Normal Form Networks can be applied (Chapter 4)
5. Generalise Logical Normal Form Networks to Logical Neural Networks (Chapter 5).
6. Derive modifications to the Logical Neural Network architecture to improve accuracy (Chapter 5)
7. Evaluate the modified Logical Neural Network structure (Chapter 6)
8. Demonstrate the possible use cases of Logical Neural Networks (Chapter 7).

Following the development of Logical Neural Networks, they are evaluated on the MNIST database and compared against standard Multi-Layer Perceptron Networks. It will be demonstrated that LNNs are a reasonable alternative to standard MLPNs as they are simpler to interpret and do not sacrifice performance.

Chapter 2

Background

This chapter introduces concepts which are used to motivate and develop the solution presented.

2.1 Interpretability

To create a system that is interpretable it is necessary to have an understanding of what it means to interpret a model and how its interpretation might be evaluated. Interpretability of Machine Learning systems has been defined as *"the ability to explain or to present in understandable terms"* [1] which is ambiguous.

When is an interpretable model necessary? [1] In some problem domains there is less risk associated with incorrect answers. An ANN which controls a virtual car has a low risk as poor or dangerous driving will only affect an artificial environment. Problem domains where there is a high risk associated with incorrect answers include Safety and Ethics. A survey, "Concrete Problems in AI Safety" [6] provides an example of potential safety issues with AI in the context of a cleaning robot. For example, if the robot's objective is to have an environment containing no mess, how can the robot be prevented from disabling its sensors which it uses to detect the mess? Alternatively, a self-driving car might be penalized for running a stop light, but what is to prevent it from disabling the sensors that detect stop lights?

It has been demonstrated that Machine Learning systems learn human-like biases from textual data [2]. A study done in 2010 [7] developed a method to analyze datasets for prejudices against particular classes of people. An analysis of the *German Credit Dataset* was conducted. It which showed that discriminatory patterns were hidden in the data. An ANN trained on this data could learn these underlying biases. If the model was interpretable, then it could be examined for discriminatory patterns.

If an ANN model is presented as being interpretable how can this be verified scientifically? There are three categories of evaluation techniques [1].

1. *"Application-Grounded Evaluation"* Conducting experiments with human subjects in a specific problem domain. If the goal is to learn an interpretable classifier that grants bank loans, then a domain expert in granting/denying loans should be used to establish interpretability.

2. *"Human-Grounded Metrics"* Designing simpler experiments which still allow for es-

tablishing interpretability in a specific domain. This situation occurs when access to domain experts is either too expensive or difficult. The tasks can be simplified to allow humans that are not domain experts to complete them.

3. *"Functionally-Grounded Evaluation"* If it is possible to define interpretability of the problem then it can be used to establish this property in the model.

2.2 Rule Extraction

A survey in 1995 focuses on rule extraction algorithms [8], identifying the reasons for needing these algorithms along with introducing ways to classify and compare them.

There are three categories that rule extraction algorithms fall into [8]. An algorithm in the **decompositional** class focuses on extracting rules from each hidden/output unit. If an algorithm is in the **pedagogical** category, then rule extraction is thought of as a learning process. The ANN is treated as a black box, and the algorithm learns a relationship between the input and output vectors. The third category, **eclectic**, is a combination of decompositional and pedagogical. An eclectic, algorithm inspects the hidden/output neurons individually but extracts rules which represent the ANN globally [9].

To further divide the categories two more distinctions are introduced. One measures the portability of rule extraction techniques, i.e., how easily can they be applied to different types of ANN's? The second is criteria to assess the quality of extracted rules; these are accuracy, fidelity, consistency, and comprehensibility [8].

1. A rule set is **Accurate** if it can generalize, i.e., classify previously unseen examples.

2. The behaviour of a rule set with high **fidelity** is close to that of the ANN from which it was extracted.

3. A rule set is **consistent** if when trained under different conditions it generates rules which assign the same classifications to unseen examples.

4. The measure of **comprehensibility** is defined by the number of rules in the set and the number of literals per rule.

These surveys provide a framework for evaluating the rules extracted using a particular technique. By introducing this content, the reader is familiar with this approach to evaluating extracted rule sets. The solution developed in this report allows for rule exaction in some situations and will be assessed against these criteria.

2.3 LIME: Local Interpretable Model-Agnostic Explanations

The paper "'Why should I Trust You?' Explaining the Predictions of Any Classifier" [10] published in 2016 presents a novel approach to the interpretation of Machine Learning models. The motivation for this research is the idea of trusting the answers provided, either in the context of an individual prediction or the model as a whole. The LIME technique explains a single prediction. Trust in the model can be developed by inspecting explanations of many individual predictions.

Essentially the LIME algorithm treats the classifier as a black-box and generates many examples by slightly perturbing the instance for which an explanation is wanted and asks the model for a classification. It then constructs a linear model of local region around the instance using the classifications of the perturbed examples. A more detailed explanation of LIME is beyond the scope of this report.

LIME is a recent method for interpreting machine learning models. Consequently, it will provide a comparison point when evaluating the solution presented in this report.

2.4 Noisy Neurons

In 2016 the concept of Noisy-OR and Noisy-AND neurons [5] was presented. These neurons are based on the discrete boolean OR and AND gates. The motivation for this work was that logical functions are directly interpretable as functions of their input. Consequently, if a problem had a natural, logical decomposition, then it could be learned by Artificial Neural Networks using Noisy neurons.

The Noisy-OR neuron is derived from the Noisy-OR relation [11], a concept in Bayesian Networks developed by Judea Pearl. A Bayesian Network represents the conditional dependencies between random variables in the form of a directed acyclic graph [12]. Figure 2.1 is a Bayesian network. It demonstrates the dependency between random variables "Rush Hour", "Raining", "Traffic", "Late To Work". The connections show dependencies, i.e., Traffic influences whether someone is late to work, and it being rush hour or raining influences whether there is traffic.

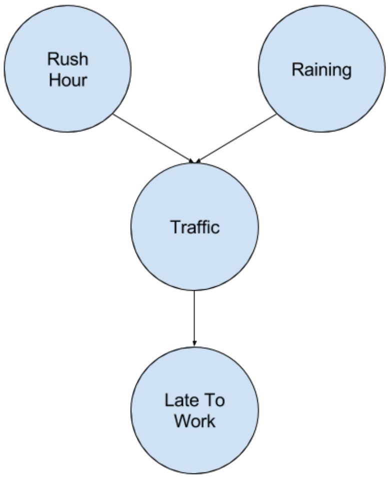


Figure 2.1

Consider a Bayesian Network consisting of a node D with parents $S_1, ..., S_n$. In other words, S_i influences the node D. Each S_i is independent of all others. The relationship between D and its parents is defined as if S_1 *OR* ... *OR* S_n is true then D is true.

This relationship is binary, but there might be some uncertainty as to whether each S_i influences D. Let ϵ_i be the probability that if $S_i = 1$ then $D = 0$, i.e. $P(D = 0|S_i = 1) = \epsilon_i$. Then $P(D = 0|S_i = 1 \; \forall i) = P(D = 0|S_1 = 1) \cdots P(D = 0|S_n = 1) = \prod_{i=1}^{n} \epsilon_i$. Therefore $P(D = 1|S_1 = 1, , S_n = 1)$ can be expressed as Equation 2.1.

$$P(D = 1|S_1 = 1, ..., S_n = 1) = 1 - \prod_{i=1}^{n} \epsilon_i \tag{2.1}$$

In the context of a neuron consider the inputs $x_1, ..., x_n$ to represent D's belief of the probability that the inputs $1, ..., n$ are True. The output of a neuron as conditionally dependent on its inputs, for a Bayesian Network each x_i is a parent of the neuron. Each ϵ_i is D's uncertainty as to whether x_i influences its output. The probability of D being True is influenced by x_i and ϵ_i. A function f is needed to compute the irrelevance of node i in light of x_i and ϵ_i. The list in Figure 2.2 gives three properties of $f(x_i, \epsilon_i)$ [5].

1. $\epsilon = 1$ means that $f(\epsilon, x) = 1$
2. $x = 0$ means that $f(\epsilon, x) = 1$
3. Monotonically increasing in ϵ and decreasing in x. In other words, fixing x results in a function that monotonically increases as ϵ get closer to 1. Fixing ϵ results in a function that monotonically decreases as x approaches 1.

Figure 2.2: Three properties of the function f

If the inputs of f are restricted to be either 0 or 1 then it becomes apparent that properties (1) - (3) describe the logic function $x \implies \epsilon$ (x implies ϵ) , this is shown in Figure 2.3.

1. *f(1,1) = 1:* This is shown by (1)
2. *f(1,0) = 1:* Shown by (2)
3. *f(0,1) = 0:* Fix $x = 1$ and let $\epsilon = 1, f(1,1) = 1$. As $\epsilon \to 0$ f is monotonically decreasing so $f(0,1) = 0$.
4. *f(0,0) = 1:* Shown by (2)

Figure 2.3: Demonstration that f is the boolean function $x \implies \epsilon$ when $x, \epsilon \in \{0,1\}$

A function $g(x_1, ..., x_m)$ is defined to be *Boolean-Like* [13] when $x_i \in \{0,1\} \forall i$ then $g(x_1, ..., x_m) \in \{0,1\}$. Consequently, on top of properties (1) - (3) f should be a *boolean like* Implies function. A choice of f which satisfies properties (1) - (3) is $f(x, \epsilon) = \epsilon^x$. It is a simple task to check that f is a *boolean like* Implies function. Figure 2.4 shows that all properties are satisfied.

$$\begin{aligned}
f(1,1) &= 1^1 = 1 \\
f(1,0) &= 1^0 = 1 \\
f(0,1) &= 0^1 = 0 \\
f(0,0) &= 0^0 = 1
\end{aligned}$$

Figure 2.4: Checking $f(\epsilon, x) = \epsilon^x$ is a Boolean Like Implies function. Note that for $f(0,0) = 0^0$ strictly this is undefined, but $\lim_{\epsilon \to 0} \epsilon^0 = 0$

This is not a unique solution but it has a property which becomes convenient later on, specifically $\log(f) = \log(\epsilon^x) = x \cdot \log(\epsilon)$.

Definition 2.4.1. A **Noisy-OR** Neuron has weights $\epsilon_1, ..., \epsilon_n \in (0,1]$ which represent the uncertainty that corresponding inputs $x_1, ..., x_n \in [0,1]$ influence the output. The activation of a Noisy-OR Neurons is given in Equation 2.2.

$$a = 1 - \prod_{i=1}^{p} (\epsilon_i^{x_i}) \tag{2.2}$$

Consider that the Noisy-OR activation just defined is a *boolean like* OR function. How can Equation 2.2 be transformed to give a Noisy-AND activation which is *boolean like*. From logic it is known that $a \wedge b = \neg(\neg a \vee \neg b)$. Considering that $f_{NOT}(a) = 1 - a$ is a *boolean like* NOT function then $f_{AND}(x_1, ..., x_n) = f_{NOT}(f_{OR}(f_{NOT}(x_1), ..., f_{NOT}(x_1)))$. Using this as intuition the Noisy-OR activation can be converted into a Noisy-AND.

Definition 2.4.2. A **Noisy-AND** Neuron has weights $\epsilon_1, ..., \epsilon_n \in (0, 1]$ which represent the uncertainty that corresponding inputs $x_1, ..., x_n \in [0, 1]$ influence the output. The activation of a Noisy-AND Neurons is given in Equation 2.3

$$a = \prod_{i=1}^{p} (\epsilon_i^{1-x_i}) \tag{2.3}$$

The noisy neurons are the building blocks for the two network architectures present in this report. Without an understanding of these concepts, it would be difficult to understand the motivations for the solution developed.

2.5 Logical Normal Form Networks

2.5.1 CNF & DNF

A boolean formula is in Conjunctive Normal Form (CNF) if and only if it is a conjunction (and) of clauses. A clause in a CNF formula is given by a disjunction (or) of literals. A literal is either an atom or the negation of an atom, and an atom is one of the variables in the formula.

Consider the boolean formula $\neg a \vee (b \wedge c)$, then the CNF is $(\neg a \vee b) \wedge (\neg a \vee c)$. In this CNF formula the clauses are $(\neg a \vee b)$, $(\neg a \vee c)$, the literals used are $\neg a$, b, c and the atoms are a, b, c.

A boolean formula is in Disjunctive Normal Form (DNF) if and only if it is a disjunction (or) of clauses. A DNF clause is a conjunction (and) of literals. Literals and atoms are defined the same as in CNF formulas.

Consider the boolean formula $\neg a \wedge (b \vee c)$, then the DNF is $(\neg a \wedge b) \vee (\neg a \wedge c)$.

2.5.2 CNF & DNF from Truth Table

Given a truth table representing a boolean formula, constructing a DNF formula involves taking all rows which correspond to True and combining them with an OR operation. To construct a CNF one combines the negation of any row which corresponds to False by an OR operation and negates it.

Theorem 2.5.1. The maximum number of clauses in a CNF or DNF formula is 2^n

Proof. Assume the goal is to find the CNF and DNF for a Boolean formula B of size n, for which the complete truth table is given. The truth table has precisely 2^n rows.

First, assume a CNF is being constructed, this is achieved by taking the OR of the negation of all rows corresponding to False, the NOT operation leaves the number of clauses unchanged. At most, there can be 2^n rows corresponding to False. consequently, there is no more than 2^n clauses in the CNF.

A similar argument shows that the same holds for DNF. □

2.5.3 Definition of Logical Normal Form Networks

In 1996 a class of networks called Logical Normal Form Networks [14] (LNFNs) where developed. Focusing on learning the underlying CNF or DNF for a boolean expression which describes the problem. The approach relies on a specific network configuration along with restriction the function space of each neuron, allowing them to only perform an OR or AND on a subset of their inputs. Such OR and AND neurons are called Disjunctive and Conjunctive retrospectively. If the trained network can achieve a low enough error rate, then rules can be extracted from the network as a Boolean CNF or DNF expression [14].

The algorithm which extracts rules from LNFNs would be Electic and indeed is not Portable as the algorithm is specific to the LNFN architecture. It is not possible to further classify the rule extraction algorithm as the research developing it lacks any experimental results. Justification is also missing making the LNFNs challenging to reproduce.

2.6 Logical Neural Networks

ANN's containing of Noisy-OR and Noisy-AND neurons are called Logical Neural Networks [5] (LNN's). If the network consists of only Noisy neurons then it a pure LNN. ANNs containing a mix of logical and standard activations where shown not to yield interpretable models and also have lower performance, consequently when LNNs are referred to it will always be in the context of Pure LNNs.

2.7 Learning Fuzzy Logic Operations in Deep Neural Networks

A paper published in September 2017 [15] has a novel approach to combine Deep Learning and Fuzzy Logic with the goal of creating artificial neural networks which can be trained using backpropagation and be more interpretable. They propose a single activation which can learn to perform many different fuzzy logic operations. The result of this paper is a neural network architecture that can perform well as a classifier but also yield a model from which fuzzy rules can be extracted. The Fuzzy Logic networks where compared against standard deep neural networks using a tanh activation function. In this comparison, they were found to have equivalent performance in some cases but worse in others. After training it was possible to extract fuzzy rules from the networks which were unable to perform well but had a simple representation. Fuzzy Logic Networks are a different approach to solving the problem outlined in this report. Consequently, they will be an interesting comparison point for the solution developed here. Does the solution presented here have comparable performance and interpretability?

Chapter 3

Foundation of Logical Normal Form Networks

This chapter motivates the concept of Logical Normal Form Networks (LNFNs) along with deriving them in terms of Noisy Neurons. The training of LNFNs is discussed by deriving a weight initialization method, choosing an algorithm to propagate gradients and justifying a choice of mini-batch size. LNFNs are also compared against Multi-Layer Perceptron Networks (MLPNs) using randomly generated truth tables as the data sets.

Consider the set of binary classification problems which have boolean input features. p is some problem in this set, with X_p and Y_p being the examples and targets retrospectively. Let B_p be the set of all boolean functions which take an $x \in X_p$ and take it to either a 1 or 0. Then finding the optimal boolean function to solve the problem p corresponds to expression 3.1 which is the function f with the smallest cross entropy loss.

$$\operatorname*{arg\,min}_{f \in B_p} \quad - \sum_{0 \leq i \leq |X_p|} (Y_{p_i} \cdot \log f(X_{p_i})) + ((1 - Y_{p_i}) \cdot \log(1 - f(X_{p_i}))) \tag{3.1}$$

How might Equation 3.1 be optimised in a way which allows f to be known? Enumerating every possible $f \in B_p$ is to computationally expensive. Alternatively, a Multi-Layer Perceptron Network (MLPN) could learn f using gradient descent. MLPNs are universal approximators so there exists a network architecture that can learn the optimal f. The problem with MLPNs is interpreting the learned function after training is difficult. The remainder of this chapter is dedicated to deriving an Artificial Neural Network Architecture which can be trained with gradient descent and reveal the function learnt after training.

Every boolean function has a unique Conjunctive and Disjunctive Normal Form (CNF & DNF). Learning the CNF of DNF of f is functionally equivalent to learning f and provides more guarantees about the structure of the solution. The three logical operations, OR, AND and NOT, describe both the CNF and DNF. A NOT operation can only occur inside a clause as a literal. Finally, the maximum number of clauses in a CNF or DNF is 2^n where n is the number of variables. A Bayesian Network representing the CNF of f could be thought of as three layers of nodes. The first layer containing all the literals (atoms and their negations), two of these nodes are dependent if they concern the same atom (one is the negation of the other). The second layer with $k \leq 2^n$ nodes representing the clauses, each is dependent on a subset of the atoms and is True if all the dependencies are True. The last layer contains a single node which is dependent on all the second layer nodes and is True if one of the clauses is True. Each clause is an OR and the last layer node is an AND. This Bayesian

Network could then be represented by a Feed-Forward Neural Network where the nodes in the first, second and third layers are input, hidden and output neurons.

Each hidden neuron is a Noisy-OR, and the output neuron is a Noisy-AND. Figure 3.1 is the structure that has been derived. By the same logic, a network architecture for learning the DNF can be derived, where the hidden layer consists of ANDs and the output of a single OR. These networks for learning CNF (Definition 3.0.1) and DNF (Definition 3.0.2) formulae are a new derivation of Logical Normal Form Networks (LNFNs) [14] which use Noisy Neurons.

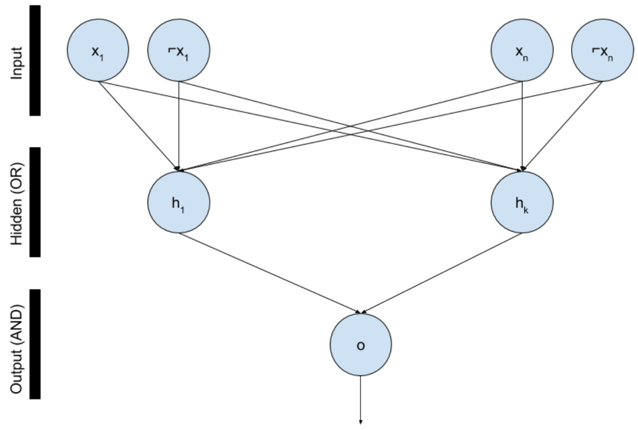


Figure 3.1: Network Architecture for Learning CNF

Definition 3.0.1. A **CNF-Network** is a three layer network where neurons in the hidden layer consist solely of Noisy-OR's, and the output layer is a single Noisy-AND.

Definition 3.0.2. A **DNF-Network** is a three layer network where neurons in the hidden layer consist solely of Noisy-AND's, and the output layer is a single Noisy-OR.

Definition 3.0.3. An **LNF-Network** is a DNF or CNF Network

Theorem 3.0.1 proves that a CNF or DNF Network should have a hidden layer size of 2^n to guarantee that the formulae can be learned.

Theorem 3.0.1. Let T be the complete truth table for the boolean formula B. Let L be an LNFN, if L has 2^n hidden units then there always exists a set of weights for L which correctly classifies any assignment of truth values to atoms.

A proof of Theorem 3.0.1 is given in Appendix A. Theorem 3.0.1 justifies using 2^n hidden units, and it guarantees that there exists an assignment of weights yielding a network that can correctly classify each item in the truth table.

3.1 Noisy Gate Parametrisation

The parametrization of Noisy gates requires weight clipping, an expensive operation. A new parametrization is derived that implicitly clips the weights. Consider that $\epsilon \in (0, 1]$, therefore let $\epsilon_i = \sigma(w_i)$, these w_i's can be trained without any clipping, after training the original ϵ_i's can be recovered. These weights must be substituted into the Noisy activations. First, consider the Noisy-OR activation.

$$
\begin{aligned}
a_{or}(X) &= 1 - \prod_{i=1}^{p}(\epsilon_i^{x_i}) \\
&= 1 - \prod_{i=1}^{p}(\sigma(w_i)^{x_i}) \\
&= 1 - \prod_{i=1}^{p}((\frac{1}{1+e^{-w_i}})^{x_i}) \\
&= 1 - \prod_{i=1}^{p}((1+e^{-w_i})^{-x_i}) \\
&= 1 - e^{\sum_{i=1}^{p} -x_i \cdot ln(1+e^{-w_i})} \\
&Let\ w_i^{'} = ln(1+e^{-w_i}) \\
&= 1 - e^{-(W^{'} \cdot X)}
\end{aligned}
$$

From a similar derivation, a re-parametrized Noisy-AND activation can be found.

$$
\begin{aligned}
a_{and}(X) &= \prod_{p}^{i=1}(\epsilon_i^{1-x_i}) \\
&= \prod_{p}^{i=1}(\sigma(w_i)^{1-x_i}) \\
&= e^{\sum_{i=1}^{p} -(1-x_i) \cdot ln(1+e^{-w_i})} \\
&= e^{-(W^{'} \cdot (1-X))}
\end{aligned}
$$

Concisely giving equations 3.2, 3.3

$$a_{and}(X) = e^{-(W^{'} \cdot (1-X))} \tag{3.2}$$

$$a_{or}(X) = 1 - e^{-(W^{'} \cdot X)} \tag{3.3}$$

The function taking w_i to $w_i^{'}$ is the soft ReLU function which is performing a soft clipping on the w_i's.

3.2 Training LNF Networks

Using equations 3.3 and 3.2 for the Noisy-OR, Noisy-AND activations retrospectively allows LNFNs to be trained without the need to clip the weights. Before LNFNs can be trained some choices must be made. These are outlined in the list below.

1. *Weight Initialization:* A method for initializing the weights must be derived. In the case of MLPNs it was shown that if initialization of weights is not carefully thought about as the number of neurons increases the learning conditions deteriorate [16].
2. *Training Algorithm:* There exist many algorithms for propagating gradients [17]. One must be chosen based on the properties of these networks.
3. *Batch Size:* What size should each batch be when training these networks.

3.2.1 Weight Initialization

Poor weight initialization leads to slow or poor convergence [18]. Weight initialization algorithms can not be universally applied to different activations. A method derived for MLPNs was shown to be ineffective for networks using other activations [19].

Deriving a Distribution For The Weights Before a weight initialization algorithm can be derived good learning conditions must be identified. The function e^{-z} squash a variable z into the range $[0, 1]$, Figure 3.2 shows a graph of this function.

At $z \geq 4$ the function e^{-z} changes very little as it gets asymptotically close to 0, this means that e^{-z} suffers from what is known as saturation. If the method of initializing weights does not consider the inputs to a neuron then at a certain point the weighted sum of inputs will always be ≥ 4 and thus the activation will always be constant. The sigmoid function suffered from this problem and one method of fixing it was careful initilizaton of the weights [16]. The approach here is to derive an initial weight distribution which keeps the variance and mean of neuron activations the same throughout the network [20].

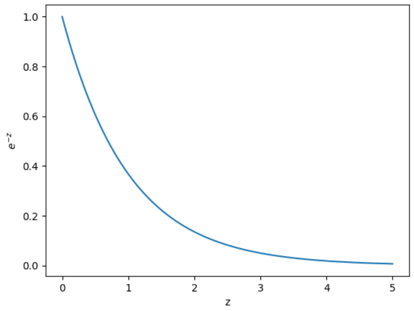


Figure 3.2: Plot of e^{-z}

As with previous approaches to solving this problem, it will be assumed the inputs to the network are independent, and weights are independent and identically distributed (i.i.d). It has previously been proved that these assumptions are enough to show the inputs to any layer are independent [20].

Ideally, the output of each neuron would be varied for each training example, i.e., $y \sim U(0, 1)$. Each training example $X = \{x_1, ..., x_n\}$ has each component $x_i \in (0, 1]$, it will be assumed that $x_i \sim U(0, 1)$.

If the input vectors and output of each neuron are both distributed $U(0, 1)$ then the input to any layer is distributed $U(0, 1)$. Based on these facts it can be argued that the weight initialization is the same for both Noisy-OR and Noisy-AND, first recall the activations for Noisy neurons given in Figure 3.3.

$$a_{AND}(X) = e^{-(w_1(1-x_1)+...+w_n(1-x_n))}$$
$$a_{OR}(X) = 1 - e^{-(w_1x_1+...+w_nx_n)}$$

Figure 3.3: Noisy Neuron Activations

Consider a random variable g, if $g \sim U(0, 1)$ then $1 - g \sim U(0, 1)$ also holds. Consequently, if $x_i \sim U(0, 1)$ then $1 - x_i$ $U(0, 1)$, also $e^{-z} \sim U(0, 1)$ then $1 - e^{-z}$ $U(0, 1)$. It is, therefore, enough to consider $e^{-(w_1x_1+...+w_nx_n)}$ when deriving the initialization method for each w_i. Each $w_i = \log(1 + e^{w_i'})$ (as derived in Section 3.1) however for this initialisation derivation this will be forgotten for now. Given that $y = e^{-z}$ $U(0, 1)$, a first step is to determine the distribution of z.

Theorem 3.2.1. If $y \sim U(0, 1)$ and $y = e^{-z}$, then $z \sim exp(\lambda = 1)$

Proof. Consider that $y = e^{-z}$ can be rewritten as $z = -\log(y)$. The Cumulative Distribution Function (CDF) will be derived for z.

$$
\begin{aligned}
F(z) &= P(Z < z) \\
&= P(-\log(Y) < z) \\
&= P(\frac{1}{Y} < e^{-z}) \\
&= P(Y \geq e^{-z}) \\
&= 1 - P(Y \leq e^{-z}) \\
&= 1 - \int_0^{e^{-z}} f(y)dy \\
&= 1 - \int_0^{e^{-z}} 1dy \\
&= 1 - e^{-z}
\end{aligned}
$$

Therefore $F(z) = 1 - e^{-\lambda z}$ where $\lambda = 1$. So z has the CDF of an exponential distribution. Consequently $z \sim exp(\lambda = 1)$ □

The problem has now been reduced to find how w_i is distributed given that $z \sim exp(\lambda = 1)$ and $x_i \sim U(0, 1)$. The approach taken is to find $E[w_i]$ and $var(w_i)$.

$$
\begin{aligned}
E[z] &= E[w_1 x_1 + \cdots + w_n x_n] \\
&= E[w_1 x_1] + \cdots + E[w_n x_n] \ (independence) \\
&= E[w_1]E[x_1] + \cdots + E[w_n]E[x_n] \\
&= n \cdot E[w_i]E[x_i] \ (i.i.d) \\
&= n \cdot E[w_i] \cdot \frac{1}{2} \\
1 &= \frac{n}{2} \cdot E[w_i] \\
E[w_i] &= \frac{2}{n}
\end{aligned}
$$

$$
\begin{aligned}
var(z) &= var(w_1 x_1 + \cdots + w_n x_n) \\
&= var(w_1 x_1) + \cdots + car(w_n x_n)
\end{aligned}
$$

$$
\begin{aligned}
var(w_i x_i) &= (E[w_i])^2 var(x_i) + (E[x_i])^2 var(w_i) + var(w_i) var(x_i) \\
&= \frac{4}{n^2} \cdot \frac{1}{2} + \frac{1}{4} \cdot var(w_i) + var(w_i) \cdot \frac{1}{12} \\
&= \frac{1}{3n^2} + \frac{1}{3} var(w_i)
\end{aligned}
$$

Consequently, the variance can be found by the following

$$\begin{aligned}
1 &= n \cdot [\frac{1}{3n^2} + \frac{1}{3} var(w_i)] \\
3 &= \frac{1}{n} + n \cdot var(w_i) \\
3n &= n^2 var(w_i) \\
var(w_i) &= \frac{3}{n}
\end{aligned}$$

From the above arguments $E[w_i] = \frac{2}{n}$ and $var(w_i) = \frac{3}{n}$. These values need to be fitted to a distribution from which weights can be sampled. Based on our initial assumptions this distribution must also generate values in the interval $[0, \infty]$. There exist many different distributions which satisfy this constraint. The distribution which had the best results was log-normal [21], for brevity only its derivation is included.

Fitting To Log Normal A Log-Normal distribution has two parameters μ and σ^2, by the definition of Log-Normal $E[w_i] = \frac{2}{n} = e^{\mu + \frac{\sigma^2}{2}}$ and $var(w_i) = \frac{3}{n} = \left[e^{\sigma^2} - 1\right] \cdot e^{2\mu + \sigma^2}$. Consequently

$$\begin{aligned}
\frac{2}{n} &= e^{\mu + \frac{\sigma^2}{2}} \\
\log(\frac{2}{n}) &= \mu + \frac{\sigma^2}{2} \\
\log(\frac{4}{n^2}) &= 2\mu + \sigma^2
\end{aligned}$$

From here this can be substituted into the other formula to give

$$\begin{aligned}
\frac{3}{n} &= \left[e^{\sigma^2} - 1\right] \cdot e^{2\mu + \sigma^2} \\
&= \left[e^{\sigma^2} - 1\right] \cdot e^{\log(\frac{4}{n^2})} \\
&= \left[e^{\sigma^2} - 1\right] \cdot \frac{4}{n^2} \\
3n &= 4 \cdot e^{\sigma^2} - 4 \\
\frac{3n+4}{4} &= e^{\sigma^2} \\
\sigma^2 &= \log \frac{3n+4}{4}
\end{aligned}$$

Finally, this substituted back gives the mean

$$\begin{aligned}
\log(\frac{4}{n^2}) &= 2\mu + \log \frac{3n+4}{4} \\
2\mu &= \log(\frac{4}{n^2}) - \log \frac{3n+4}{4} \\
\mu &= \frac{1}{2} \cdot \log \frac{16}{n^2(3n+4)}
\end{aligned}$$

Giving the parameters for the log normal distribution below

$$\sigma^2 = \log \frac{3n+4}{4} \tag{3.4}$$

$$\mu = \frac{1}{2} \cdot \log \frac{16}{n^2(3n+4)} \tag{3.5}$$

Weight Initialization Algorithm Figure 3.4 gives the algorithm for initializing the weights of Noisy neurons. Recall $w = f(w^{'})$ where $w^{'}$ can be any real value, consequently to obtain the initial weights each w must be inversely transformed using f^{-1}.

```
function constructWeights(size):
$w_i \sim LN$ (for i = 0 to size)
return $f^{-1}(\{w_0, ...\})$
```

Figure 3.4: Weight Initialization Algorithm for LNNs

3.2.2 Training Algorithm

The ADAM Optimizer [22] is the learning algorithm which will be used. For the convenience of an adaptive learning rate and because it has been shown that networks of this form yield a sparse representation, which the ADAM algorithm works well with.

3.2.3 Batch Size

The batch size will be fixed to one. Preliminary results showed that training an LNFN with a batch size of 1 converged faster.

3.3 LNF Network Performance

Preliminary testing initially gave poor performance on the simplest of logic gates. After implementing the weight initialization algorithm (derived in Section 3.2.1) the results became promising. They showed that LNFN's could learn good classifiers on boolean gates, i.e., NOT, AND, NOR, NAND, XOR, and Implies. How do LNFNs perform against standard MLPNs which are known to be universal function approximators? Two different MLPNs will be used as a benchmark

1. One will have the same configuration as the LNFNs, i.e., 2^n hidden neurons.

2. The other has two hidden layers, both with N neurons.

The testing will consist of selecting 5 random boolean expressions for $2 \leq n \leq 9$ and training each network 5 times, each with random initial conditions. Figure 3.5 shows a comparison between all 4 of the networks and Figure 3.6 shows just the LNFN's.

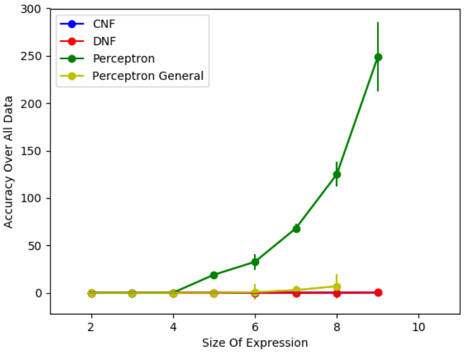


Figure 3.5

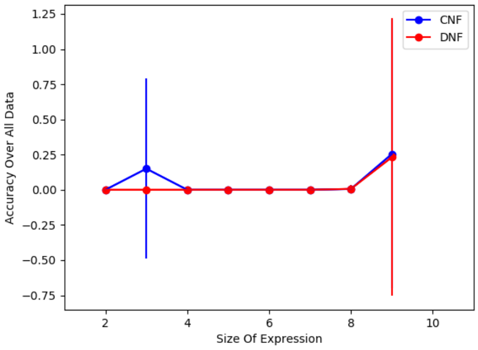


Figure 3.6

Figure 3.5 shows that the LNFNs have statistically equivalent performance to one of the configurations of MLPNs and the other Perceptron network with 2^n hidden neurons has poor performance. Figure 3.6 shows that CNF and DNF networks have statistically equivalent performance.

3.4 LNF Network Generalization

These networks can perform as well as standard perceptron networks, but so far they have been trained on a complete data set. In practice, this will almost never be the case. Standard ANN's are widely used because of their ability to generalize. Here the generalization capability of LNFNs will be tested against that of MLPNs

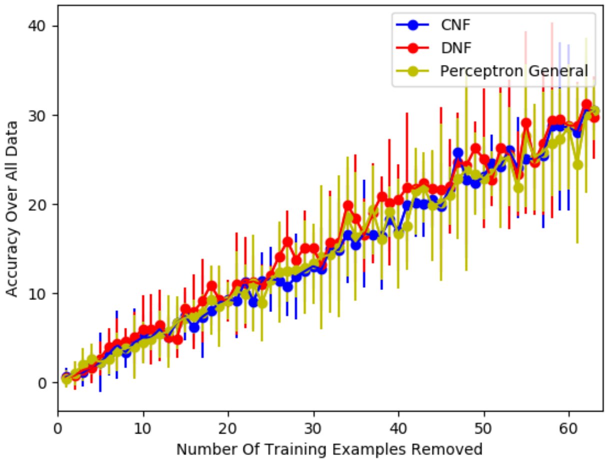


Figure 3.7

Figure 3.7 shows a comparison between the generalization ability of CNF, DNF and Perceptron networks. The graph shows the performance over all training data when successively removing elements from the training set. It demonstrates that the CNF and DNF networks generalize as well as the perceptron networks when trained over boolean problems with 6 inputs, this trend continues as n increases up to 9. Training LNFNs on boolean problems with more than 9 inputs is to expensive.

3.5 LNF Network Rule Extraction

Consider the weights for a logical neuron $W = \{w_1, ..., w_n\}$. These can be converted to $\epsilon_i = \sigma(w_i)$ where $\epsilon_i \in [0, 1]$ and represents the relevance input x_i has on the neuron's output.

To extract boolean rules from the network it must be possible to interpret each neuron as a logical function acting on a subset of its inputs. For this to be the case, at the conclusion of training either $\epsilon_i \approxeq 1$ or $\epsilon_i \approxeq 0$ needs to be true. If each ϵ_i is either 1 or 0 then the neuron is a logical function of the inputs which have corresponding $\epsilon \approxeq 0$. Conjecture 3.5.1 was derived from experimental evidence by training LNFNs over complete truth tables and inspecting the weights, and it is the foundation of the algorithm given in Figure 3.8. Ideally, Conjecture 3.5.1 would be proved, but that is out of scope for this report.

Conjecture 3.5.1. For an LNFN network trained on a binary classification problem with boolean inputs as the loss approaches 0 (i.e., the correct CNF or DNF has been found) the weights $\{w_1, ..., w_n\}$ approach ∞ or $-\infty$. Consequently each $\epsilon_i = \sigma(w_i)$ (where $\sigma(\cdot)$ is the sigmoid function) approaches 0 or 1.

The Algorithm displayed in figure 3.8 extracts rules from CNFNs. It takes the output weights (ow) and hidden weights (hw) as input and outputs a boolean expression. A similar algorithm can be derived for DNFNs, it is omitted but can be obtained by simply switching the logical operations around. In practice, many clauses in the extracted expression contain redundant terms, such as clauses that are a tautology or a duplicate of another.

```
atoms = $\{x_1, ...x_n, \neg x_1, ..., \neg x_n, \}$

function extractRulesCNFN(ow, hw)
  ow = $\sigma$(ow)
  hw = $\sigma$(hw)
  relvHidden = [hw[i] where ow[i] := 0]

  and=And([])
    for weights in relvHidden
      or=Or([atoms[i] where weights[i]:=0])
      and.add(or)

  return and
```

Figure 3.8: Rule Extraction Algorithm (for CNFN)

How does training over incomplete truth tables affect the generalization of extracted rules and what factors could influence this?

Consider B to be the set of all boolean problems with n inputs. What is the cardinality of B? There are 2^n rows in the truth table and 2^{2^n} ways to assign true/false values to these rows, each way corresponding to a different boolean function. Consequently $|B| = 2^{2^n}$. Consider some $b \in B$ represented by 2^n rows of a truth table, removing one row from the training data means there are now two possible functions that could be learned, one where the removed row corresponds to true and the other to false. As more rows are removed, this problem is compounded. If m rows are taken then there are 2^m possible functions that b could represent.

To test the generalization of the CNFN and DNFN rule sets LNFNs will be trained over a randomly generated boolean problem with 4 inputs. The training set will reduce in size one by one, and the extracted rules will be evaluated over the entire tuth table. For each training set size, the experiment will be repeated 30 times. Figures 6.40 & 3.10 shows the number of examples removed from the training set on the x-axis. The y-axis shows the number of incorrectly classified instances over the entire dataset. These experiments show that when no examples are removed, then the rules can achieve a perfect accuracy. As instances are removed from the training set the rules can generalize on average.

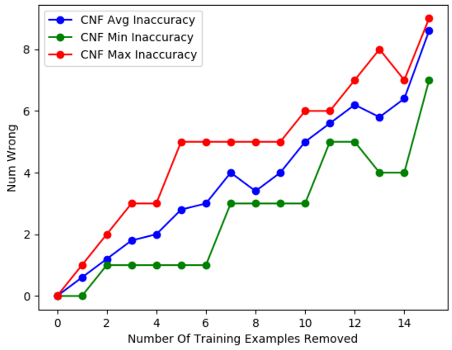


Figure 3.9: CNFN trained using an incomplete truth table

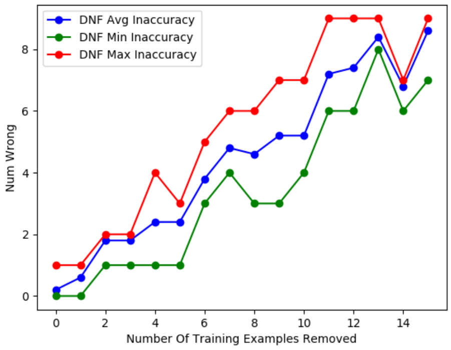


Figure 3.10: DNFN trained using an incomplete truth table

When constructing a CNF from a truth table (See Section 2.5.2) only the rows corresponding to false are considered. Is it then possible that by only removing rows which correspond to false a CNFN can learn rules which achieve a perfect accuracy? Figure 3.11 represents this situation.

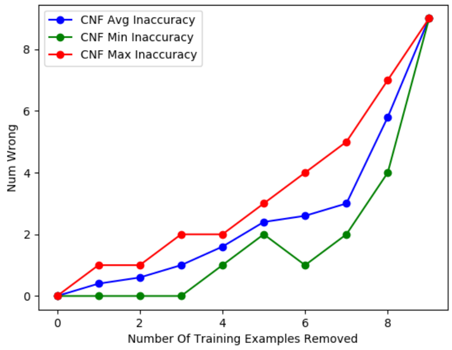


Figure 3.11: CNFN trained using an incomplete truth table, only rows corresponding to false are removed

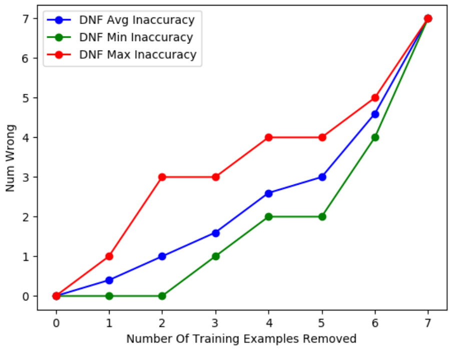


Figure 3.12: DNFN trained using an incomplete truth table, only rows corresponding to true are removed

Figures 3.11 & 3.12 show CNFNs and DNFNs trained over partial data retrospectively. In the case of CNFNs only entries corresponding to true in the truth table are removed and for the DNFNs only false entries. These graphs show that careful removal of each training example does not result in rules that perform better. These results indicate that the generalization of extracted rule sets depends on the initial conditions rather than what examples are removed.

3.6 Summary

In this chapter, a method for training LNFNs was derived. LNFNs where shown to have statistically equivalent performance and generalization to MLPNs. Finally, it was shown that rules can be extracted from LNFNs and that these rule sets were able to generalise.

Chapter 4

Expanding the Problem Domain of Logical Normal Form Networks

The LNFNs presented in Chapter 3 have only been shown to be applicable when learning boolean truth tables. This chapter investigates extending LNFNs to support multi-class classification problems and continuous feature spaces.

4.1 Multi-class Classification

Extending an ANN to support multi-class classification is achieved by adding more output neurons, one for each class. The value output neuron i can be considered the probability of class i. The network's output is now a vector representing the class distribution.

For example if given a problem with 3 classes then $\{1, 0, 0\}$, $\{0, 1, 0\}$ and $\{0, 0, 1\}$ represent class 1, 2 and 3 retrospectively. The LNFN would have 3 output neurons, each representing a bit in the output vector. During the training process if the true class of an example was 1 then the desired output vector would be $\{1, 0, 0\}$. This process of converting a categorical variable to a binary string is known as **One-Hot Encoding**

Definition 4.1.1. An LNFN to solve a k class classification problem is unchanged apart from the number of output neurons which is k.

4.1.1 Application to Lenses Problem

The Lenses problem [23] is multi-class and contains a natural rule system making it an ideal problem for evaluating LNFNs. The goal is to determine whether (if any) contact lenses should be fitted to the patient. There are 3 classes, fit soft, fit hard and do not fit any. Each example has four features; three are binary and one categorical (of size 3). Applying One-Hot encoding to the categorical variable yields a set of instances each with length 6.

The LNFN will be evaluated against an MLPN with a two-layer structure where the layer widths are $2 \cdot n$ and n retrospectively. The performance of the three classifiers will be compared using Leave-One-Out Cross-Validation.

	Error Rate	Error Rate CI (95%)
CNF Net	0.0122	(0.0000, 0.0417)
DNF Net	0.0104	(0.0000, 0.0417)
MLPN Net	0.0000	(0.0000, 0.0000)

Table 4.1: Network Performance On Lenses Problem

Table 4.1 demonstrates that the CNF & DNF Networks perform comparably to an MLPN as the confidence intervals for the errors overlap. Now that the LNFN network has three output neurons it should be possible to extract three rules describing each of the classes. Consider that each problem instance is of the following form $\{a, b, c, d, e, f\}$ where a, b, c, d, e, f are all atoms.

f refers to the *tear production rate* being normal or reduced if $f = \mathit{False}$ or True retrospectively. Describing the other atoms is not beneficial as they refer to medical terms which are unimportant.

The list of rules in Figure 4.1 have been extracted from a CNFN after being trained over the complete Lenses data set. These rules contain redundancies which could be filtered out automatically.

- Class 1: $(a \vee \neg d) \wedge (e \vee \neg f) \wedge (\neg(\neg f)) \wedge (\neg(\neg e \vee \neg f)) \wedge (\neg(e \vee \neg f))$
- Class 2: $(a \vee b \vee d \vee e \vee \neg c) \wedge (\neg(e \vee \neg f)) \wedge (\neg(e \vee \neg f)) \wedge (\neg(e \vee \neg f))$
- Class 3: $(c \vee e \vee \neg b \vee \neg f) \wedge (e \vee \neg d \vee \neg f) \wedge (d \vee \neg e \vee \neg f) \wedge (b \vee c \vee \neg a \vee \neg f) \wedge (b \vee c \vee d \vee e \vee \neg a \vee \neg d \vee \neg e \vee \neg f)$

Figure 4.1: CNF rules extracted from CNFN

Immediately it is possible to find useful information about this problem that was not obvious before. For example $\neg f = \mathit{True} \implies$ Class 3, or *if the tear reduction rate is reduced then do not fit contact lenses*. Alternatively, DNF rules could be constructed using a DNFN, and this possibility is demonstrated in the following three rules.

- Class 1: $(a \wedge d \wedge e \wedge f \wedge \neg b \wedge \neg c) \vee (a \wedge d \wedge e \wedge f \wedge \neg b \wedge \neg c) \vee (e \wedge f \wedge \neg d)$
- Class 2: $(f \wedge \neg c \wedge \neg d \wedge \neg e) \vee (d \wedge f \wedge \neg e)$
- Class 3: $(e \wedge e \wedge \neg a) \vee (c \wedge f \wedge \neg a \wedge \neg b \wedge \neg d \wedge \neg e) \vee (\neg f) \vee (\neg f)$

The formulae extracted from the DNFN confirm previous knowledge obtained from the CNFN, namely $\neg f = \mathit{True} \implies$ Class 3. Table 4.2 demonstrates the performance of extracted rules over the Lenses dataset. The confidence intervals of networks and their rule set overlap, so there is no statistically significant difference in error rate between the network and rule set in terms of their accuracy on the data.

	Rule Error Rate	Rule Error Rate CI (95%)
CNF Rules	0.0122	(0.0000, 0.0417)
DNF Rules	0.0156	(0.0000, 0.0417)

Table 4.2

The results have shown that LNFNs can be applied to boolean multi-class classification problems.

4.2 Features with Continuous Domains

It is possible for the inputs to a Noisy Neuron to be in the range $[0, 1]$, but when does it make sense to do so? Moreover, what do these continuous features mean? The inputs can be thought of as being the probability that the feature is present. In the discrete case either the input is there (a 1) or not (a 0). Then in the continuous case, each input feature is thought of as a probability.

How about the case where the feature range is not confined to $[0, 1]$. An option is to normalize the feature spaces, forcing them into a format compatible with LNFNs. The problem then comes down to meaning. Is possible to think of these normalized features as a probability? Consider an arbitrary (un-normalized) feature f. Assume $f \in [f_{min}, f_{max}]$ then the larger the normalized feature f_N the closer its true value is to f_{max}. Similarly if $\neg f_N = 1 - f_N$ is large then the true value is closer to f_{min}. f_N could then be the probability of sampling a $v \sim U(f_{min}, f_{max})$ that is less than f. This is not the only meaning normalized features could have; it is only an example.

Another consideration to make is, how can trained LNFN models be interpreted if the features are continuous? When the weights are binary then a Noisy neuron can be considered as a logical gate of its inputs with $\epsilon \approxeq 0$. The weights in a trained LNFN do not converge to 0 or 1 when the input features are continuous so a more general method to interpret LNFNs is needed.

Influence Model. One potential way to interpret LNFNs would be to compute the influence each input feature has on the output neurons. This method of interpretation is named an **Influence Model**. The weights for a Noisy neuron directly correspond to the influence each input has on the neuron's output.

Consequently, it is a trivial task to find an Influence Model for describing how each input feature affects the hidden layer outputs. This Influence Model is just the weights of each hidden neuron. Consider Figure 4.2 and the problem computing an Influence Model describing the influence each x_i has on the output of some o_j. The activation of each h_j is what influences the output layer. However, each h_j is influenced not only by the x_i of interest but also all the others.

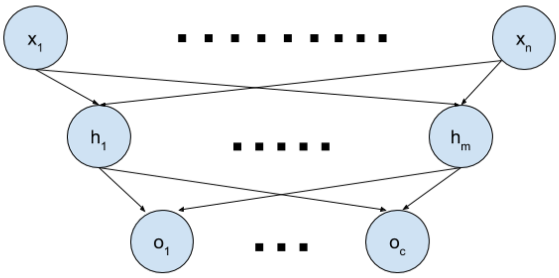


Figure 4.2: Example Network

This new $\epsilon_i^{'}$ is dependent on all the other x's, not just the x_i of interest. Consequently there is not an obvious solution to computing Influence Model for LNFNs.

Partial Influence Models While it is not obvious how to compute an Influence Model describing an LNFN entirely it is trivial to compute one describing the influence between input features and the hidden layer output. Using this Partial Influence Model can describe the hidden neurons in terms of the input features. The weights belonging to each output neuron could then be used to find the most important hidden neurons.

Continuous Rules Another way to interpret LNFNs in a continuous domain is to extract continuous rules. A threshold will be set, any input with a weight below the threshold will be considered important. Consequently, each neuron will be considered a function of its inputs that have a corresponding weight less than or equal to a given threshold t. A potential issue with this method is that the less important features are removed after the threshold is applied, this might result in more incorrect classifications by the rules. A smaller threshold will lead to simpler rules with perhaps a higher error rate whereas a larger threshold will give more complex rules with a possibly higher accuracy. This process also discards the true ϵ's which could potentially affect the accuracy of the rules.

4.2.1 Application To Iris Problem

The Iris dataset [23] is the problem of classifying a plant as one of three classes. Each example consists of four features; all are measurements taken from the plant.

All features are in centimeters and not confined to the range $[0, 1]$. Table 4.3 gives the results of training the Normalised Iris problem on LNF networks. The results show that while the performance can vary a lot, it is possible for both the CNF and DNF networks to obtain.

	Error Rate	Error Rate CI (95%)
CNF Network	0.027	(0.000, 0.111)
DNF Network	0.066	(0.000, 0.156)

Table 4.3: LNFN performance on an Iris problem testing set

If attempting to interpret these models meaning must be assigned to each of the normalized features. Informally, an atom is "truer" if the un-normalized feature is larger and "falser" if it is smaller. A NOT operation inverts this property. Table 4.4 shows the performance of extracted continuous rules on a testing set. These statistics provide evidence that continuous rules have poor performance.

	Error Rate	Error Rate CI (95%)
CNF Rules	0.439	(0.156, 0.867)
DNF Rules	0.323	(0.156, 0.511)

Table 4.4: Performance of rules extracted from LNFN on an Iris problem testing set

The question remaining is, how do these rules contribute to interpreting the LNFN model. Figure 4.3 shows the raw continuous rules extracted from a DNFN. Features $\{a, b, c, d\}$ correspond to preferring larger values of { sepal length, sepal width, petal length, petal width }. On the other hand $e = \neg a, f = \neg b, g = \neg c, h = \neg d$.

The list in Figure 4.4 shows some conclusions about the classes which can be drawn from the raw continuous rules. Having a small petal width and length means the class is more likely to be *Iris Setosa*.

1. *Iris Setosa:* $(g \wedge h) \vee (g \wedge h) \vee (g \wedge h) \vee (g \wedge h)$
2. *Iris Versicolour:* $(c \wedge e \wedge f \wedge h) \vee (c \wedge f \wedge h) \vee (c \wedge f \wedge h) \vee (c \wedge e \wedge f \wedge h) \vee (c \wedge f \wedge h)$
3. *Iris Virginica:* $(c \wedge d \wedge f) \vee (c \wedge d \wedge f) \vee (d \wedge f) \vee (c \wedge d \wedge f) \vee (c \wedge d \wedge f) \vee (c) \vee (c \wedge d \wedge f)$

Figure 4.3: Raw continuous rules extracted from a DNFN, with a threshold of 0.5

Alternatively having a larger petal length means the instance is more likely to be *Iris Versicolour* or *Iris Virginica*.

The attributes **petal width** & **sepal width** differentiate between the classes *Iris Versicolour* or *Iris Virginica*.

1. *Iris Setosa:* Smaller **petal width** ∧ Smaller **petal length**
2. *Iris Versicolour:* Larger **petal length** ∧ Smaller **petal width** ∧ Smaller **sepal width**
3. *Iris Virginica:* Larger **petal length** ∨ (Large **petal width** ∧ Small **sepal width**)

Figure 4.4: Rules representing the iris problem, extracted from a CNFN

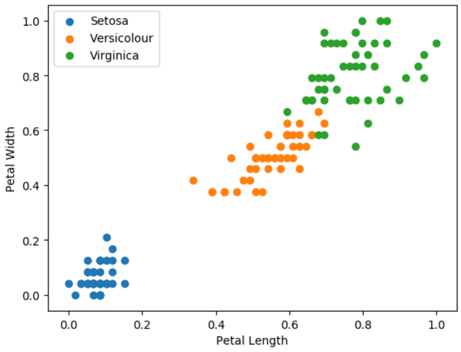


Figure 4.5: Scatter plot of Iris dataset, showing the petal length on the x-axis and petal width on the y-axis

Figure 4.5 plots the petal width and length features against each other, demonstrating the conclusions drawn make sense.

This section has shown that LNFNs can be applied to any problem with continuous features by normalization. This operation creates an additional problem; namely, it becomes difficult to interpret what the normalized features mean. Future work into LNNs might aim to develop a better way to convert feature domains to probabilities.

4.3 Summary

This chapter has demonstrated that LNFNs can be applied to more than just learning boolean truth tables. They have been shown to achieve good accuracy on multi-class classification problems, with and without discrete input features. Methods for interpreting LNFNs applied to continuous feature spaces where explored; these methods where then shown to provide useful insights into the Iris dataset.

In the context of the Lenses problem, it was possible to extract boolean rules from the LNFNs which had a statistically equivalent performance to the networks. These rules also provided insight into the problem which was not apparent from the data. The LNFNs where able to achieve a low error rate (sometimes 0%) on the Iris data set. It was possible to extract useful information from the trained models which highlighted the most important features when making classifications. The experiments with the Iris problem also revealed limitations of LNNs

Chapter 5

Logical Neural Networks

This chapter discusses development of Logical Neural Networks as a generalization of LNFNs. There are two critical issues with LNFNs. Firstly, the number of hidden units becomes infeasible as the size of the input space increases. Secondly, if the problem size is small enough and the network can be trained the volume of hidden neurons allows for the possibility to memorize the input data. Using what has been learned about LNFNs the class of Logical Neural Networks (LNNs) can be defined.

Definition 5.0.1. A Logical Neural Network is an ANN where each neuron has a noisy activation.

LNNs have a more flexible structure, allowing for deeper networks and hidden layers with a variable number of neurons. The current LNN model has been shown to have poor performance when compared to a Multi-Layer Perceptron Network [5]. An LNN, consisting of Noisy-OR hidden units and Noisy-AND outputs, was shown to perform worse than an MLPN. Two issues are created by removing the restrictions imposed by the LNFN definition (Definition 3.0.3) which must be addressed

1. Noisy neurons cannot consider the presence of the negation of an input. This issue was a problem for LNFNs as well, however, given that only the negations of atoms need to be considered to learn a CNF or DNF it was easily fixed by presenting the network with each atom and its negation. The problem can not be solved so easily for LNNs. A boolean formula cannot always be represented by only AND and OR gate, i.e, the set of operations $\{AND, OR\}$ is not functionally complete.

2. Another problem faced by LLNs that are not restricted to be ether a CNFN or DNFN is that the structure of the network will have a higher impact on whether the problem can be learned.

Resolving Issue 1 involves making our operation set functionally complete, this requires the NOT operation. There are two ways to include the NOT operation in the LNNs; one is to augment the inputs to each layer, so it becomes the regular input concatenated with its negation. Another is to derive a parametrization of Noisy gates which can learn to negate inputs. However, both these have no way to enforce mutual exclusivity between an input and its negation.

5.1 Modified Logical Neural Network

5.1.1 Connections Between Layers & Parameters

Figure 5.1 provides a new structure for the connections between the lectures.

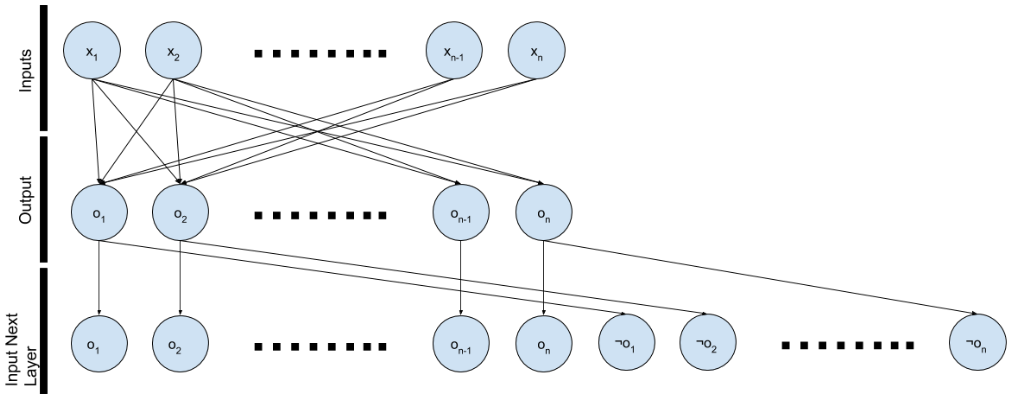


Figure 5.1

Figure 5.1 shows the new LNN structure which includes an implicit NOT operation. The input to each layer consists of the true output of the previous plus the negated output from the last layer. If a layer has 20 outputs then there are 40 inputs to the following layer.

5.1.2 Softmax Output Layer

The softmax function takes a vector of real values to a vector of values in the range $[0, 1]$ that equal 1 when summed. The softmax function highlights the difference between the maximum and smaller values by increasing this difference. It is used in multi-class classification to generate a probability distribution over the different outcomes and promote mutual exclusivity between the different classes.

The old LNN structure does not include a Softmax layer as the output layer. A traditional Softmax layer would not be effective as the neuron outputs are limited to the range $[0, 1]$. Figure 5.2 shows an example of the softmax of a vector containing probabilities, instead of highlight the difference between 0.1 and 0.9 the resulting vector has the values closer together. Consider the following vector of probabilities $P = \{p_1, ..., p_n\}$ where p_i is the probability the given example belongs to class i.

$$\begin{aligned} A &= \{01, 0.9\} \\ SoftMax(A) &= \{\frac{e^{0.1}}{e^{0.1} + e^{0.9}}, \frac{e^{0.9}}{e^{0.1} + e^{0.9}}\} \\ &\approxeq \{0.29, 0.71\} \end{aligned}$$

Figure 5.2: Demonstration of Softmax on probabilities

Then define $p_i^{'} = \frac{p_i}{\sum_j p_j}$, performing this action to generate a vector $P^{'}$ where each of the classes are mutually exclusive. This operation can be thought of as a "Logical Softmax". Adding a "Logical Softmax" may guarantee mutual exclusivity of each class but will it cause the network to be less interpretable? Consider that without with no softmax then the network outputs directly represent the probabilities that the input vector is class k given the parameters. Once the "Logical Softmax" has been introduced then it becomes less clear what the non softmaxed probabilities represent. The softmax introduces a dependency between the output neurons of the network which might cause a decrease in interpretability. This is a question which will be explored during experimentation with the modified LNN architecture.

Chapter 6

Evaluation Of Logical Neural Networks

To evaluate Logical Neural Networks their performance and interpretability are explored. In Chapter 2 different methods for evaluating the interpretability of models where presented. The method used in this report is *"Application-Grounded Evaluation"* [1] which relies on domain experts to assess the interpretability of a model. The two problems MNIST [24] and Tic-Tac-Toe [23] will be used in this evaluation.

These problems are a suitable choice as their simplicity makes most humans domain experts. Tic-Tac-Toe has a simple rule set that is easy to comprehend. MNIST is the problem of classifying handwritten digits, a task that is performed on a daily basis by most humans. Five criteria that will be used for this evaluation.

1. Performance comparison between the old and new Logical Neural Network Structures.
2. Performance comparison between Multi-Layer Perceptron Network and Logical Neural Networks.
3. Performance comparison between Logical Neural Networks with and without a Logical Softmax.
4. Comparison of interpretability between MLPNs and new/old LNNs
5. Comparison of interpretability between MLPNs (using LIME) and new/old LNNs

It was conjectured that AND neurons were not effective for low-level feature extraction [5]. Throughout this evaluation, AND neurons will be used for this task to determine whether the conjecture holds.

6.1 Performance of Logical Neural Networks

To evaluate the performance of Logical Neural Networks (LNNs) different configurations of Logical Networks will be trained on the MNIST dataset. Any experiments with an LNN architecture will be trained with and without a Logical Softmax.

1. **(OR → AND) Old Architecture:** This will consist of 30 hidden OR neurons.
2. **(OR → AND) Architecture:** Same as 1 but with the new LNN architecture

3. **(OR → AND → AND) Architecture:** 60 hidden OR neurons, 30 hidden AND neurons

4. **(OR) Architecture:** Inputs are directly connected to the outputs which have a Noisy-OR activation

5. **(AND) Architecture:** Inputs are directly connected to the outputs which have a Noisy-AND activation

6. **(AND) Old Architecture:** Inputs are directly connected to the outputs which have a Noisy-AND activation, this network is also running on the old architecture.

Sigmoid models will also be run to provide a performance comparison point for the Logical Neural Networks.

Results Each network is trained 30 times to average the results over different initial conditions. Tables 6.1 & 6.2 display the Sigmoid and LNN results retrospectively. Each error rate reported is obtained from an unseen test set.

Network Config	Error Rate	Error Rate CI	Error Rate (with SM)	Error Rate CI (with SM)
60 → 30	0.034	(0.031, 0.038)	0.035	(0.030, 0.040)
30	0.046	(0.042, 0.050)	0.045	(0.041, 0.050)
N/A	0.085	(0.085, 0.085)	0.084	(0.084, 0.084)

Table 6.1: Results of experiments with Sigmoid models

Network Config	Error Rate	Error Rate CI	Error Rate (LSM)	Error Rate CI (LSM)
(OR → AND) Old	0.105	(0.098, 0.115)	0.048	(0.043, 0.052)
(OR → AND)	0.088	(0.079, 0.094)	0.042	(0.039, 0.046)
(OR → AND → AND)	0.053	(0.046, 0.060)	0.032	(0.029, 0.036)
(OR)	0.382	(0.381, 0.384)	0.334	(0.331, 0.336)
(AND)	0.137	(0.135, 0.139)	0.076	(0.075, 0.079)
(AND) Old	0.312	(0.311, 0.314)	0.111	(0.109, 0.114)

Table 6.2: Results of experiments with Logical Neural Network models

The experimental results lead to the following conclusions.

1. *New LNN Architecture Gives Better Performance Than the Old:* The confidence intervals for test set performance do not overlap for Architectures **(OR → AND) Old** and **(OR → AND)** (without LSM). This performance increase can also be observed from Architectures **(AND) Old** and **AND**.

2. *Adding An LSM Improves Performance:* Every LNN using the new architecture gets a statistically significant performance increase when a LSM is added.

3. *It Is Unclear What Provides The Largest Improvement, New Structure or LSM:* The experiments show that both the new architecture and LSM give a statistically significant improvement in performance. From observing **(OR → AND) Old** and **(OR → AND)** (with LSM) it is possible to see that when a LSM is added then the change in architecture does not introduce an increase in performance. However the opposite is true when comparing the (AND) and (AND) Old networks.

4. *Noisy-AND Neurons are Good at Low-Level Feature Extraction:* The AND network with an LSM has a statistically significant performance increase when compared to the OR network with an LSM or Sigmoid net with no hidden layers.

6.2 Interpretability of Logical Neural Networks

There are two cases of interpretability; one where the input features are boolean and the other when they are continuous.

6.2.1 Discrete Case (Rule Extraction)

To assess the rule extraction of LNNs the Tic Tac Toe [23] problem will be the benchmark.

This problem involves classifying tic-tac-toe endgame boards and determining whether ′x′ can win. There are 9 categorical attributes, representing each cell of the board; each attribute can be ′x′ (x has a piece here), ′o′ (o has a piece here) or ′b′ (blank).

This gives a total of 27 attributes after conversion to a binary string. There are a total of 958 instances, 70% of which will be used for training, the rest for testing. Using the new LNN with the structure described in Figure 6.1 (using 30 hidden OR neurons) can achieve error rates displayed in Table 6.3. Each experiment is averaged over 30 runs

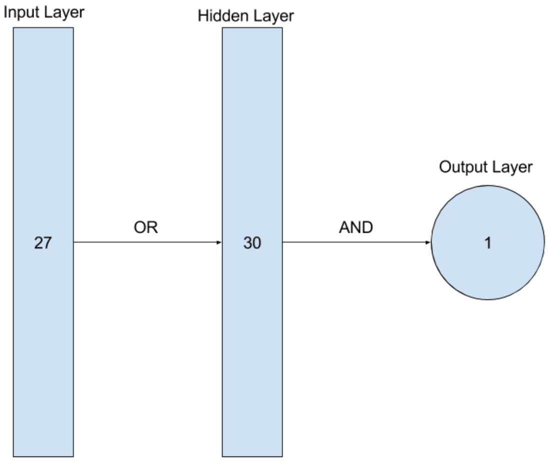


Figure 6.1: Network architecture for learning Tic-Tac-Toe

	Net Error Rate	Net Error Rate CI	Rule Error Rate	Rule Error Rate CI 95%
Training	0.0035	(0.0035, 0.0035)	0.0000	(0.0000, 0.0000)
Testing	0.0015	(0.0015, 0.0015)	0.0259	(0.0104, 0.0451)

Table 6.3: Results of experiments with Logical Neural Networks on the Tic Tac Toe problem

If instead ANDs are used as the hidden neurons and ORs for the outputs then the LNN can achieve statistically equivalent performance to the architecture in Figure 6.1. Table 6.4 shows the results from these experiments.

	Net Error Rate	Net Error Rate CI	Rule Error Rate	Rule Error Rate CI 95%
Training	0.0035	(0.0035, 0.0035)	0.0000	(0.0000, 0.0000)
Testing	0.0015	(0.0015, 0.0015)	0.0038	(0.0, 0.0139)

Table 6.4: Results of experiments with Logical Neural Networks on the Tic Tac Toe problem on a AND - OR Network

These experiments provide substantial evidence against the conjecture stating that ANDs are not effective for extracting low-level features.

Evaluation of LNN Rules

Figure 6.2 shows a rule taken at random from the AND-OR network. This rule is saying that if the middle column contains all X's then X has won. There is redundancy in these rules as a cell can only be occupied by either an X, O or nothing.

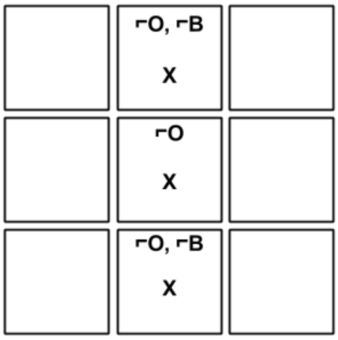


Figure 6.2: A pictorial example of a rule extracted from the AND-OR network

A future goal might be to implement a way to build mutual exclusivity of attributes into the network; this could result in further simplification of rules.

The rule extraction algorithm given in Figure 3.8 is an eclectic, algorithm. This category describes algorithms that examine the network by inspecting each neuron but extract rules which represent the network as a whole [9]. The algorithm is not portable as it can only be applied to networks with the LNN architecture.

Finally, what is the quality of the rules extracted from LNN? This is measured by the Accuracy, Fidelity, Consistency, and Comprehensibility [8].

1. **Accurate:** As demonstrated experimentally the extracted rules can generalize well when presented with unseen examples.

2. **Fidelity:** The experiments show that the rules perform very similar to the network they where extracted from as both have a similar performance on the testing set.

3. **Consistency:** These rule sets are consistent, shown by the low error rate when the neural network is trained from many different initial conditions.

4. **Comprehensibility:** The upper limit of the total number of rules is a function of the networks size. The OR-AND model yields a complex set of rules, with a total 35 of the clauses. Whereas the AND OR model only utilizes 11 clauses. The network structure also contributes to the comprehensibility of the rules. For instance, each clause in the AND OR rules represent a situation where the rule set is true. Whereas in the OR AND model each clause must true.

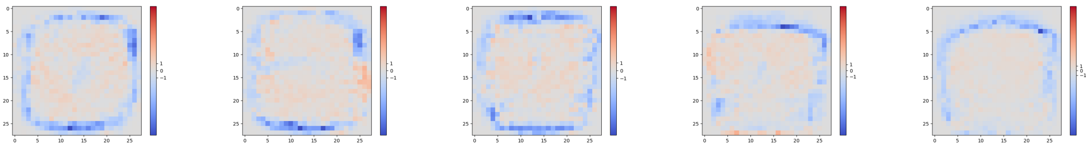
6.2.2 Continuous Case

Chapter 4 discusses three methods for interpreting Logical Normal Form Networks which can also be applied to the generalized LNN architecture. Extracting a continuous rule set would be to large too interpret given MNIST has 784 features. For this reason, Influence and Partial Influence models will be constructed and presented as images.

LNNs with no hidden layer can be interpreted easily by showing pictorial representations of Influence models. Interpreting Multi-Layer LNNs will be achieved by constructing Partial Influence models and then displaying pictorial representations of the most important features that contribute to the classification of an example as the digit 1. Multi-Layer perceptron networks will be visualised by constructing a heat map of the weights.

No Hidden Layer Networks

Sigmoid Network Figure **??** shows the weights from a sigmoid network. The blue/red represents the negative/positive weights. Each neuron representation has a blue border around the average digit which the neuron corresponds to; this is especially visible in the representation for 0.

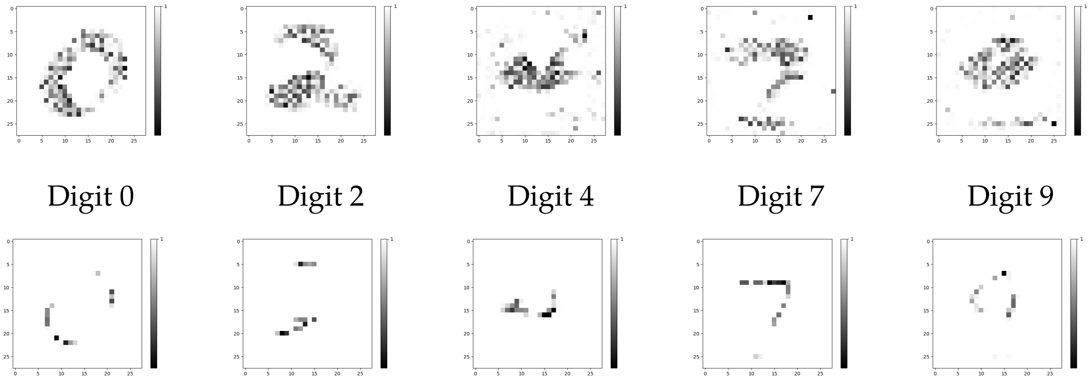


Digit 0 Digit 2 Digit 4 Digit 7 Digit 9

Figures representing the output neurons of a sigmoid neural network with no hidden layers

AND Network Old Architecture (With and With Out LSM) The images in Figure 6.19 represent how relevant each input feature is towards a classification of the digits 0, 2, 4, 7 and 9. The top and bottom rows represent the relevance for an AND architecture with and without LSM retrospectively. The darker pixels represent more relevant features, where white is completely irrelevant.

The network without an LSM is using the pixels which occur in all representations of each digit, whereas the network with an LSM is using the average filling to achieve its classifications. Both models are interpretable as it is possible to understand the logic used in their decision making. Each of the models describes the problem differently, each uncovering distinct information. The model without an LSM shows describes what is common between every drawing of that digit. The model with an LSM describes which parts of each digit vary the most/least. The benefit of the model with an LSM is that the pictorial representations appear to look more like the digits which they classify. However the model without an LSM has a sparser representation, so the model is dependent on less information.



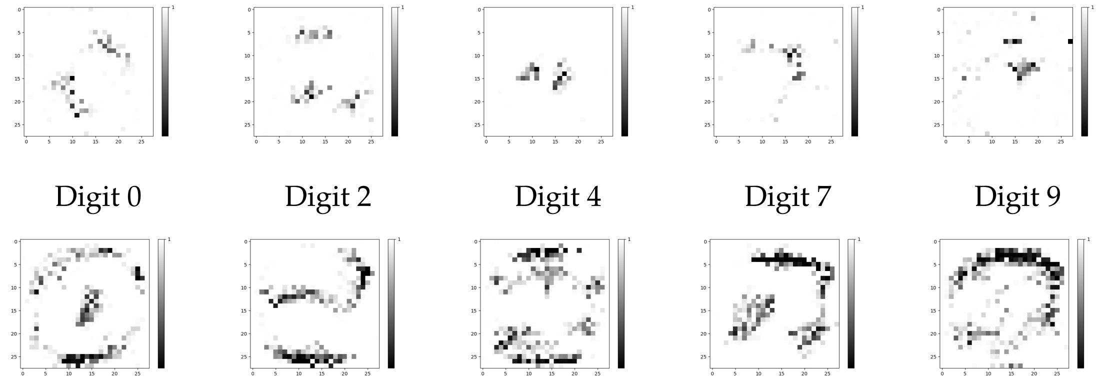
Digit 0 Digit 2 Digit 4 Digit 7 Digit 9

Digit 0 Digit 2 Digit 4 Digit 7 Digit 9

Figure 6.19: Images representing the weighted connections between the inputs and outputs of an AND network with the old architecture. The top and bottom rows are with/without an LSM respectively

AND Network (With LSM) The model here is an AND network running on the new architecture (which considers the NOTs of inputs) with an LSM. The new architecture means that each input feature can positively contribute to a classification or negatively contribute (i.e., if its present then the classification is less likely). Figure 6.30 shows the how relevant each input feature is with regards to a positive or negative classification for the digits 0, 2, 4, 7 and 9. The top row of images represent positively weighted inputs, and the bottom row represents the negatively weighted inputs.

The inputs which are positively weighted are the pixels that occur in many of the representations of the digit. Negatively weighted inputs represent the border of the digit, if pixels on this border are present then the neuron is less likely to be active. Using the classification of 0 as an example; the network does not like pixels in the middle as the center of a zero should be empty. The outer circle represents the border of the average 0, if these pixels are present then its less likely to be a zero as most instances of a 0 do not have pixels present there.

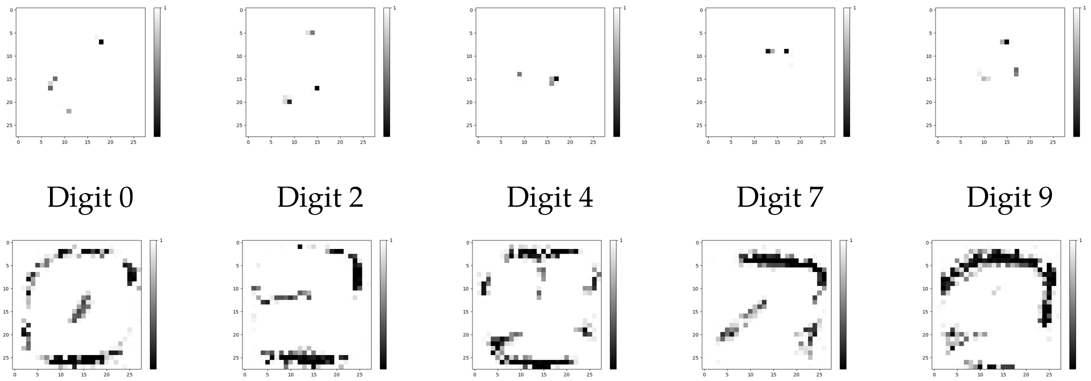


Digit 0 Digit 2 Digit 4 Digit 7 Digit 9

Not Digit 0 Not Digit 2 Not Digit 4 Not Digit 7 Not Digit 9

Figure 6.30: Images representing the weighted connections between the inputs and outputs for an AND network with an LSM. The top and bottom rows represent the positive/negative contributions retrospectively.

AND Network (Without LSM) Figure **??** shows the positive/negative contributions to classification of the digits 0, 2, 4, 7 and 9. These features display the same patterns as the AND model with an LSM (Figure 6.30) however the representations here are less noisy and have harder boundaries.



Digit 0 Digit 2 Digit 4 Digit 7 Digit 9

Not Digit 0 Not Digit 2 Not Digit 4 Not Digit 7 Not Digit 9

Figure 6.41: The top/bottom rows of images represent the positively/negatively weighted input features retrospectively

Conclusion of Interpretability for LNNs with No Hidden Layer The logical Softmax introduces more noise into the weights but does not diminish the interpretability of the system. Using the new structure (i.e., adding NOTs of inputs) appears to change the means of which the LNN classifies the digits. Without NOTs, the network positively weights the pixels which make-up the filling of the digits. On the other hand when the nots are added the network negatively weights pixels sitting just outside the border of the average digit. These experiments provide evidence towards Logical Neural Networks being more interpretable than MLPNs. While it is possible to observe some of the MLPNs decision-making processes, the LNN is more transparent and simpler leading to a more interpretable model.

Single Hidden Layer Networks

To interpret these multi-layer networks Partial Influence models will be constructed and pictorial representations of the most important features that contribute to classification as a one will be displayed.

Sigmoid Network The features in Figure 6.43 positively contribute to the classification as a 1, and do not appear to have any clear relation to the digit 1. The features in Figure 6.42 contribute to the classification not being 1; they also appear not to have any relationship with the digit 1.

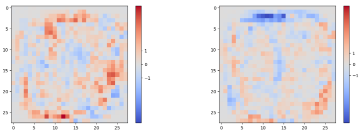


Figure 6.42: Features that negatively contribute to the classification of 1

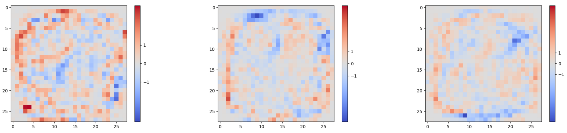


Figure 6.43: Features that positively contribute to the classification as a 1.

OR → AND Network Old Structure (With Out LSM) It is not immediately apparent how the features in Figure 6.44 make up the digit 1. Each hidden feature is an OR of it's inputs, as such only one of the dark pixels needs to be present for the neuron to be active. The first and last could be the stem of a 1 whereas the middle one contains a dark bar at the bottom which could be the base.

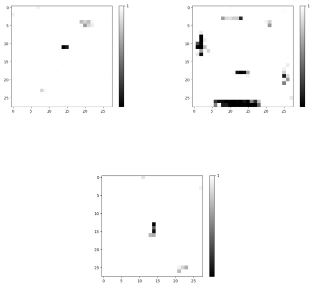


Figure 6.44: Features positively contributing to classification as a 1 for the OR → AND Network

OR → AND Network Old Structure (With LSM) Figure 6.45 shows hidden features extracted from an OR-AND Network. These features are not representative of a 1.

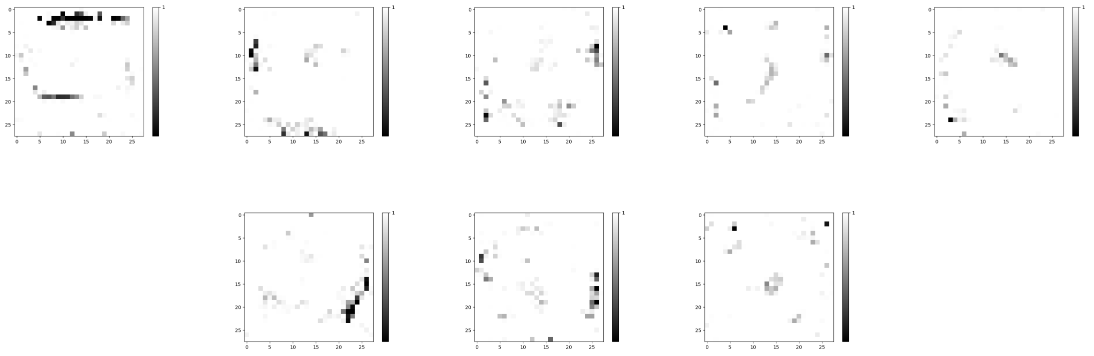


Figure 6.45: Features positively contributing to the classification of 1 for the OR → AND Network Old Structure (With LSM)

OR → AND Network (With Out LSM) The images in Figure 6.46 represent the hidden features which positively contribute to the classification of an example as the digit 1. The top and bottom rows correspond to the input features that positively/negatively contribute to each neurons activation. It is not immediately apparent what the features in Figure 6.46 represent.

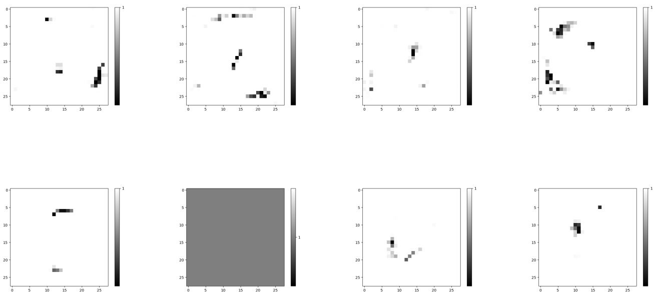


Figure 6.46: Features positively contributing to classification as a 1 for OR → AND Network (With Out LSM)

Figure 6.47 shows the single feature which if present then the classification is less likely to be the digit 1. This feature appears to be highly active if the pixels lying outside an area which looks like the average 1 shape. While this model is difficult to interpret it does provide more information than previous OR-AND models using the old LNN architecture.

OR → AND (With LSM) Figure 6.49 show the positive features for an OR AND model with an LSM. The features extracted from this network do not appear to correspond to a 1, not in a way which is immediately interpretable.

The presence of the features represented in Figure 6.48 mean the classification is less likely to be a 1. These features do not appear to relate to the digit 1, but instead pixels that would not usually occur in the digit 1.

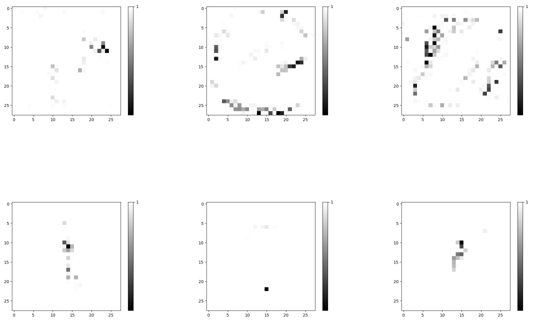


Figure 6.48: Features contributing to the classification not being 1 for OR → AND (With LSM)

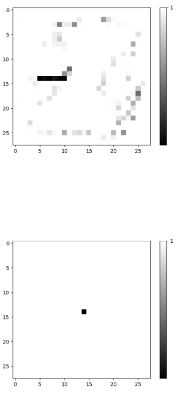


Figure 6.47: Features contributing to classification not being 1 for OR → AND Network (With Out LSM)

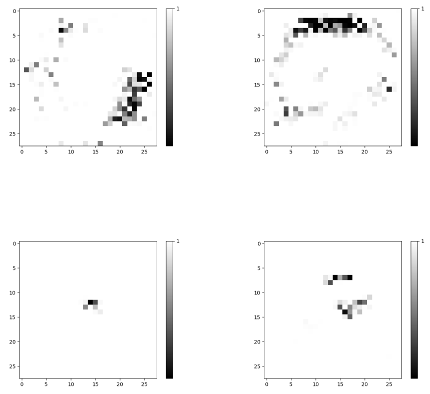


Figure 6.49: Features positively contributing to the classification being 1 for OR → AND (With LSM)

AND → OR Model (With Out LSM) Figure 6.50 shows the single feature which positively contributes to the classification as the digit 1. This feature has learned to positively identify the stem of a 1 and negatively identify .

This network has a small number of features associated with each classification. Regarding classifying the digit 1, there is only one hidden feature. This feature has learned to positively identify the stem of a 1 and negatively identify the pixels outside the border of a 1.

AND → OR Model (With LSM) Similarly to the AND-OR Model without the LSM this model appears to positively like inputs on the stem of a 1 and dislike any pixel outside the common border of a 1. These features are shown in Figure 6.51

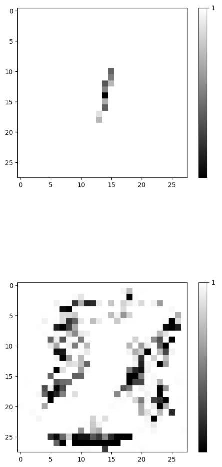


Figure 6.50: Features positively contributing to classification as 1 for AND → OR Model (With Out LSM)

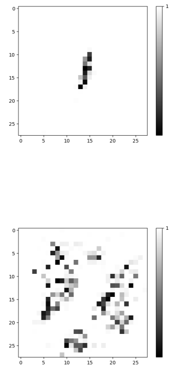


Figure 6.51: Features contributing to classification not being 1 for AND → OR Model (With Out LSM)

Conclusion of Interpretability for LNNs with One Hidden Layer The results of the experiments show that in the context of multi-layer LNNs adding an LSM introduces more noise into the feature maps. Similarly to before the sigmoid weights do not appear to relate to digit 1 in any meaningful way.

6.2.3 Results of Interpretability Experiments

From the above experiments and discussions, the following conclusions can be made.

1. *Rules Extracted are Interpretable:* The Tic-Tac-Toe problem experiments showed that the rules extracted from LNNs where interpretable.

2. *Adding NOTs Improves Interpretability:* Using the new structure with NOTs allows the network to place a greater emphasis on the presence or absence of various pixels. Without the Nots not many of inputs are of great importance; instead many have a relatively small relevance.

3. *Adding a Logical Soft Max does not hinder the interpretability (on the new architecture):* By comparing the two different network architectures with and without an LSM, it is possible to see that the interpretability is not directly affected and the features representing the classification of the digit 1 are not significantly changed.

4. *Interpretability of Network is Heavily Dependent on Structure:* As shown through the previous examples the OR → AND networks harder to interpret than the AND → OR architecture.

6.3 Comparison Between LNNs and Existing Methods

6.3.1 Comparison between LNN Interpretability and LIME

LIME aims to build trust in the model by explaining how specific predictions are made. LIME is also model agnostic, so it is a highly portable algorithm. In comparison, LNN interpretability comes down to directly identifying what input features contribute to each output neuron being active. Figure 6.52 shows the LIME algorithm applied to three different ANNs, two LNNs and one MLPN. Each image shows what features are important in classifying the example as a 1.

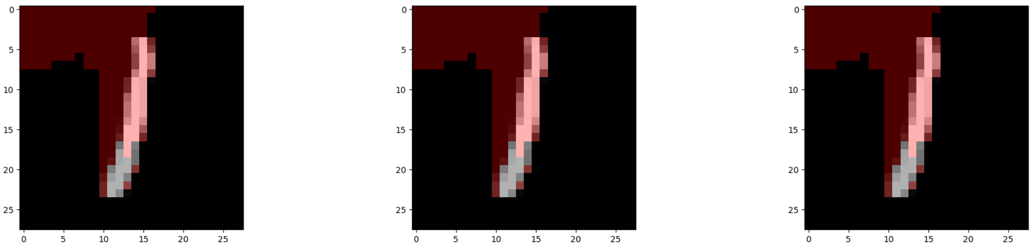


Figure 6.52: LIME Intepretation of classifiers. Left is the LNN AND model (no hidden layers). Middle is a 2 layer LNN with 60 and 30 nodes respectively, finally the activations are OR-AND-AND LNN. Lastly right is an MLPN with 2 hidden layers, with 60 and 30 nodes respectively.

The LIME filters out the features which have a low importance. Figure 6.51 says that for the three different classifiers the same features are most important to correctly classifying this instance of a 1.

6.3.2 Comparison Between LNNs and Fuzzy Logic Networks

The paper presenting Fuzzy Logic Networks (FLN) evaluates their performance four benchmark problems including the Vehicle [23] dataset. The Vehicle problem concerns attempting to classify a given silhouette as one of four vehicles types. The data set consists of 18 features extracted from car silhouette images.

	Net Error Rate	Net Error Rate CI	Rule Error Rate	Rule Error Rate CI 95%
Training	0.101	(0.079, 0.132)	0.649	(0.380, 0.762)
Testing	0.173	(0.130, 0.224)	0.648	(0.398, 0.763)

Table 6.5: Results of experiments with Logical Neural Networks on the Vehicle Problem using an OR → OR → AND network architecture

Tabel 6.5 show the error rates of OR → OR → AND LNN on the Vehicle problem. The LNN network achieved an average error rate of –& on a testing set compared to 28.71% error rate for the FLN. The difference between the LNN and rule error rates is statistically significant. While the rules extracted from an LNN perform worse than the network its self, they have comparable performance to rules taken from an FLN. An LNN rule set obtains 64.8% average error rate on the test set whereas the FLN rules achieve an average error rate of 67.8%.

One advantage that the FLN has over LNNs is that the rule sets are more compact and thus are more straightforward to interpret, this would suggest that FLN rules are better. Future work on LNNs might aim to implement an algorithm which can automatically simplify the extracted rules; this could lead to simpler expressions.

6.4 Summary Of Logical Neural Network Evaluation

Through the experiments carried out, the following observations can be made. The new LNN structure (adding NOTs) results with a statistically significant increase in accuracy with some network architectures. The new LNN structure achieved statistically equivalent or better performance than the MLPN nets with an equivalent number of hidden neurons/layers. For any LNN (new structure or old) adding a Logical Softmax improved performance by a statistically significant margin. Any LNN (new or old structure) was more interpretable than the corresponding MLPN. These experiments have therefore shown that the LNN networks have statistically equivalent performance to MLPNs on the MNIST dataset. Evidence has also been provided in support of LNNs have a more interpretable model than MLPNs.

The LIME algorithm is more robust than LNN Interpretability in the sense that it is model agnostic. On the other hand, interpreting LNNs provides a clearer picture of the model's decision-making process. FLNs where shown to be better than LNNs because of their ability to generate simpler rule sets. It could, however, be possible to close this gap by developing an algorithm to simplify the rules learned by an LNN.

Chapter 7

Application to Autoencoders

This chapter demonstrates the flexibility of the LNN architecture by applying them in the context of Auto Encoders.

Autoencoders [25] [26] are a network architecture trained to learn an encoding of the data. An autoencoder is trained by presenting it with an instance, which it then converts to an alternate feature space using an encoder, and then back using a decoder, the objective is to minimise the loss between the instance and its reconstruction. The encoder and decoders are Artificial Neural Networks.

An application of autoencoders is to learn a new encoding which has a smaller dimension, i.e., dimensionality reduction. The case where there is one linear layer doing the encoding and decoding is called **Linear Autoencoder**. Logical Autoencoders are proposed (Definition 7.0.1) as an alternative means to lower the dimensions of a dataset.

Definition 7.0.1. A **Logical Autoencoder** is an Autoencoder where the encoder and decoder are LNNs

Linear, Sigmoid and Logical Autoencoders will be compared. The network structure will be the same in all cases, only differing in the activations used. Experiments carried out will compress the MNIST feature space (784 dimensions) into 20. The accuracy and interpretability of the features will be explored. Each model was trained for 30 epochs.

Result of Linear Autoencoder (LAE) A linear autoencoder obtained an MSE of 21.34 on the training and 21.25 on the test data. Figure 7.1 shows a visualization of some features learned by the Linear Auto Encoder

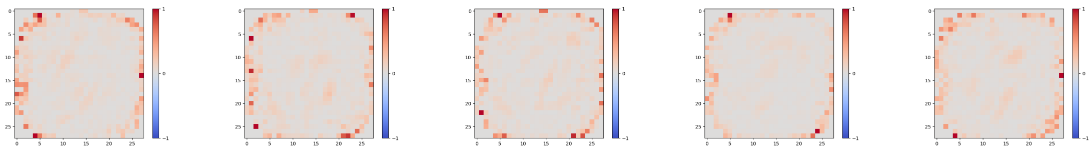


Figure 7.1: Visualisations of 5 features learned by the Sigmoid Auto Encoder. The red colors correspond to positive weights and the blue to negative

Result of Sigmoid Auto Encoder (SAE) A Sigmoid Autoencoder achieved an MSE of 14.43 on the training data and 14.25 on the testing data. Figure 7.2 show the visual representations of the features learned.

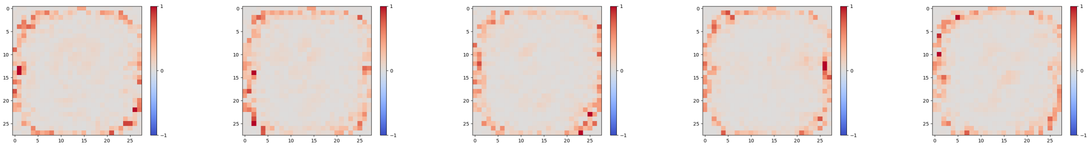


Figure 7.2: Visualisations of 5 features learnt by the Sigmoid Auto Encoder. The red colours correspond to positive weights and the blue to negative

Result of Logical Auto Encoder (LoAE) A logical autoencoder, consisting of a single AND layer for both the encoder and decoder, was able to compress the feature space to 20 dimensions and achieve a Mean Square Error (MSE) of 20.55 on the training set and 20.22 on the testing set. Figure 7.3 shows five features learned by the AND-Logical Autoencoder. Each image in the top row represents the input features which positively contribute to the activation. The bottom row corresponds to the inputs which negatively contribute to each activation.

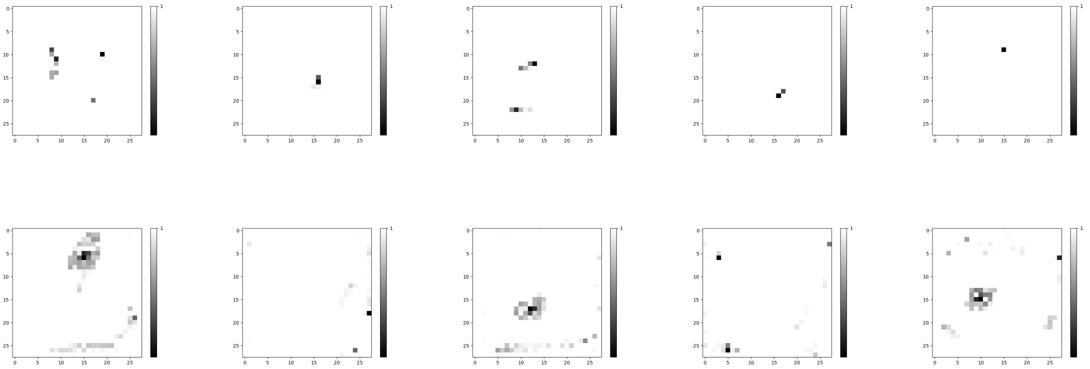


Figure 7.3: Visualisations of 5 features learned by the Logical Auto Encoder. Darker regions correspond to pixels which are relevant.

Discussion While the results show that the Sigmoid Auto Encoder (AE) outperforms the Logical one this is an example of how Logical Neural Networks can be applied to different situations in Machine Learning where the goals are different from classification. Comparing Figures 7.1 & 7.2 & 7.3 show that the Logical Autoencoder takes a different approach to decomposing the MNIST dataset, where the Sigmoid and Linear took to be similar. Features learned by the Logical AE are sparse and possible interpretable as each one can be viewed as a logical AND of the input features.

Chapter 8

Conclusion

The growing number of situations where Artificial Neural Networks (ANNs) are used is driving the development of interpretable models. Being able to defend an ANNs decision can protect users from unsafe or ethically biased systems and protect companies utilizing such systems from breaching EU regulations.

A study of Logical Normal Form Networks developed algorithms initialize network weights (leading to good learning conditions) and extract rules from trained models. Through experimentation, such networks were demonstrated to have statistically equivalent performance and generalization to Multi-Layer Perceptron Networks. Training LNFNs on the Lenses and Iris datasets demonstrated their ability to learn multi-class classification problems and give insight into the data from their interpretable trained representation.

The foundational work developed the tools to derive modifications to the LNN structure giving them statistically equivalent performance to standard Multi-Layer Perceptron Networks and improving the interpretability of the learned models. Consequently, this report has shown that LNNs are a suitable alternative to Multi-Layer Perceptron Networks (of an equivalent size), obtaining a simpler and more interpretable model without sacrificing accuracy. By observing the weights of LNNs trained over the MNIST dataset, it was possible to determine what input features contributed to each classification and verified that the logic learned was sensible. LNNs where also shown to have comparable performance to recently developed network architectures which take a similar approach to what was done here. Beyond applications to classification tasks Logical Neural Networks where applied to Autoencoders. While the performance of Logical Autoencoders was worse than standard Sigmoid Autoencoders, it demonstrates the flexibility of LNNs and the range of situations where they can be applied.

Perhaps the most significant limitation of LNNs comes down to interpreting multi-layer logical neural networks. Because of this limitation, any future work which can develop better ways to interpret these models would increase the power of LNNs. These future studies might also focus on establishing the model interpretability more scientifically, on a larger number of problem domains and individuals who have the domain knowledge to assess how interpretable the model is.

This report has provided a formal foundation for Logical Neural Networks and shown their ability not only to learn accurate but also interpretable models. Limitations of LNNs have been identified, but this does not diminish their potential, but instead, provides avenues to increase their application.

Bibliography

[1] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017.

[2] A. Caliskan, J. J. Bryson, and A. Narayanan, "Semantics derived automatically from language corpora contain human-like biases," *Science*, vol. 356, no. 6334, pp. 183–186, 2017.

[3] C. of European Union, "General data protection regulation," 2016.

[4] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a" right to explanation"," *arXiv preprint arXiv:1606.08813*, 2016.

[5] J. Laurenson, "Learning logical activations in neural networks," 2016.

[6] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[7] S. Ruggieri, D. Pedreschi, and F. Turini, "Data mining for discrimination discovery," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 2, p. 9, 2010.

[8] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-based systems*, vol. 8, no. 6, pp. 373–389, 1995.

[9] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich, "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1057–1068, 1998.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, ACM, 2016.

[11] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[12] R. E. Neapolitan *et al.*, *Learning bayesian networks*, vol. 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[13] R. J. Williams, "The logic of activation functions," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, pp. 423–443, 1986.

[14] C. Herrmann and A. Thier, "Backpropagation for neural dnf-and cnf-networks," *Knowledge Representation in Neural Networks, S*, pp. 63–72, 1996.

[15] L. B. Godfrey and M. S. Gashler, "A parameterized activation function for learning fuzzy logic operations in deep neural networks," *arXiv preprint arXiv:1708.08557*, 2017.

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[18] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[20] S. K. Kumar, "On weight initialization in deep neural networks," *arXiv preprint arXiv:1704.08863*, 2017.

[21] N. Balakrishnan, *Continuous multivariate distributions*. Wiley Online Library, 2006.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] M. Lichman, "UCI machine learning repository," 2013.

[24] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits,"

[25] P. Baldi and Z. Lu, "Complex-valued autoencoders," *Neural Networks*, vol. 33, pp. 136–147, 2012.

[26] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

Appendices

Appendix A

Proofs

A.1 Proof Of Theorem 3.0.1

Proof. Theorem 3.0.1 Let T be the truth table for a boolean function B. The atoms of B are $x_1, ..., x_n$. T has exactly 2^n rows. Construct an LNFN, L, in the following manner. L has 2^n hidden units and by definition L has one output unit. The inputs to L are $i_1, ..., i_{2n}$ where i_1, i_2 represent $x_1, \neg x_1$ and so on. Let $\epsilon_b = 1$ for every neuron.

Let h_k denote hidden unit k. h_k has the weights $\epsilon_{k,1}, ..., \epsilon_{k,2n}$, where $\epsilon_{k,m}$ represents input i_m's relevance to the output of h_k. Similarly the output unit o has weights $\mu_1, .., \mu_{2^n}$ where μ_m represents the relevance of h_m to the output of o.

Assume L is a DNF Network. Starting from row one of the table T, to row 2^n. If row a corresponds to False then set $\mu_a = 1$ (i.e. hidden node a is irrelevant), otherwise the row corresponds to True, then $\mu_a = Z$, where Z is a value close to 0 (any weight for a Noisy neuron cant be exactly 0). For each $\epsilon_{a,m}$ if the corresponding literal occurs in row a of the truth table then $\epsilon_{a,m} = Z$ other wise $\epsilon_{a,m} = 1$.

Claim: For some assignment to the atoms of B, $x_1 = v_1, ..., x_n = v_n$ where $v_i \in \{0, 1\}$. Then $L(i_1, ..., i_{2n}) = B(x_1, ..., x_n)$.

Assume $B(x_1, ..., x_n) = 1$ for the assignment $x_1 = v_1, ..., x_n = v_n$ corresponding to row a of T. Then if i_k is not considered in row a then $\epsilon_{a,k} = 1$ and if it is present then $i_k = 1$. The output of h_a is given by

$$\begin{aligned} &= \prod \epsilon_{a,m}^{1-i_m} \\ &= Z^{\sum_{i_k=1}(1-i_k)} \\ &= Z^0 \end{aligned}$$

Demonstrating that $\lim_{Z\to 0} Out(h_a) = \lim_{Z\to 0} Z^0 = 1$. Consider the activation of o, it is known that $\mu_a = Z$ consequently $\lim_{Z\to 0} \mu_a^{h_a} = \lim_{Z\to 0} Z^1 = 0$, therefore

$$\lim_{Z\to 0} Out(o) = 1 - \prod_{m=1}^{2^n} \mu_m^{h_m} \tag{A.1}$$

$$= 1 - 0 = 1 \tag{A.2}$$

Therefore $L(i_1, ..., i_{2n}) = 1$. Alternatively if $B(x_1, ..., x_n) = 0$ then no hidden neuron will have activation 1, this can be demonstrated by considering that any relevant neuron (i.e. corresponding $\mu \neq 1$) will have some input weight pair of i_m ϵ_m such that $\epsilon_m^{i_m} = 0$. Consequently it can be said that for all m $\mu_m^{h_m} = \mu_m^0 = 1$, therefore the output unit will give 0, as required.

Now assume that L is a CNF Network. The weights can be assigned in the same manner as before, except rather than considering the rows that correspond to True the negation of the rows corresponding to False are used. If a row a corresponds to True then $\mu_a = 1$, otherwise $\mu_a = Z$ and for any literal present in the row then the input to L which corresponds to the negated literal has weight Z, all other weights are 1.

Claim: For some assignment to the atoms of B, $x_1 = v_1, ..., x_n = v_n$ where $v_i \in \{0, 1\}$. Then $L(i_1, ..., i_{2n}) = B(x_1, ..., x_n)$.

In this configuration it must be shown that every hidden neuron fires when the network is presented with a variable assignment which corresponds to True and there is always at least one neuron which does not fire when the assignment corresponds to False. Assume for a contradiction that for a given assignment $B(x_1, ..., x_n) = 1$ but $L(i_1, ..., i_{2n}) = 0$. Then there is at least one hidden neuron which does not fire. Let h_a be such a neuron. Consequently for any input weight combination which is relevant $\epsilon_{a,m}^{i_m} = 1$, so $i_m = 0$ for any relevant input. Let $i_{r_1}, ..., i_{r_k}$ be the relevant inputs then $i_{r_1} \vee ... \vee i_{r_k} = False$, so $\neg(\neg i_{r_1} \wedge ... \wedge \neg i_{r_k}) = False$, a contradiction as then $B(x_1, ..., x_n)$ would be False.

Now assume for a contradiction $B(x_1, ..., x_n) = 0$ but $L(i_1, ..., i_{2n}) = 1$. Then there exists some h_a with output 1 where it should be 0. Consequently there exists at least one input/weight pair with $\epsilon_{a,m}^{i_m} = 1$ that should be 0. Let $i_{r_1}, ..., i_{r_k}$ be all the relevant inputs, at least one relevant input is present i_r. Consequently $i_{r_1} \vee ... \vee i_{r_k} = True$, therefore $\neg(\neg i_{r_1} \wedge ... \wedge \neg i_{r_k}) = True$, a contradiction as then $B(x_1, ..., x_n)$ is True.

□