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

The growing popularity of Artificial Neural Networks is driving the development of systems that are not only accurate but have a decision making process that can be defended. The report develops a novel neural network architecture which builds in an interpretable structure. Experiments on the MNIST dataset showed these models have statistically equivalent performance to Multi Layer Perceptron Networks with a more interpretable learned model.

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.

1.1 Motivation

The number of situations in which ANN systems are used is growing. This has lead to an increasing interest in systems which are not only accurate but also provide a means to understand the logic use to derive their answers [5]. 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 things 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 [5].

In the context of safety a Machine Learning (ML) system often can not 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 being biased towards a protected class of people. A paper published in 2017 demonstrated that standard machine learning algorithms trained over text data learn stereotyped biases [4]. An ANN trained with the intent to be fair could result in a system which discriminates because of biases in the data used to train it.

Another pressure causing the development of interpretable ML systems is changing laws. In 2018 the European Union (EU) General Data Protection Regulation (GDPR or "right to explanation") will come into affect. The GDPR will require algorithms which profile users based on their personal information be able to provide "Meaningful information about the logic involved" [16]. This law would effect a number of situations where ML systems are used. For example banks, which use ML systems to make loan application decisions [7].

Using a number of simulated data researchers trained an ANN to compute the probabil-

ity of loan repayment [7]. The simulated data consisted of white and non-white individuals, both groups with the same probability of repayment. As the proportion of white to non-white individuals in the data set increased the ANN became less likely to grant loans to non-white individuals. This is a statistical phenomenon called uncertainty bias. 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 and breaches of the regulation will incur fines [7].

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 [12].

Existing methods for interpreting ANNs are post-hoc algorithms to extract some meaning from standard ANN's. The approach presented in this report builds intepretability into the ANN through a structure based off logical functions. This report presents 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 very specific 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 data base and compared against standard Multi-Layer Perceptron Networks. It will be demonstrated that LNNs make interpretation a simpler task without sacrificing performance.

Chapter 2

Background

This chapter presents concepts related to the study conducted in this report. After each idea is introduced and discussed the relation to this work is explored.

2.1 Intepretability

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. Intepretability of Machine Learning systems has been defined as *"the ability to explain or to present in understandable terms"* [5] which is ambiguous.

When is an interpretable model necessary? [5] In some problem domains there is no risk associated with incorrect answers, in this situation there may be less interest a model which can be interpreted.

Problem domains where there is a high risk associated with incorrect answers include Safety and Ethics. A survey, "Concrete Problems in AI Safety" [1] 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 a known environment containing no mess, how can the robot be prevented from disabling its sensors which it uses to detect the mess?

It has been demonstrated that Machine Learning systems learn human like biases from textual data [4]. A study done in 2010 [19] developed a method to analyse datasets for biases against particular classes of people. Analysis was conducted on the *German Credit Dataset* which showed that discriminatory decisions where 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 in a scientific way? There are three categories of evaluation techniques [5].

The titles are from another paper but descriptions are in my own words. Improve description here

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

2. *"Human-Grounded Metrics"* Designing simpler experiments which still allow for establishing intepretability in a specific domain. This situation occurs when access to domain experts is either to 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 intepretability in terms of the problem then it can be used to establish this property in the model.

2.1.1 Relation to Solution

The report is presenting an interpretable model. Consequently it is important to understand what it means for a model to be interpretable and how this property can be evaluated in the solution presented.

2.2 Rule Extraction

A survey in 1995 focuses on rule extraction algorithms [2], identifying the reasons for needing these algorithms along with introducing ways to categorise and compare them.

There are three categories that rule extraction algorithms fall into [2]. An algorithm in the **decompositional** category 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, **electic**, is a combination of decompositional and pedagogical. An Electic algorithm inspects the hidden/output neurons individually but extracts rules which represent the ANN globally [21].

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 the extracted rules, these are accuracy, fidelity, consistency, comprehensibility [2].

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

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

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.

2.2.1 Relation to Solution

These survayes provide a framework for evaluating the rules extracted using a particular technequic. The solution developed in this report allows for rule exaction in some situations. By introducing this content the reader is familiar with this approach to evaluating extracted rule sets.

Rule extraction algorithms make up a large portion of the current methods to interpret knowledge inside an ANN.

2.3 LIME: Local Interpretable Model-Agnostic Explanations

Prehapse add more here?

The paper "'Why should I Trust You?' Explaining the Predictions of Any Classifier" [17] published in 2016 presents a novel approach to 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 technequic provides an explanation of a single prediction. Trust in the model can be developed by inspecting explanations of many individual predictions.

2.3.1 Relation to Solution

LIME is a recent and effect method for interpreting models. Consequently it will provide an effective benchmark for comparing the solution presented in this report.

2.4 Noisy Neurons

Multi-Layer Perceptron Networks (MLPNs) are universal function approximators [10], as such can achieve a high accuracy across a broad range of problems. There are many equivalent weight representations of an MLPN which give the same solutions, this makes interpreting the network difficult [12]. By restricting the possible relationships between a neurons inputs and outputs to be a single function then only the relevance of each feature needs to be inferred. Consequently the problem interpretation becomes easier. The two functions OR and AND are easy to understand, which is a good reason to pick them over other functions.

In 2016 the concept of Noisy-OR and Noisy-AND neurons [12] where derived from the Noisy-OR relation [20], 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 [15].

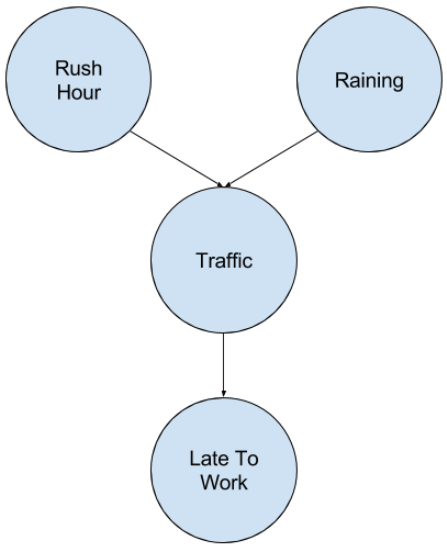


Figure 2.1

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 you are late to work, and it being rush hour or raining influences whether there is traffic.

Consider a Bayesian Network with a node D and $S_1, ..., S_n$ being Ds parents. In other words S_i influences the node D. Each S_i is independent from 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 uncertantly as to whether each parent influences the child. Let ϵ_i be the uncertainty that S_i influence D. If $\epsilon_i = 0$ then parent i influences the child, otherwise if $\epsilon_i = 1$ it does not. Therefore $P(D = 1|S_1 = 1, , S_n = 1)$ can be defined as Equation 2.1. Given all the parents are 1, if the child is influenced by at least one of its parents then $\prod_{i=1}^{n} \epsilon_i = 0$ and it is on. On the other hand if it is not influenced by any of its parents then $\prod_{i=1}^{n} \epsilon_i = 1$ so it is off.

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

Equation 2.1 shows the noisy or relation. In the context of a neuron the inputs $x_1, ..., x_n$ represent the probability that inputs $1, ..., n$ are true. The output of a neuron as conditionally dependent on its inputs, in terms of a Bayesian Network the x_i's is a parent of the neuron. Each ϵ_i is the uncertainty as to whether x_i influences the neurons output. How can weights and inputs be combined to create a final activation value for the neuron? First consider a function $f(\epsilon, x)$ which computes the irrelevance of input x. Some conditions [12] that can be placed on f are given in the following list.

1. $\epsilon = 1$ means that $f(\epsilon, x) = 1$
2. $x = 1$ means that $f(\epsilon, x) = 1$
3. Monotonically increasing in ϵ and decreasing in x. Let $f(x, \epsilon) = \epsilon^x$. The definitions for Noisy-OR and Noisy-AND gates can now be given.

The function $f(\epsilon, x)$ is 1 (i.e x is irrelevant) if x does not influence the output ($\epsilon = 1$) essentially cancelling these irrelevant inputs out in a AND or OR. The noisy activations can therefore be a logical function over $f(\epsilon_{x_i}, x_i)$ for all i.

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.

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

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

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

Both these parametrisations reduce to discrete logic gates when there is no noise, i.e. $\epsilon_i = 0$ for all i.

2.4.1 Relation to Solution

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, an atom is one of the variables in the formula.

Consider the boolean formula $\neg a \vee (b \wedge c)$, 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 $\neg a \wedge (b \vee c)$, 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 exactly 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 are at most 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 [8] (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 is able to achieve a low enough accuracy then

rules can be extracted from the network in terms of a Boolean CNF or DNF expression [8].

The algorithm which extracts rules from LNFNs would be Electic and certainly 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 difficult to reproduce.

2.5.4 Relation to Solution

The motivation for the development of the solution presented in this report (Logical Neural Networks) can be derived as a more flexable version of Logical Normal Form Networks.

2.6 Logical Neural Networks

ANN's containing of Noisy-OR and Noisy-AND neurons are called Logical Neural Networks [12] (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 to not yield interpretable models and also have lower performance, consequently when LNNs are refered to it will always be in the context of Pure LNNs.

2.6.1 Relation to Solution

The solution presented is a modified Logical Neural Network structure.

Chapter 3

Foundation of Logical Normal Form Networks

This chapter motivates the concept of Logical Normal Form Networks (LNFNs) along with re-deriving them in terms of Noisy Neurons. The training of LNFNs is discussed in the form of deriving a weight initialization method, choosing a 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 inputs. Consider some problem p 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 simply the function f with the smallest cross entropy loss.

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

How might a interpretable network architecture that can learn these functions be constructed? The following facts will be helpful, any boolean function has a unique Conjunctive and Disjunctive Normal Form (CNF & DNF), both the CNF and DNF are described by the boolean operations NOT, OR and AND, in a CNF or DNF a not can only occur on a literal and the maximum number of clauses in a CNF or DNF is 2^n where n is th number of inputs.

One option is to use a standard Multi-Layer Perceptron Network (MLPN). MLPNs have been shown to be universal function approximators [10] but are diffcult to intepret. Learning the CNF or DNF of the optimal function is an equivalent problem. For now consider the problem of learning the CNF. This can be done with hidden layer of size k and an output layer with a single neuron. The hidden neurons only need to perform the OR operation on a subset of inputs. The output layer only need perform an AND of a subset of the hidden neurons.

Using Noisy-OR and Noisy-AND (See Section 2.4) such a network can be constructed. Noisy neurons can not compute the not of inputs so the input layer must be modified to include the negation of each input. Figure 3.1 is the structure that has been derived.

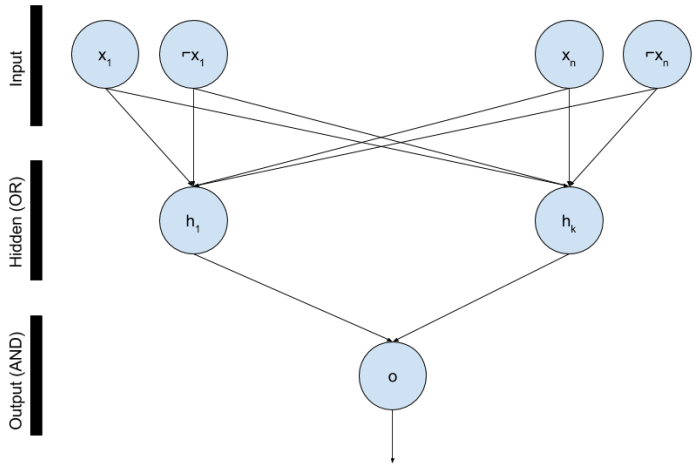


Figure 3.1: Network Architecture for Learning CNF

By the same logic a network architecture for learning the DNF can be derived, the hidden layer consists of ANDs and the output of a single OR. These networks for learning CNF and DNF formulae are a new derivation of Logical Normal Form Networks (LNFNs) [8], the difference being Noisy neurons are used instead of the previously derived Conjunctive and Disjunctive neurons. Definitions of CNF Networks, DNF Networks and LNFNs are now given.

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. A **LNF-Network** is a DNF or CNF Network

It must also be determined how many hidden units the LNFN will have, it is known that 2^n, n being the number of atoms, is an upper bound on the number of clauses needed in a CNF and DNF formula (see Theorem 2.5.1).

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.

Proof. 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

$$\lim_{Z\to 0} Out(o) = 1 - \prod_{m=1}^{2^n} \mu_m^{h_m} \tag{3.2}$$

$$= 1 - 0 = 1 \tag{3.3}$$

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.

□

Theorem 3.0.1 provides justification for using 2^n hidden units, it guarantees that there at least exists an assignment of weights yielding a network that can correctly classify each item in the truth table.

3.1 Noisy Gate Parametrisation

The parametrisation of Noisy gates require weight clipping, an expensive operation. A new parametrisation 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.

Now these weights must be substituted into the Noisy activation. Consider the Noisy-OR activation.

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

From a similar derivation we get the activation for a Noisy-AND.

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

Concisely giving equations 3.4, 3.5

$$a_{and}(X) = e^{-(W^{'} \cdot (1-X) + b^{'})} \tag{3.4}$$

$$a_{or}(X) = 1 - e^{-(W^{'} \cdot X + b^{'})} \tag{3.5}$$

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 3.5 and 3.4 for the Noisy-OR, Noisy-AND activations retrospectively allows LNFNs to be trained without the need to clip the weights.

Before these networks can be trained a number of 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 initilization of weights isnt carefully thought about then as the number of neurons increases the learning conditions deteriate [6].
2. *Training Algorithm:* There exsists a number of different algorythms for propergating gradients [18]. 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

Here a weight initlization method is derived specifically for networks using Noisy activation functions.

Deriving a Distribution For The Weights Before a weight initialization algorithm can be derived good learning conditions must be identified. 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 this fact it can be argued that the weight initialization is the same for both Noisy-OR and Noisy-AND. Recall the activations for Noisy neurons

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

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.

Strictly speaking each $w_i = \log(1+e^{w_i'})$ (as derived in Section 3.1) however for the purposes of this initialisation derivation it will be assumed that $w_i \in (0\infty]$.

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 re written as $z = -\log(y)$.

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

Therefore $F(z) = 1 - e^{-\lambda z}$ where $\lambda = 1$. 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 $var(w_i)$.

$$
\begin{aligned}
E[z] &= E[w_1x_1 + \cdots + w_nx_n] \\
&= E[w_1x_1] + \cdots + E[w_nx_n] \ (independence) \\
&= E[w_1]E[x_1] + \cdots + E[w_n]E[x_n] \\
&= n \cdot E[w_i]E[x_i] \ (i.i.d) \\
&= n \cdot E[w_i] \cdot \frac{1}{2} \\
1 &= \frac{n}{2} \cdot E[w_i] \\
E[w_i] &= \frac{2}{n}
\end{aligned}
$$

$$
\begin{aligned}
var(z) &= var(w_1x_1 + \cdots + w_nx_n) \\
&= var(w_1x_1) + \cdots + car(w_nx_n)
\end{aligned}
$$

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

Consequently the variance can be found by the following

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

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 that weights can be sampled from. Based on our initial assumptions this distribution must also generate values in the interval $[0, \infty]$. There exists a number of different distributions which satisfy this constraint, for example Beta Prime, Log Normal, Poison. Based on experimental evidence the Log Normal distribution had the best results so for brevity only its derivation is included.

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

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

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

$$\sigma^2 = \log \frac{3n+4}{4} \tag{3.6}$$

$$\mu = \frac{1}{2} \cdot \log \frac{16}{n^2(3n+4)} \tag{3.7}$$

Weight Initialization Algorithm Through experiments the weights sampled from a Log Normal distribution perform better than when sampled from a Beta Prime distribution. The algorithm for weight initialization can now be given but first consider that each weight that is sampled from the Log Normal distribution has the following property $w \sim LN$, $w = f(w')$ where w' can be any real value, consequently to obtain the initial weights each w must be inversely transformed back to the space of all real numbers.

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

Figure 3.2: Weight Initialization Algorithm for LNNs

3.2.2 Training Algorithm

The ADAM Optimizer [11] 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

Through experimentation it was observed that LNFNs respond well to a mini batch size of 1. The question remains as to why higher batch sizes dont work

3.2.4 Preliminary Results

Preliminary testing showed that LNFN's are able to learn good classifiers on boolean gates, i.e. NOT, AND, NOR, NAND, XOR and Implies. It is also possible to inspect the trained weights and see that the networks have learnt the correct CNF or DNF representation.

3.3 LNF Network Performance

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 3.3 shows a comparison between all 4 of the networks and Figure 3.4 shows just the LNFN's.

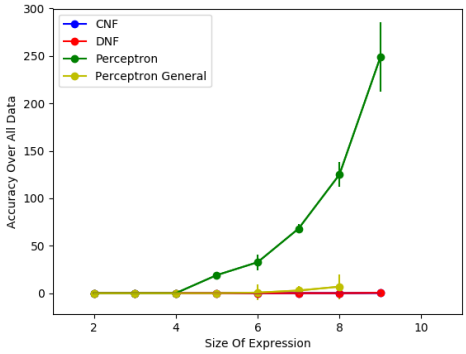


Figure 3.3

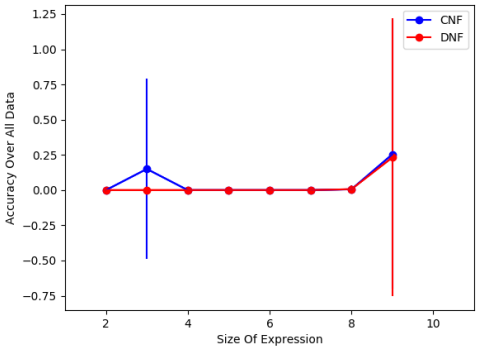


Figure 3.4

3.4 LNF Network Generalization

These networks are able to 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

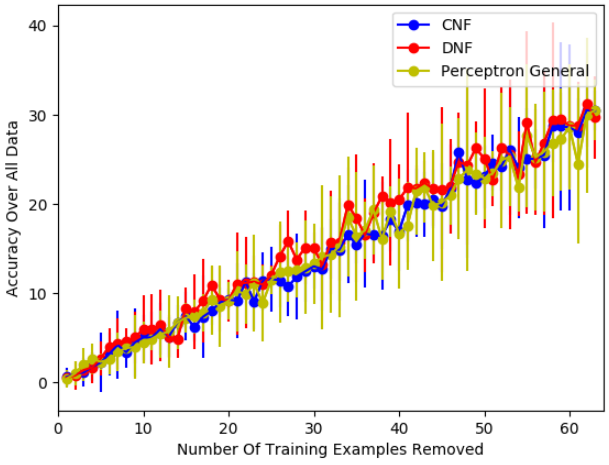


Figure 3.5

Figure 3.5 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 neurons 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 is the foundation of the following rule extraction algorithm, it was derived from experimental evidence by training LNFNs over complete truth tables and inspecting the weights. 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 ϵ_i approaches 0 or 1.

The Algorithm displayed in figure 3.6 extracts rules from CNFNs. It takes the output weights (ow) and hidden weights (hw) as input and outputs the 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.

```
1  atoms = {x_1, ¬x_1, ...x_n, ¬x_n, }
2
3  function extractRulesCNFN(ow, hw)
4    ow = σ(ow)
5    hw = σ(hw)
6    relvHidden = [hw[i] where ow[i] := 0]
7
8    and = And([])
9      for weights in relvHidden
10       or = Or([atoms[i] where weights[i] := 0])
11       and.add(or)
12
13   return and
```

Figure 3.6: Rule Extraction Algorithm (for CNFN)

In practice many clauses in the extracted expression contain redundant terms, such as clauses that are a tautology or a duplicate of another. Filtering these out is not an expensive operation.

Section 3.4 discusses the generalization capabilities of LNFNs compared to MLPNs and shows that they are statistically equivalent. How does training over incomplete truth tables effect 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 learnt, 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.

When constructing a CNF or DNF from a truth table (See Section 2.5.2) in the case of CNF only the rows corresponding to false are considered and for the DNF only rows corresponding to true. Despite the fact that if m rows are removed from the training set then there are 2^m possible functions that could represent the partial truth table. Does this same principle apply to the rules extracted from CNF or DNF networks?

Figure 6.41 shows how the rule set of an CNFN generalizes as examples are removed. Figure 3.8 shows the same but for DNFNs.

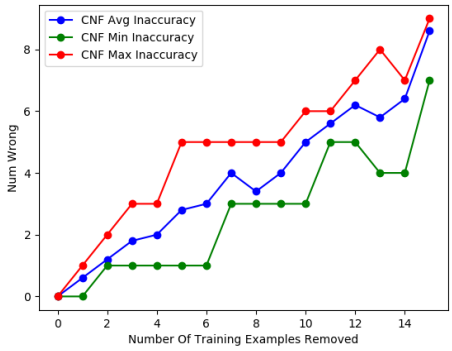


Figure 3.7

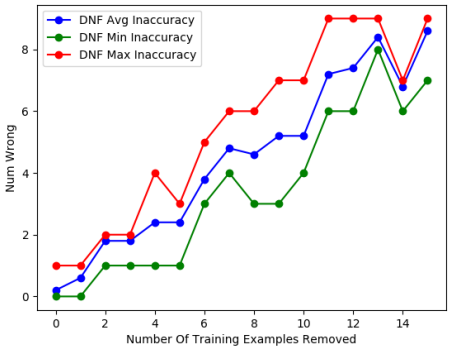


Figure 3.8

In Figures 6.41 & 3.8 the training examples which are removed get randomly selected. How is the performance effected if the removed examples are chosen more carefully? In the next experiment only examples corresponding to false are removed and the resultant training set is given to a DNFN.

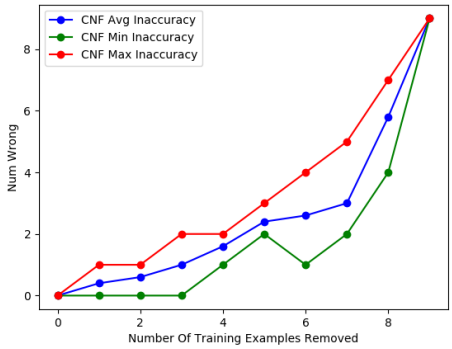


Figure 3.9

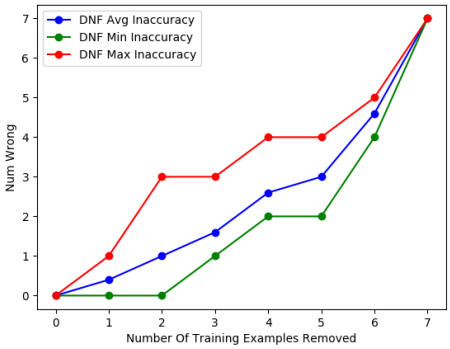


Figure 3.10

Figures 3.9 & 3.10 shown CNFNs and DNFNs trainined over partial data retrospectively. In the case of CNFNs only etries corosponding 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 peform better.

Chapter 4

Expanding the Problem Domain of Logical Normal Form Networks

The LNFNs presented in Chapter 3 have only been shown to be appiliciple when learning boolean truth tables. This chapter investigates extending LNFNs to support multi class classification problems and continous 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 networks 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.

The Lenses problem [14] is multi class and contains a natural rule system making it an ideal problem for evaluating LNFNs. 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 width are $2 \cdot n$ and n retrospectively.

The performance of the two classifiers will be compared using Leave-One-Out Cross-Validation. The structure of the MLPN differs in the number of hidden layers/units, there are two hidden layers, one with $2 \cdot n$ hidden units and the other with n

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

Table 4.1: Network Peformance 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 $f = False$ or $True$ retrospectively. Giving a description of the other atoms is not beneficial as they refer to medical terms which are unimportant.

The following rules have been extracted from a CNFN after being trained over the complete Lenses data set. Any duplicate clause or Tautology has been filtered out, the resultant extracted formula has also been manually simplified (so they can be displayed and understood better).

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

Immediately it is possible to find useful information about this problem that was not obvious before. For example $\neg f = True \implies$ Class 3, or *if the tear reduction rate is reduced then do not fit contact lenses*.

The DNFN might be more applicable as the rules will be more insightful. Each clause in the formulas represents a situation where the class is true. The formulae extracted from the DNFN confirm the previous knowledge extracted, $\neg f = True \implies$ Class 3.

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

Table 4.2 demontres the peformance of extracted rules over the Lenses dataset. The confidence intervals of networks and rule sets overlap so there is no stistically significant difference 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

4.2 Features with Continuous Domains

The input features to an LNFN are probabilities, as such they are allowed to be continuous but must be in the range $[0, 1]$. Consider attempting to learn the Iris data set [14] with an LNFN. One option would be to normalize all the variables to force them into the expected range $[0, 1]$. The values of all the features after normalization are compatible with an LNFN, but consider the meaning, can these normalized features do not represent probabilities.

It is possible to think of these normalised features as the probability that the true value is the upper bound of the original range. For example if the features range from 0 to 2, and one example has a feature value of 1, when normalised it becomes $\frac{1}{2}$ or a 50% probability of being a 2. For a moment forget about trying to find meaning in these normalised features. What happens if an LNFN is trained on the Iris problem? Table 4.3 gives the results of training the normalised Iris problem on LNF networks. The results show that while the peformance can very alot it is possible for both the CNF and DNF networks to obtain 100% accuracy.

	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: tab:iris-network-peformance-comp

Given that the input features to the network are continous it is not possible to extract boolean rules. It is possible to find the most important features in each classification from inspecting the weights. If attempting to intepret these models meaning must be assigned to each of the normalised features. Consider an abatary (un-normalised) feature f. Assume $f \in [f_min, f_max]$ then the larger the normalised 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 small.

The list below describes the attributes of each class on an CNF network which obtained an error rate of 0% on the testing set. These discovered rules reveal intesting information about the dataset. Having a small petal width and length mean the class is more likely to be *Iris Setosa*. Alternatively having a larget petal width means the instance is more likely to be *Iris Virginica*.

Finally the *Iris Versicolour* rule is more complicated and seemes contridictory. If the instance has a small petal width and length but also has a large petal width or length.

1. *Iris Setosa:* Smaller **petal width** AND Smaller **petal length**
2. *Iris Versicolour:* Smaller **petal width** AND Smaller **petal length**. AND (Larget **petal width** OR Larger **petal length**)
3. *Iris Virginica:* Larger **petal width**

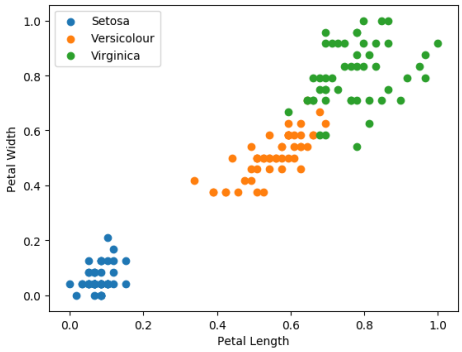


Figure 4.1: Scatter plot of Iris dataset, showing the petal length on the x-axis and petal width on the y-axis

Figure 4.1 clearly demonstrats the rules for the two classes, *Iris Setosa* and *Iris Virginica*. The Figure also gives insite as to the meaning of the class rule for *Iris Versicolour*, these instances are generally centered around the middle with a trend of increasing length and width. The two features **sepal width** and **sepal length** are used in the classification algorythm but have a very small weighting which is why they have been omitted.

4.3 Discussion

This chapter has demonstraited 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 descrete input features.

In the context of the Lenses problem it was possible to extract boolean rules from the LNFNs which had a stistically equivelent peformance to the networks. These rules also provided insite into the problem which was not obvious 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 usful information from the trained models which highlighted the most important features when making classifications.

Chapter 5

Logical Neural Networks

This chapter discusses development of Logical Neural Networks as a generalization of LNFNs. There are two key issues with LNFNs. Firstly the number of hidden units becomes infeasible as the amount of inputs 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 memorise the input data. Using what has been learnt 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 [12]. An LNN, consisting of Noisy-OR hidden units and Noisy-AND outputs, was shown to perform worse than an MLPN. There are two key issues caused by removing the restrictions imposed by the LNFN definition (Definition 3.0.3) which must be addressed

1. Noisy neurons do not have the capacity to consider the presence of the negation of an input. This 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 can not 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 learnt.

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 simply augment the inputs to each layer appending so it receives the input and its negation. Another is to derive a parametrisation 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.

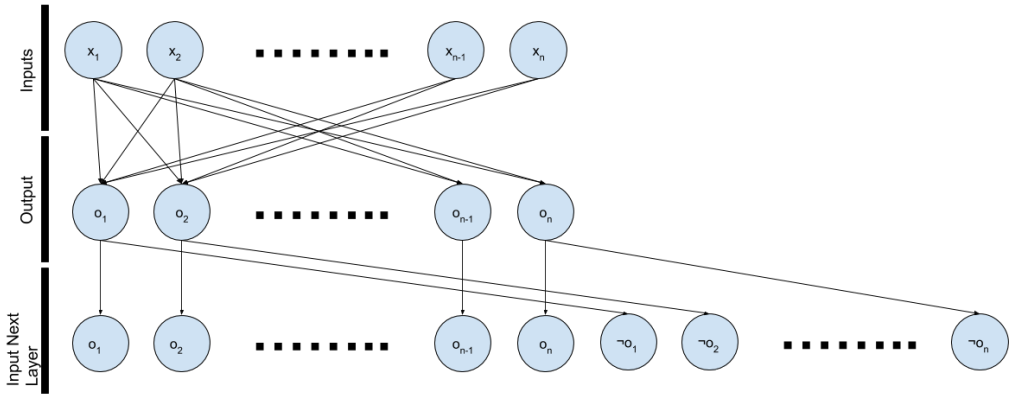


Figure 5.1

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 multiclass classification to generate a probability distrubution 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 function on probabilities, insted of highilight 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 classes 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 intepretability. 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 intepretability are explored. In Chapter 2 different methods for evaluating the intepretability of models where presented. The method used in this report is *"Application-Grounded Evaluation"* [5] which relies on domain experts to assess the intepretability of a model. The two problems MNIST [13] and Tic-Tac-Toe [14] will be used in this evaluation.

These problems are a sutable choice as their simplisity 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 peformed on a daily bais by most humans.

There are **5** 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 intepretability between MLPNs and new/old LNNs
5. Comparison of intepretability between MLPNs (using LIME) and new/old LNNs

It was conjectured that AND neurons where not effective for low level feature extraction [12]. Throughout this evaluation, AND neurons will be used in the first hidden layer to determine whether this conjecture holds.

6.1 Performance of Logical Neural Networks

To evaluate the performance of Logical Neural Networks (LNNs) a number of different configurations will be trained on the MNIST dataset. Each experement will be trained for 30 epochs. Any experements with an LNN archetchure will be trained with and without an Logical Softmax

1. **(OR $\rightarrow$ 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. **(AND → OR) Architecture:** 30 hidden AND neurons
4. **(OR → AND → AND) Architecture:** 60 hidden OR neurons, 30 hidden AND neurons
5. **(OR) Architecture:**
6. **(AND) Architecture:**
7. **(AND) Old Architecture:**

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 retrespectively. 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)
(AND → OR)	0.098	(0.073, 0.141)	?	?
(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 experemental results lead to the folowing conclusions

1. *New LNN Architecture Gives Better Performance Than the Old:* The confidence intervals for test set performance do not overlap for Archetcures **(OR → AND) Old** and **(OR → AND)** (without LSM). This can also be observed from Archetchures **(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 Strucure or LSM:* The experements show that both the new archetchure and LSM give a stistically significant improvement in peformance. 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 archetchure does not introduce an increase in peformance. However the oposite is true when comparing the (AND) and (AND) Old networks.

6.2 Intepretability of Logical Neural Networks

There are two cases of intepretability. In the situation where input and outputs are descrete then intepretablity becomes Rule Extraction. The other situation is where the inputs are continous.

6.2.1 Discrete Case (Rule Extraction)

To assess the rule extraction of LNNs the Tic Tac Toe [14] problem will be the becnchmark.

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) is able to achieve an error rates displayed in Table 6.3. Each experiment is averaged over 30 runs

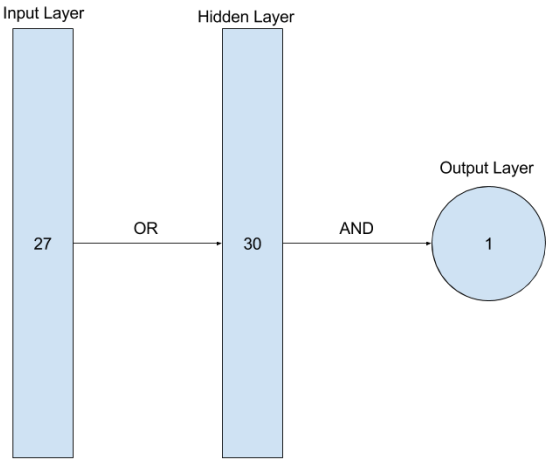


Figure 6.1: Network archetchure for learning Tic-Tac-Toe

	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 insted ANDs are used as the hidden nerons and ORs for the outputs then the LNN is able to achieve stistically equivelent peformance to the archetchure in Figure 6.1. Table 6.4 shows the results from these experements.

	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 strong 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.

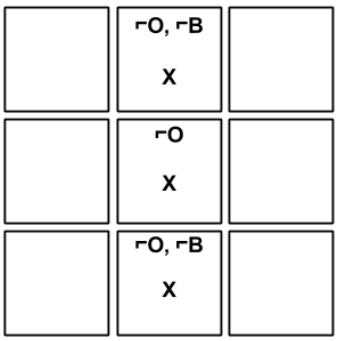


Figure 6.2: A pictorial example of a rule extracted from the AND OR network

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

The rule extraction algorithm given in Figure 3.6 is an electic algorithm as this category describes algorithms that examine the network at the neuron level but extract rules which represent the network as a whole [21]. 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, Fedelity, Consistency and Comprehensibility [2].

1. **Accurate:** As demonstrated experimentally the extracted rules are able to generalise well to unseen examples.
2. **Fedelity:** 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 a number of different initial conditions.
4. **Comprehensibility:** The upper limit of the size of the rules is related to the number of layers and neurons. The OR AND model obtains a complex set of rules, with a total 35 of the clauses. The AND OR model only utilizes 11 clauses. The 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. Where as in the OR AND model each clause must true.

6.2.2 Continuous Case

In the continuous case extracting boolean rules is not possible as each input no longer represents a boolean but rather a degree of truth. In terms of MNIST, being able to construct an image representing how relevant each input feature is to each of the possible output neurons would allow visual verification that the network has learnt something meaninful.

The weights from one layer to another directly correspond to how much influence the corresponding input has. Consquently determining the influence that the inputs to a layer have on the outputs of the layer is not a diffcult task. For example trying to compute the influence each x_i has on the output of some h_j. However computing the influence that a neuron has across layers is not so simple. This is the case of asking how x_i influences the output of o_j.

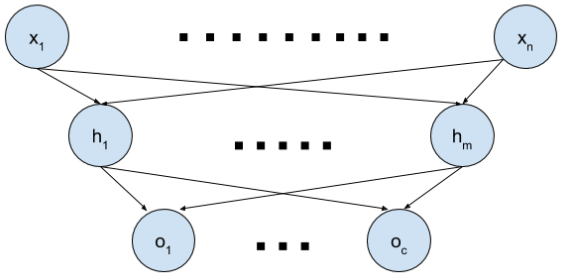


Figure 6.3: Example Network

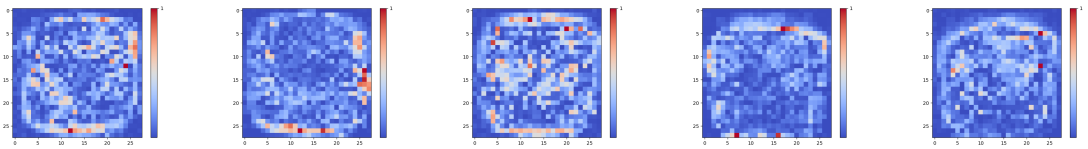
Consider this problem of trying to determine how each x_i effects each o_j and why it might be diffcult. In essence it equivalent to trying to find $\epsilon_i^{'}$ in Equation 6.1. Note that exponent on each ϵ refers to the neuron it belongs to.

$$(\epsilon_i^{'})^{x_i} = \prod_{k=0}^{m} (\epsilon_m^{o_j})^{\prod_{b=0}^{n} (\epsilon_b^{h_k})^{x_b}} \tag{6.1}$$

This new ϵ is dependent on all the other x_i's, not just the one of interest. Consequently it is only possible to directly compute the influence of the input features with respect to the output from first layer of hidden neurons. Intepreting Multi Layer LNNs insted will be achieved by displaying the most relevent hidden features for a specific classification. Intepretability will be assessed by comparing various MNIST models using the new architecture again each other but also the original LNN model. Part of this assessment will be to determine what effects the LSM has on the model interpretation.

No Hidden Layer Networks

Sigmoid Network In the depictions of weights from a sigmoid network the blue represents a negative value and the red represents are positive weights. The features do not appear to represent any intepretable features of the digits.



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 6.20 represent how relvent each input feature is towards a possitive classification of the digits 0, 2, 4, 7 and 9. The top and bottom rows represents the relevences for an AND archetchure with and without LSM retrespectively. The darker pixles represent more relevent features, where white is completly irrelevent.

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 border to achieve its classifications. Both models are intepretable as it is possible to understand the logic used in their decision making. Each of the models describe the probelm in a different way, uncovering different 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 the digits which they classify. However the model without an LSM has a sparser representation so the model is depnedent on less information.

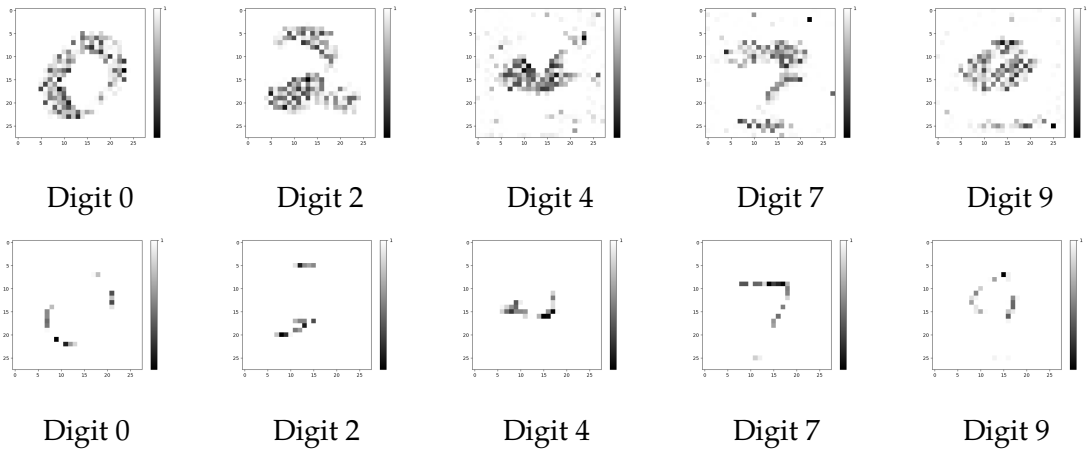


Figure 6.20: Figures representing the weighted connections between the inputs and outputs in an AND network with the old archetchure. Top row is with an LSM and bottom row is with out an LSM

AND Network (With LSM) The model here is an AND network running on the new archtchure (which considers the NOTs of inputs) with an LSM. The new archetchure 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 6.31 shows the how relevent 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 centre 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.

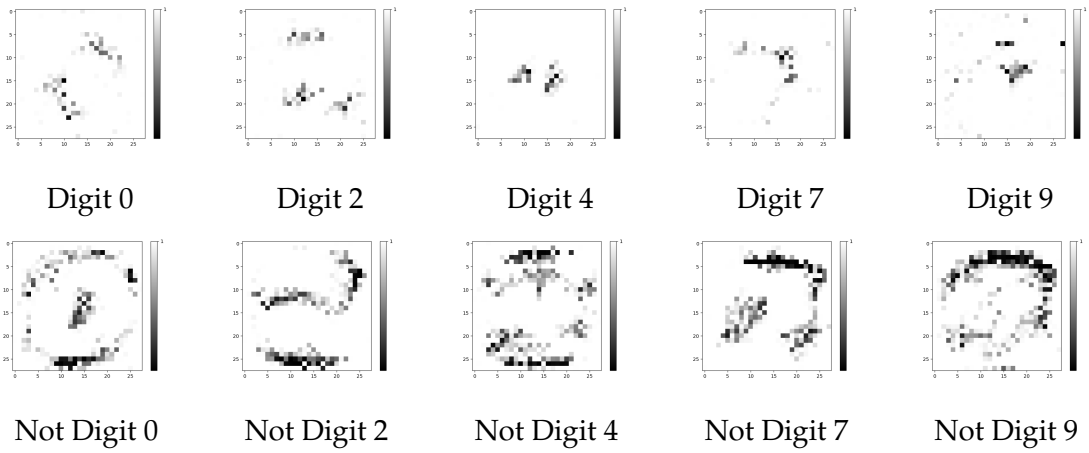
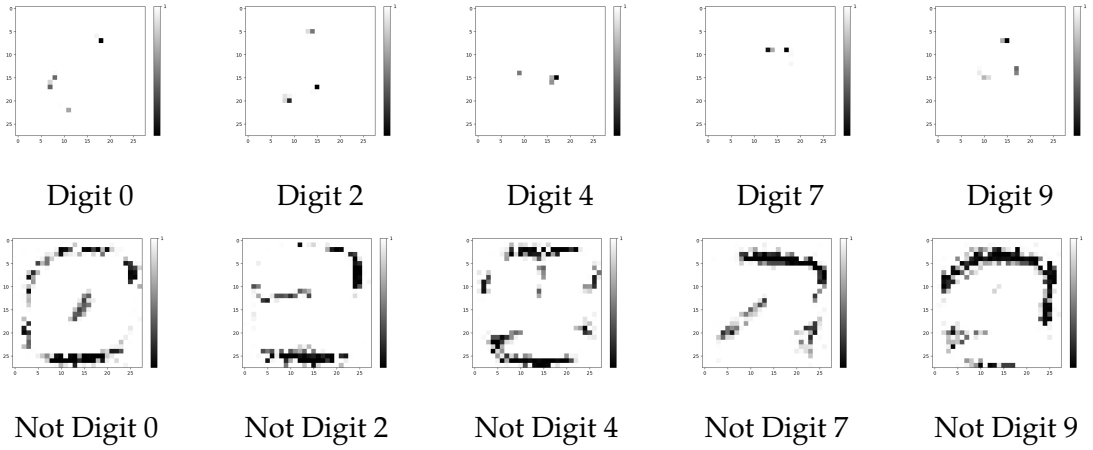


Figure 6.31: Figures representing the weighted connections between the inputs and outputs for an AND network with an LSM. The top and bottom rows represent the positive/negative contributions retrespectively.

AND Network (With out 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 6.31) however the representations here are less noisy and have harder boundaries.



Conclusion of Intepretability for LNNs with No Hidden Layer The logical Softmax introduces more noise into the weights but does not diminish the intepretability 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 weight 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.

Compared to the Sigmoid network the LNNs are more interpretable as it is possible to make determinations about how the network is making its classification decisions. These experements provide strong evidence in favour of the conclusion that LNNs with no hidden layers using new LNN architecture and LSM improve the intepretability when compared to MLPNs and old structure LNNs.

Single Hidden Layer Networks

Interpreting these multilayer models is a more difficult task as the hidden layer introduces dependencies between the inputs when considering how they influence the classification. To test the intepretability of these networks assume the goal is to verify that the digit 1 is being classified in a sensible way. With each network the hidden neurons which have the most influence on the classification be 1 will be displayed along with the ones which influence the classification not being 1.

Sigmoid Network The hidden features learnt by the Sigmoid Network do not appear to represent any meaningful information about the digit 1.

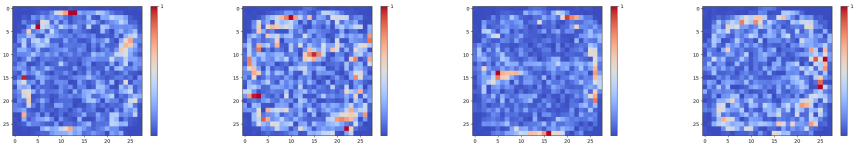


Figure 6.42

OR → AND Network Old Structure (With Out LSM) It is not immedatily apparent how the features in Figure 6.43 make up the digit 1. Each hidden feature is an OR of its inputs, as such only one of the dark pixles needs to be active for the neuron to be on. The first and last could be the stem of the 1 where as the middle one contains a dark bar at the bottom which could be the base.

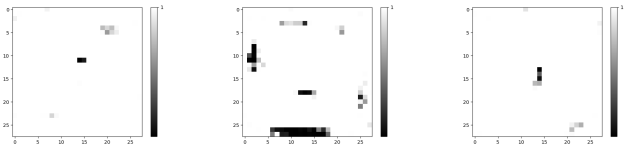


Figure 6.43: Features positively contributing to classification as a 1 for the OR → AND Network

OR → AND Network Old Structure (With LSM) In a similar fashion of the OR AND Network without an LSM these features are not clearly representative of a 1. Interestingly these weighting maps show that the features generally do not place a lot of emphasis on any particular inputs.

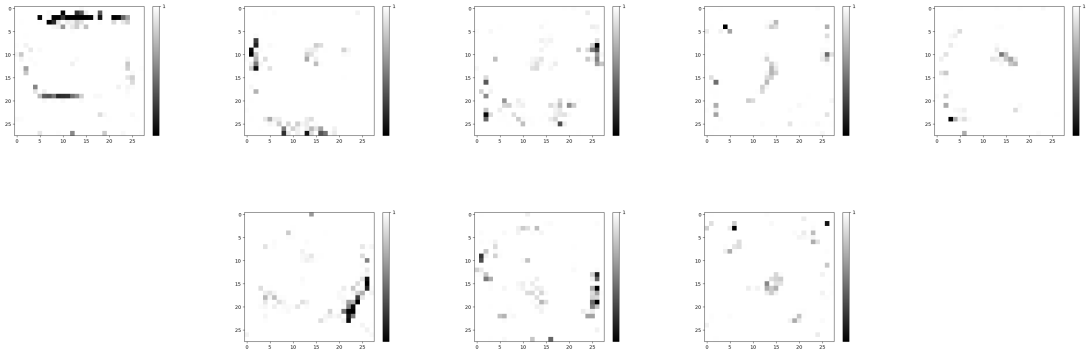


Figure 6.44: Features positively contributing to the classification of 1 for the OR → AND Network Old Structure (With LSM)

OR → AND Network (With Out LSM) This network is running on the new architecture and as such each feature has two ”channels”. One representing the inputs that the feature positively likes and the other the inputs which are negatively liked (i.e. neuron is more

active when these inputs are not present). The images in Figure 6.45 represent the hidden features which if present contribute to the classification being the digit 1. These features are sparse and place a high weight on the inputs which they like.

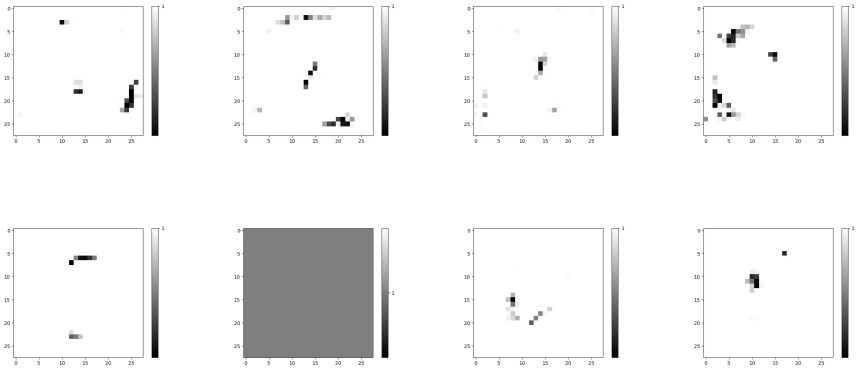


Figure 6.45: Features positively contributing to classification as a 1 for OR $\rightarrow$ AND Network (With Out LSM)

It is not immediatly obvious what the features showin in Figure 6.45 represent. Figure 6.46 shows the single feature which if present then the classification is less likely to be the digit 1. This feature appears to be highley active if the pixles lieing outside an area which looks like the average 1 shape.

While this model is diffcult to intepret it does provide more information than previous OR AND models using the old LNN archetchure.

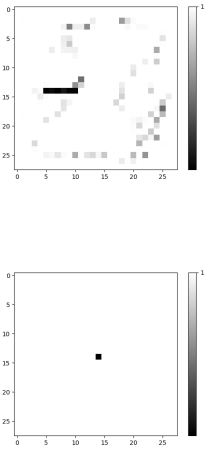


Figure 6.46: Features contributing to classification not being 1 for OR $\rightarrow$ AND Network (With Out LSM)

OR → AND (With LSM) Figure 6.47 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 presense 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 rather pixles that would not usially occour in the digit 1.

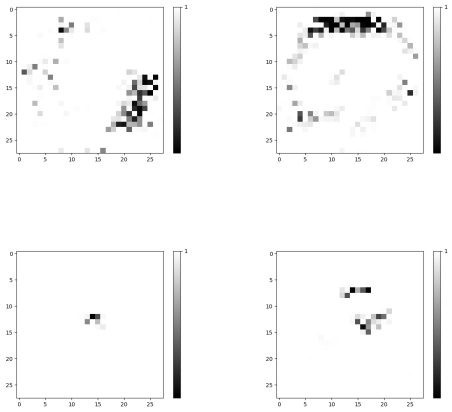


Figure 6.47: Features positively contributing to the classification being 1 for OR → AND (With LSM)

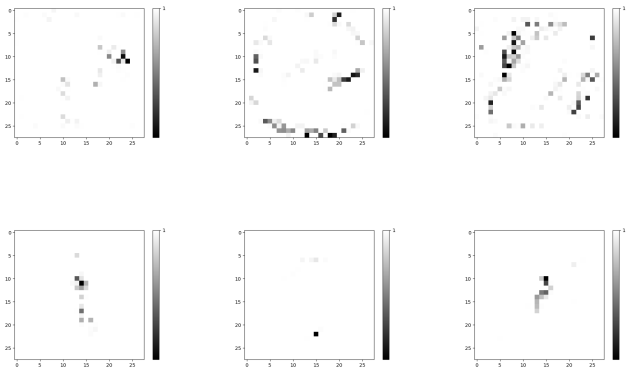


Figure 6.48: Features contributing to the classification not being 1 for OR → AND (With LSM)

AND → OR Model (With Out LSM) Figure **??** shows the single feature which positively contributes to the classification as the digit 1. This feature has learnt to positively identify the stem of a 1 and negatively identify
This network has a small number of features associated with each classification. In terms of the classifying the digit 1 there is only 1 hidden features. This feature appears to classify a 1 by positively liking the stem of a 1 and disliking the anything outside the average 1.

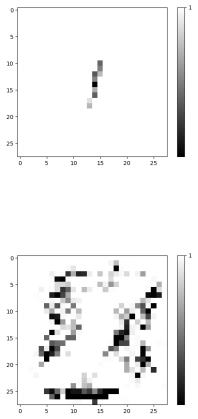


Figure 6.49: Features positively contributing to classification as 1 for AND → OR Model (With Out LSM)

AND → OR Model (With LSM) Similarly to the AND OR Model with out the LSM this model appears to positively like the inputs on the stem of a 1 and dislike any pixel outside the average border of a 1.

Conclusion of Intepretability for LNNs with No Hidden Layer The results of the experiments show that in the context of multi layer adding an LSM introduces more noise into the feature maps. And similarly to before the sigmoid weights do not appear to relate to digit 1 in any meaningful way.

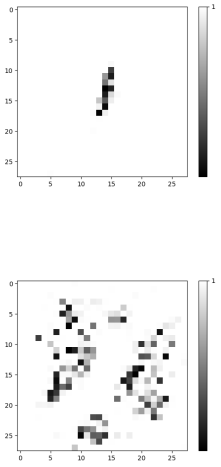


Figure 6.50: Features contributing to classification not being 1 for AND → OR Model (With Out LSM)

6.2.3 Comparason between LNN Intepretability and LIME

The aim of LIME is to build trust in the model by explaining how a specific predictions are arrived at. Whereas LNN intepretability comes down to directly identifying what input features contribute to each output neuron being active. The LIME algorythm will be used to intepret an MLPN and LNN. It is able to determin

6.2.4 Results of Intepretability Experiments

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

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

2. *Adding Nots Improves Intepretability:* 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, rather many have a relatively small relevance.

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

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

6.3 Summary Of Logical Neural Network Evaluation

Through the experiments carried out the following observations can be made.

1. *Performance comparison between the old and new Logical Neural Network Structures:* The new LNN structure (adding NOTs) results with a statistically significant increase in accuracy with some network architectures.

2. *Performance comparison between Multi Layer Perceptron Network and new LNNs:* The new LNN structure achieved statistically equivalent or better performance than the MLPN nets with an equivalent number of hidden neurons/layers.

3. *Performance comparison between networks with and without Logical Softmax:* For any LNN (new structure or old) adding a Logical Softmax improved performance by a statistically significant margin.

4. *Comparison of intepretability between MLPNs and new/old LNNs:* Any LNN (new or old structure) was more interpretable than the corresponding MLPN.

5. *Comparison of intepretability between MLPNs (using LIME) and new/old LNNs:*

These experements have therefor shown that the LNN networks have stistically equivelent peformance to MLPNs. Evidence has also been provided in support of LNNs have a more intepretable model than MLPNs.

These experiments have therefore provided evidence demonstrating that the modified LNN structure and logical softmax improves upon the current state of LNNs.

Chapter 7

Application to Auto Encoders

This chapter demonstraits the flexability of the LNN archecthure by applying them in the context of Auto Encoders.

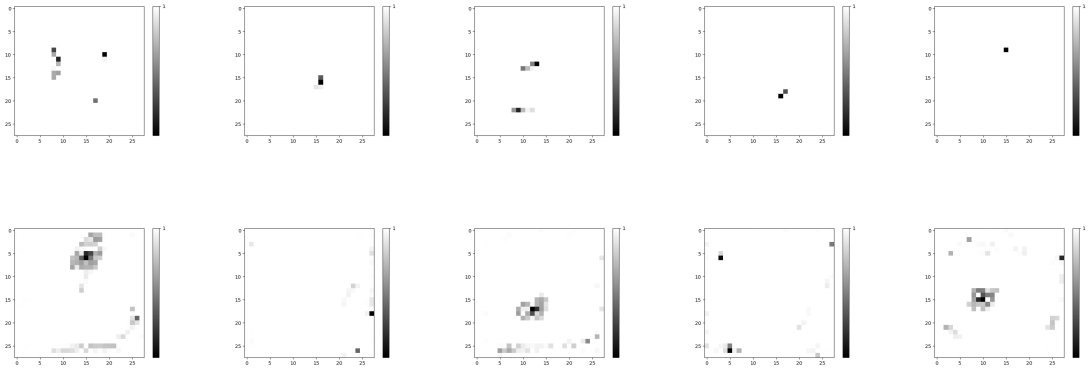
Auto encoders [3] [9] are a network architecture trained with unsupervised learning for a purpose of learning an encoding of the data. An application learning a new encoding is dimensionality reduction. The aim is to take the input to some reduced feature space and then from this feature space back to the original input with the smallest error. The case where there is one linear layer doing the encoding and decoding is called **Linear Autoencoder**, this architecture corresponds to performing Principle Component Analysis (PCA) on the data.

For some data sets, such as MNIST, PCA is not an effective way to reduce the dimensions of the data. For this reason 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

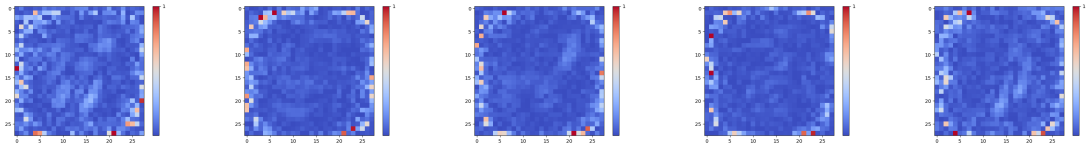
The experiments carried out will be to compress the MNIST feature space (784 dimensions) into 10 dimensions using different Auto encoder architectures. The accuracy and intepretability of the features will be explored. Each model was trained for 30 epochs

Result of Logical Auto Encoder (LoAE) A logical auto encoder, 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.

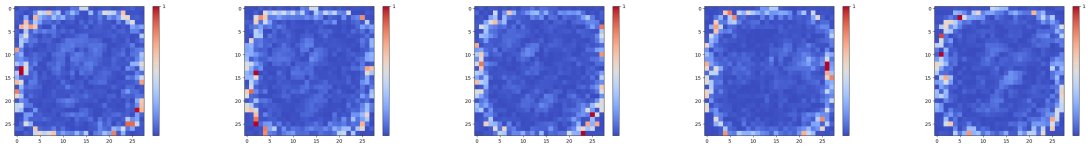


Re Compute images of features

Result of Linear Auto Encoder (LAE) A linear auto encoder obtained an MSE of 21.34 on the training and 21.25 on the test data.



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.



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. The features learnt by the Logical AE are sparse and possible interpretable as each one can be viewed as a logical AND of the input features.

A Logical Auto Encoder provides a means to trade off between the intepretableity of the model and the performance.

Chapter 8

Conclusion

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

A study of Logical Normal Form Networks developed algorythms initilize network weights (leading to good learning conditons) and extract rules from trained models. Through experementation such networks where demonstraited to have stistically equivelent peformance and generalization to Multi-Layer Perceptron Networks. Training LNFNs on the Lenses and Iris data sets demonstraited their ability to learn multi class classification problems and give insite into data by its intepretable trained representaion.

The foundational work developed the tools to derive modifications for the LNN structure giving them stistically equivelent peformance to standard Multi-Layer Perceptron Networks and improving the intepretability of the leant models. Consequently this report has shown that LNNs are a sutable alternative to Multi Layer Perceptron Networks, obtaining a simpler and more intepretable model without scarfising acuracy. By obesrving the weights of LNNs trainined over the MNIST data set it was possible to determin what input features contributed to each classification and verify that the logic learnt was ”sensible”.

Beyond applications to classification tasks Logical Neural Networks where applied to Auto Encoders. While the peformance of Logical Auto Encoders was worse than standard Sigmoid Auto Encoders it demonstraits the flexability of LNNs and the range of situations where they can be applied.

Prehapse the biggest limitation of LNNs comes down to intepreting multi layer logical neural networks. It is possible to directly extract the influence each input feature has on the ouputs in LNNs with no hidden layers. However hidden layers introduce dependencies between input features making it diffcult to find a direct influence of inputs on the outputs. Because of this limitation any future work which can develop better ways to intepret these models would increase the power of LNNs.

Future studys might also focus on establishing the model intepreability in a more scienfic manner. On a larger number of problem domains and individuals who have the domain knowledge to assess how intepretable the model is.

This report has provided a formal foundation for Logical Neural Networks and shown

their ability to not only learn accurate but also intepretable models. While limitations have been identified this does not deminish potential of LNNs but rather provides avenues to increase their application.

Bibliography

[1] AMODEI, D., OLAH, C., STEINHARDT, J., CHRISTIANO, P., SCHULMAN, J., AND MANÉ, D. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565* (2016).

[2] ANDREWS, R., DIEDERICH, J., AND TICKLE, A. B. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems 8*, 6 (1995), 373–389.

[3] BALDI, P., AND LU, Z. Complex-valued autoencoders. *Neural Networks 33* (2012), 136–147.

[4] CALISKAN, A., BRYSON, J. J., AND NARAYANAN, A. Semantics derived automatically from language corpora contain human-like biases. *Science 356*, 6334 (2017), 183–186.

[5] DOSHI-VELEZ, F., AND KIM, B. Towards a rigorous science of interpretable machine learning.

[6] GLOROT, X., AND BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), pp. 249–256.

[7] GOODMAN, B., AND FLAXMAN, S. European union regulations on algorithmic decision-making and a" right to explanation". *arXiv preprint arXiv:1606.08813* (2016).

[8] HERRMANN, C., AND THIER, A. Backpropagation for neural dnf-and cnf-networks. *Knowledge Representation in Neural Networks, S* (1996), 63–72.

[9] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *science 313*, 5786 (2006), 504–507.

[10] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[11] KINGMA, D., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] LAURENSON, J. Learning logical activations in neural networks.

[13] LECUN, Y., AND CORTES, C. The MNIST database of handwritten digits.

[14] LICHMAN, M. UCI machine learning repository, 2013.

[15] NEAPOLITAN, R. E., ET AL. *Learning bayesian networks*, vol. 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[16] OF EUROPEAN UNION, C. General data protection regulation, 2016.

[17] RIBEIRO, M. T., SINGH, S., AND GUESTRIN, C. 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* (2016), ACM, pp. 1135–1144.

[18] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[19] RUGGIERI, S., PEDRESCHI, D., AND TURINI, F. Data mining for discrimination discovery. *ACM Transactions on Knowledge Discovery from Data (TKDD) 4*, 2 (2010), 9.

[20] RUSSELL, S., NORVIG, P., AND INTELLIGENCE, A. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25* (1995), 27.

[21] TICKLE, A. B., ANDREWS, R., GOLEA, M., AND DIEDERICH, J. The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks 9*, 6 (1998), 1057–1068.