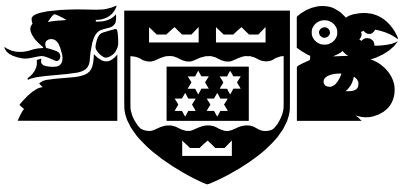
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



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

PO Box 600
Wellington
New Zealand

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

Logical Neural Networks: Opening the black box

Daniel Thomas Braithwaite

Supervisor: Marcus Frean

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

Abstract

Artificial Neural Networks (ANN's) are powerful because of their ability to approximate any function and generalize to unseen data, however they are essentially a black box, any knowledge learnt by an ANN is difficult to interpret. The ANN's presented in this report are more interpretable than standard Multilayer Perecptron Networks (MLPN's), moreover they have statistically equivalent performance and generalization to MLPN's when trained over boolean truth tables.

Contents

Chapter 1

Introduction

Artificial Neural Networks (ANN's) are commonly used to model supervised learning problems. NN's often achieve higher accuracy than other methods because they are able to approximate any continuous function. A well trained NN can generalize well but it is very difficult to interpret how the network is operating, this is called the black-box problem.

The growing number of situations in which ANN systems are used has lead to an increasing interest in systems which are not only accurate but also provide explanations of there answers [2]. This increasing interest is driven by the verity of applications of ANNs where incorrect or biases answers can have significant effects on users.

Safety and ethics are two concrete examples of such situations where explanations of reasoning would provide a method to ensure an ANN can be implement with mitigated risk [2]. In the context of Safety an ML system often can not be tested against all possible situations as it is computationally infeasible, the reasoning the ML system uses could be examined for flaws that could be result in a failure **Example Here**. On the other hand it is important to consider the implications of an ANN being biased towards a protected class of people, **Example Here**

Another pressure causing the development of this field is changing laws, in 2018 a European Union (EU) regulation will come into effect requiring algorithms which make decisions based on user level predictors must provide an explanation [3]. This EU regulation is heavily related to non-discrimination, and requires that measures be implemented which prevent discriminatory effects [3].

Explain Why Neural Networks Are Difficult To Interpret

Consider some function f, then the operation of restricting a neuron to f is given by only allowing a function given by f operating on the set of weighted inputs (each input multiplied by a corresponding weight) to be learnt. By restricting the function set that each neuron can learn is it possible to create a more interpretable network? This report develops a class of networks where the function space of each neuron is restricted to be a predefined operation which can be interpreted as Boolean expression i.e. an AND or OR of its inputs.

Boolean functions are by nature discrete, as such do not have a continuous differential, making them unsuitable for training with an algorithm such as Backpropagation. This report makes use of Noisy-OR and Noisy-AND neurons [6], which are generalized continuous parametrisations of OR and AND gates.

ANN's consisting of Noisy neurons are called Logical Neural Networks (LNN's). LNN's have been used to classify the MINST dataset with promising results, while not being able to achieve a state of the art accuracy they have been shown to yield simpler (sparser) and more interpretable weight vectors [6]. It was also shown that using Noisy neurons in combination with standard sigmoid units results in poor performing networks without the benefit of being more interpretable [6], for this reason networks of this kind are not considered.

This report takes a different approach to using these interpretable Noisy neurons, instead they are placed in specific configurations which allow learning Conjunctive or Disjunctive Normal Form expressions, these are called Logical Normal Form Networks (LNFN's) and are a subset of LNN's.

This report demonstrates by experimentation that when provided a complete truth table for a boolean expression, the performance of LNFN's has no statistically significant differences to that of a Multi-Layer Perceptron (MLPN) Network. The LNFN's are also able to generalize, obtaining statistically equivalent performance to a MLPN when given incomplete truth tables.

By inspecting the weights of each Noisy neuron it is possible to determine a relationship between the inputs and output. Rule extraction algorithms exist as a method to extract knowledge from NN's [1], if LNFNs are more interpretable then are rules able to be extracted from them? When first considering problems with Boolean inputs and outputs, if a low enough error is achieved when training an LNFN over a complete truth table, it is possible to extract boolean rules from each neuron, consequently it is possible to obtain a boolean expression which represents the ANN and original truth table. Training over entire a complete data set is an unlikely scenario, this report investigates what effect training with incomplete truth tables has on any extracted formula. Only being able to apply LNFNs to Boolean problems does not make them very useful, this report investigates ways these networks can be applied to problems with inputs in a continuous domain.

The restriction placed on the function space of each neuron, while improving the interpretability, intuitively will also hinder their ability to be universal approximators. Along with the investigation of LNFNs and their potential, the limitations are also explored.

Chapter 2

Background

2.1 Core Concepts

2.1.1 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 $(\neg a \wedge b) \vee (\neg a \wedge c)$.

2.1.2 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 2.1.1. 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. □

2.2 Literature Review

A survey in 1995 focuses on rule extraction algorithms [1], identifying the reasons for needing these algorithms along with introducing ways to categorise and compare them. Motivation behind scientific study is always crucial so why is understanding the knowledge contained inside Artificial Neural Networks's (ANN's) important? The key points identified are that the ANN might of discovered some rule or patten in the data which is currently not known, being able to extract these rules would give humans a greater understanding of the problem. Another, perhaps more significant reason is the application of ANN's to systems which can effect the safety of human lives, i.e. Aeroplanes, Cars. If using an ANN in the context of a system involving human safety it is important to be certain of the knowledge inside the ANN, to ensure that the ANN wont take any dangerous actions.

There are three categories that rule extraction algorithms fall into [1]. 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. Electic accounts for algorithms which inspect the hidden/output neurons individually but extracts rules which represent the ANN globally [10].

To further divide the categories two more classifications 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 [1].

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.

One rule extraction algorithm presented in 2000 by Tsukimoto is able to extract boolean rules from ANNs in which neurons have monotonically increasing activations, such as the sigmoid function [11]. The algorithm can be applied to problems with boolean or continuous inputs however only the boolean case will be considered here.

Each neuron is written as a boolean operation on its inputs, this is done in the following manner. Consider a neuron m in an MLPN with n inputs, construct a truth table, T, with n inputs. Let $f_i(i \in [0, 2^n])$ represent the activation of m when given row i as input and define $r_i = \wedge_{j=1}^{n} l_j$, the conjunction of all literals in the row i in T, e.g. if $n = 2$ then for row $(1, 0)$ $r_i = x_1 \wedge \neg x_2$. The approximation of m is given by

$$\vee_{i=1}^{2^n} g_i \wedge r_i \tag{2.1}$$

where g_i is given by

$$g_i = \begin{cases} 1 & \text{if } f_i \geq \frac{1}{2} \\ 0 & \text{if } f_i < \frac{1}{2} \end{cases}$$

By starting with the first hidden layer and progressing through the network an expression for the network can be extracted. Figure 2.1 shows an algorithm for extracting rules from an MLPN.

```
1  function extractRulesMLPN(network)
2    prev_expressions = network.inputs
3    for layer in network
4      expressions = {}
5      patters = all input patters for layer
6      for each neuron (n) in layer
7        exp = And(Or({literals(p), p∈patterns n(p) > 1/2}))
8        expressions.add(substitute prev_expressions into exp)
9
10     prev_expressions = expressions
```

Figure 2.1: Rule Extraction Algorithm presented in [11]

The algorythm presented in figure 2.1 is certainly exponential time in terms of n, a polynomial time algorithm is also presented but deriving such an algorithm is beyond the scope of this report.

In 1996 a class of networks, called Logical Normal Form Networks (LNFNs), where developed [4], 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 [4].

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, much justification is also missing making the LNFNs difficult to reproduce.

In 2016 the concept of Noisy-OR and Noisy-AND neurons where described [6], networks containing such Noisy neurons are called Logical Neural Networks (LNNs). Pure LNNs contain only Noisy neurons. This report aims to re derive LNFNs as a subset of Pure LNNs, that is using only Noisy neurons.

Noisy neurons are derived from the Noisy-OR relation[6], developed by Judea Pearl [9], a concept in Bayesian Networks. A Bayesian Network represents the conditional dependencies between random variables in the form of a directed acyclic graph.

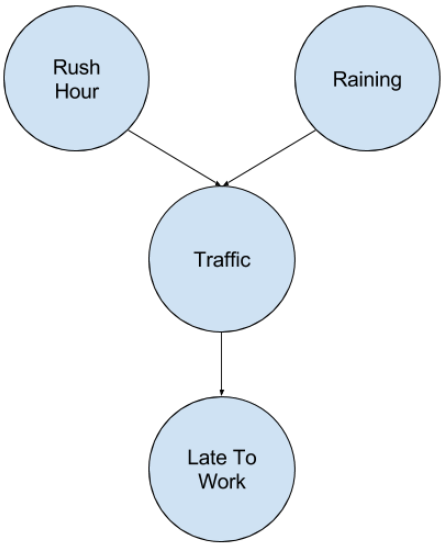


Figure 2.2

Figure 2.2 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 having the following configuration, take some node D with $S_1, ..., S_n$ as parents i.e. S_i influences the node D, each S_i is independent from all others. The relationship between D and its parents is if S_1 OR ... OR S_n is true then D is true. Let ϵ_i be the uncertainty that S_i influence D then $P(D = 1|S_1 = 1, , S_n = 1)$ can be defined.

$$P(D = 1|S_1 = 1, ..., S_n = 1) = 1 - \prod_{i=1}^{n} \epsilon_i \tag{2.2}$$

Equation 2.2 shows the noisy or relation [6]. In the context of a neuron, the inputs $x_1, ..., x_n$ represent the probability that inputs $1, ..., n$ are true. Consider the output of a neuron as conditionally dependent on the inputs, in terms of a Bayesian Network each x_i is a parent of the neuron. Each ϵ_i is the uncertainty as to whether x_i influences the output of the neuron. 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 that can be placed on f are given in [6]. (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.

Definition 2.2.1. A **Noisy-OR** Neuron has weights $\epsilon_1, ..., \epsilon_n \in (0, 1]$ which represent the irrelevance of corresponding inputs $x_1, ..., x_n \in [0, 1]$. The activation of a Noisy-OR Neurons is.

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

Definition 2.2.2. A **Noisy-AND** Neuron has weights $\epsilon_1, ..., \epsilon_n \in (0, 1]$ which represent the irrelevance of corresponding inputs $x_1, ..., x_n \in [0, 1]$. The activation of a Noisy-AND Neurons is.

$$a = \prod_{i=1}^{p} (\epsilon_i^{1-x_i}) \cdot \epsilon_b \tag{2.4}$$

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

ANN's containing of Noisy-OR and Noisy-AND neurons are called Logical Neural Networks (LNN's), if the network consists of only Noisy neurons then it a pure LNN. LNN's have been applied to the MINST dataset with promising results. Experiments with different combinations of logical and standard (sigmoid, soft-max) neurons have shown that pure LNN's where able to achieve an error of 8.67%, where a standard perceptron/softmax network was able to achieve an error of 3.13%. This reduction in performance does not come without reward, the pure LNN yields a simpler (sparser) and more interpretable network [6]. Training LNN's which are not pure have been shown to have reduced performance (compared to standard ANN's) and no interpretability benefit.

While the concept given in [4] is the foundation for the work given in this report, the approach presented is different that what has been done. A simpler parametrisation for disjunctive and conjunctive neurons is used, along with better justification and more investigation of the capabilities of LNFN's.

LNFNs are based off the idea of learning an underlying boolean formula, what happens when learning problems which consist of continous features? One option is to simply train the networks using the continuous features, the other is to descretise any continuous value.

There are a number of categories used to sort various descretization algorithms [8], three of these categories are defined below.

1. Supervised vs. Unsupervised: Supervised descretization algorithms use class information, as opposed to unsupervised which do not.

2. Dynamic vs. Static: Dynamic methods descretize features as the classifier is being built. A static algorithm performs all descretization before the lerning begins.

3. Global vs. Local: Global methods descretize the entire instance space where as local only descretize a subset of the total space.

The types of descretization algorithms which will be considered are Static and Global. Dynamic does not make a lot of sense when the data is used in training an ANN and a Local does not make sense if we are using a Static method. As for Supervised vs. Unsupervised it does not matter, though since we do have access to the labels using them could provide better performance.

Basic descretizing methods are Equal width and frequency binning, these are unsupervised algorithms. Equal width divides the interval up into K equally sized bins, where as equal frequency divides the interval into K bins where each one has the same number of instances. While implementation is easy it is not without a cost, namely results could be poor if the distribution of features is not uniform [8]. The two methods also require that K be pre defined, this could require an expensive trial and error process. [8].

A supervised technique is called Recursive Minimal Entropy Partitioning (RMEP). Let $S = \{s_1, ..., s_n\}$ be a set of instances, each with n attributes. The goal of RMEP is to partition the range of each attribute. The entropy of a set is given by equation 2.5

$$E(S) = \sum_i p_i \cdot log(p_i) \tag{2.5}$$

Chapter 3

Foundation of Logical Normal Form Networks

Consider problems with a boolean expression describing the relation ship between binary inputs and outputs. This is certainly a restricted space of problems, however its a logical place to start if defining the activation functions of neurons as a ”boolean like” function.

With this restriction in place any problem must be described by a boolean expression, this information alone does not help with constructing interpretable networks. It is known that any boolean expression has a CNF and DNF. It is possible to construct networks which can learn the underlying CNF or DNF, such networks are called Logical Normal Form Networks (LNFNs) [4]. The research developing LNFNs has little justification for key decisions, consequently is difficult to understand and reproduce.

Using the idea of LNFNs [4] and Pure Logical Neural Networks [6] new definitions are given for LNFNs in terms of Noisy-OR and Noisy-AND neurons.

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

A CNF or DNF formula contains clauses of literals which is either an atom or a negation of an atom. To account for this the number of inputs to the network will be doubled, the inputs will be all the atoms and negations of the atoms, i.e. if x_1, x_2 are the atoms then $x_1, \neg x_1, x_2, \neg x_2$ are the inputs to the 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 2.1.1).

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

$$\lim_{Z\to 0} Out(o) = 1 - \prod_{m=1}^{2^n} \mu_m^{h_m} \tag{3.1}$$

$$= 1 - 0 = 1 \tag{3.2}$$

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

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

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

In this configuration it must be shown that every hidden neuron fires when the network is presented with a variable assignment which corresponds to True and there is always at least one neuron which does not fire when the assignment corresponds to False. Assume for

a contradiction that for a given assignment $B(x_1, ..., x_n) = 1$ but $L(i_1, ..., i_{2n}) = 0$. Then there is at least one hidden neuron which does not fire. Let h_a be such a neuron. Consequently for any input weight combination which is relevant $\epsilon_{a,m}^{i_m} = 1$, so $i_m = 0$ for any relevant input. Let $i_{r_1}, ..., i_{r_k}$ be the relevant inputs then $i_{r_1} \vee ... \vee i_{r_k} = False$, so $\neg(\neg i_{r_1} \wedge ... \wedge \neg i_{r_k}) = False$, a contradiction as then $B(x_1, ..., x_n)$ would be False.

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

□

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 3.3, 3.4

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

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

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 3.4 and 3.3 for the Noisy-OR, Noisy-AND activations retrospectively allows LNFNs to be trained without the need to clip the weights.

Training the networks on all input patterns at the same time lead to poor learning, whereas training on a single example at a time had significantly better results. The ADAM Optimizer is the learning algorithm used, firstly for the convenience of an adaptive learning rate but also because it includes the advantages of RMSProp which works well with on-line (single-example) learning [5], which LNFNs respond well to.

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 perceptron networks which we know to be universal function approximators. Two different perceptron networks 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 3.1 shows a comparison between all 4 of the networks and figure 3.2 shows just the LNFN's.

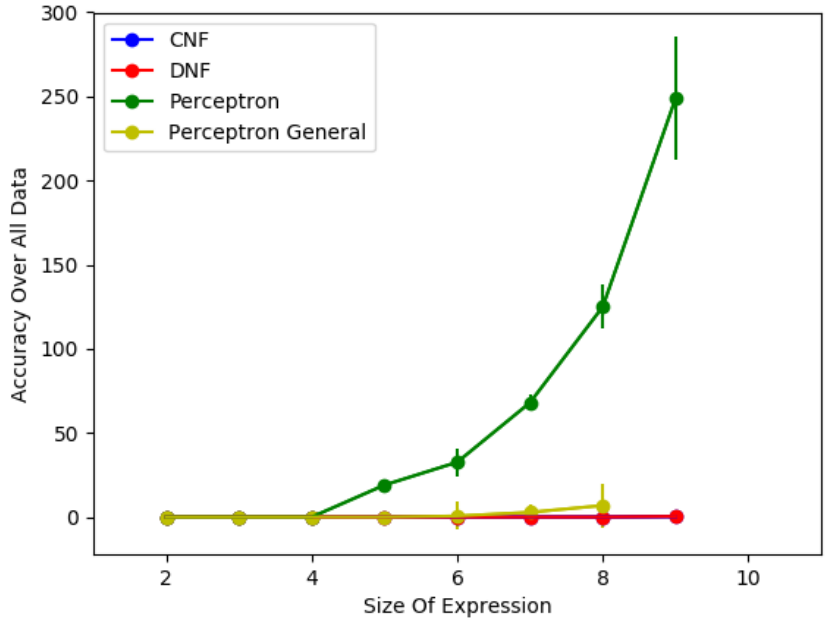


Figure 3.1

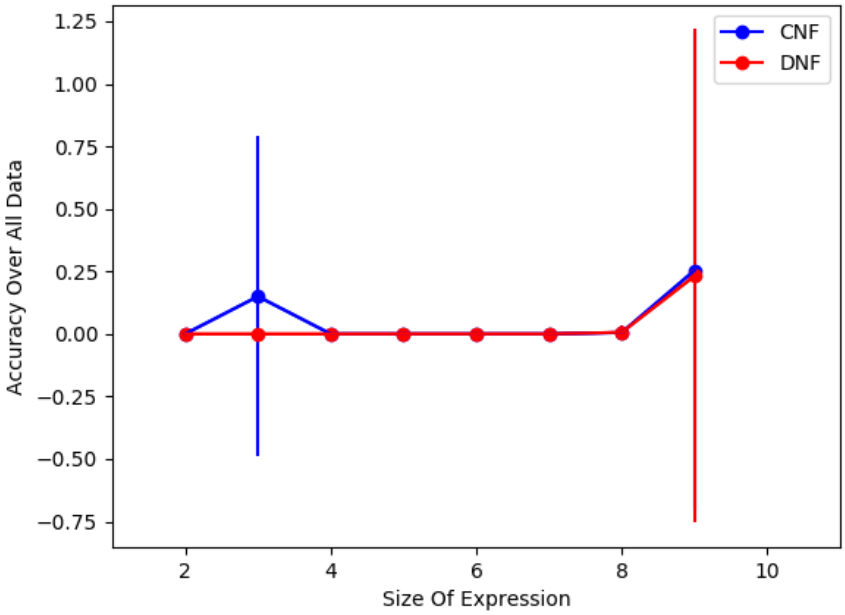


Figure 3.2

Figure 3.1 shows that neither of the perceptron networks perform as well as the LNF Networks as n increases. Figure 3.2 shows on average there are no statistically significant differences between the CNF or DNF networks. What is not present in Figure 3.2 is that sometimes the CNF network consistently out performs the DNF and visa versa, theoretically both should be able to learn any boolean expression.

What causes some expressions to be harder to learn for one type of LNFN compared to another?

3.4 LNF Network Generalization

These networks are able to perform as well as standard perceptron networks but so far they have been getting the complete set of data, in practice this will almost never be the case. Standard ANN's are widely used because of their ability to generalize, for LNFN's to be useful they must also be able to generalize.

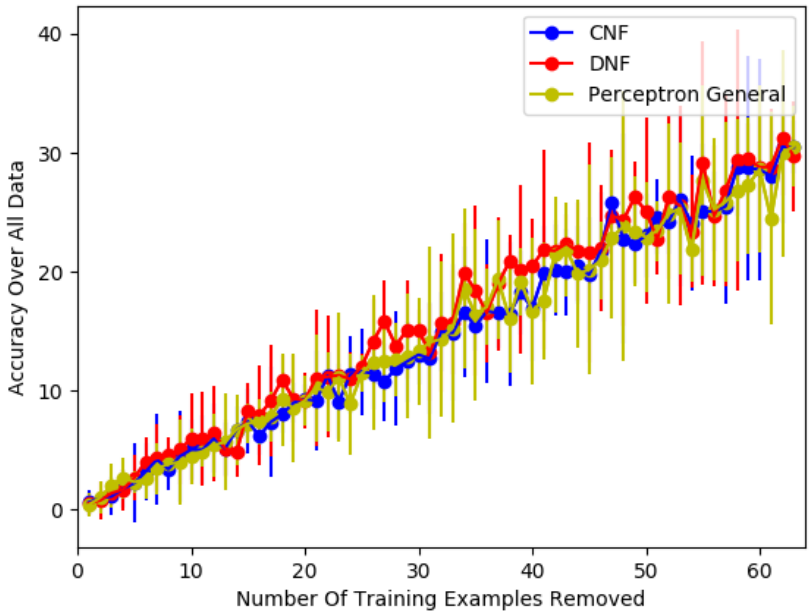


Figure 3.3

Figure 3.3 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 the boolean formula has 6 inputs, this trend continues as n increases up to 9.

3.5 LNF Network Rule Extraction

Given the logical nature of LNFNs, is it possible to extract boolean rules. 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 meaningful rules from the network using $\{\epsilon_1, ..., \epsilon_n\}$ it is important that at the conclusion of training each $\epsilon_i \approxeq 1$ or $\epsilon_i \approxeq 0$. If this is the case then it is possible to interpret the neuron as a purely logical function, say that the neuron in question was a Noisy-OR, then the neuron can be seen as performing a logical OR on all the inputs with correspond-

ing $\epsilon \approxeq 0$.

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 3.4 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 3.4: Rule Extraction Algorithm (for CNFN)

In practice many clauses in the extracted expression contain redundant terms, i.e. 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 $|B| = 2^{2^n}$. So 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 as discussed in Section 2.1.2, 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, learning the CNF and DNF may alleviate some of the issues caused by it, another possibility is to combine the CNF and DNF formulas to create a better rule set.

Figure 3.5 shows how the rule set of an CNFN generalizes as examples are removed, figure 3.6 shows the same but for DNFNs.

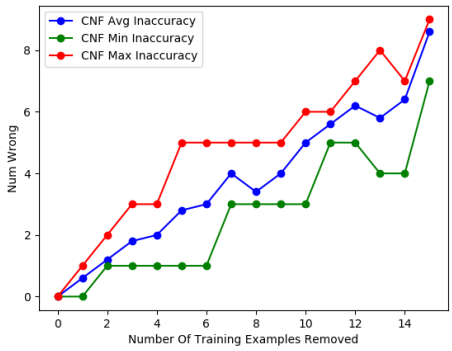


Figure 3.5

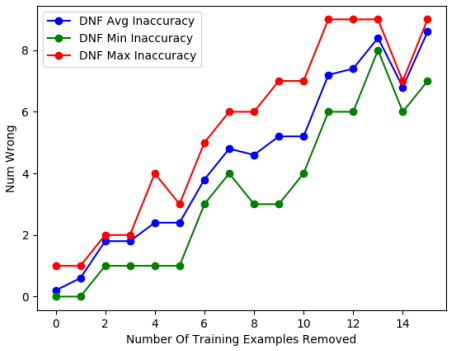


Figure 3.6

In figures 3.5 & 3.6 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.

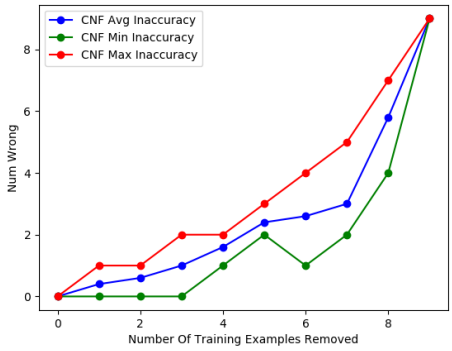


Figure 3.7

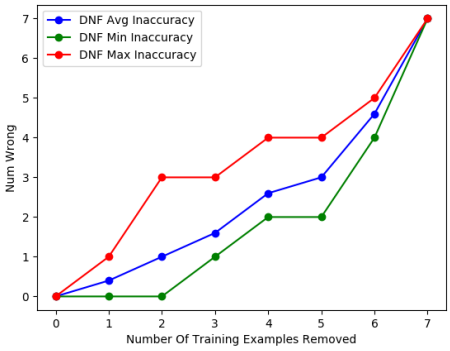


Figure 3.8

Figures 3.7 & 3.8 demontrait CNFNs and DNFNs trainined over partial data retrospectively. In the case of CNFNs only true entries of the truth table are removed and for the DNFNs only false entries. For the most part the minimum line is lower however the average line is roughly the same, this would indicate that while the choice of examples removed has an effect initial conditions have a more significant influence on the generalization performance.

Chapter 4

Investigation of Logical Normal Form Networks

4.1 Developing Applications

The LNFNs presented in Chapter 3 have very limited applications, the class of problems they can be applied to all have the form of binary classification where the features are Boolean. To make LNFNs more useful they need a broader scope.

4.1.1 Multi-class Classification

How can LNFNs be developed] to support classification of more than two classes. If attempting to learn a Multi-class Classification problem with n distinct classes $c_1, ..., c_n$, then n LNFNs would have to be trained, each a learning a binary classification problem c_i, $\neg c_i$.

An intuitive way to extend LNFNs to support Multi-class Classification is to add more output neurons and use One-hot encoding. If we have 3 classes then 100, 010 and 001 represent class 1, 2 and 3 retrospectively, then the LNFN would have 3 output neurons, each representing a bit in the One-hot encoded string.

Definition 4.1.1. The structure of an LNFN to solve an k class classification problem where each instance is described by n features has $2n$ inputs, 2^n hidden units and k output units. A final Softmax layer is added, as is convention for these types of problems.

A simple problem which lends its self naturally to this is the Lenses problem [7], a three class classification problem, each instance has 4 features, 3 of which are binary and the other has three possible values. This problem can be easily converted into one which can be used with an LNFN, each new problem instance will have 6 features, the three binary remain the same and the categorical one is expanded into 3.

How does an LNFN network perform when compared to a MLPN on the Lenses problem. The performance of the two classifiers will be compared using Leave-One-Out (LOE) Cross-Validation. The structure of the MLPN only 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 (Cross Entropy)	Confidence Interval (95%)
CNF Net	6.663	(6.468, 7.198)
DNF Net	6.660	(6.468, 7.197)
PCEP Net	6.751	(6.468, 7.997)

Table 4.1

Table 4.1 demonstrates that the CNF & DNF Networks perform comparably to an MLPN as the confidence intervals for the error 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. The following rules can be extracted from a CNFN when trained over the complete Lenses data, 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, namely $\neg f = True \implies$ Class 3. Table A.1 shows these rules applied to all the problem instances in the Lenses data set, it demonstrates that these extracted rules are able to fully describe the Lenses problem.

The DNFN might be more applicable as the rules will be more insightful, given its structure as an OR of ANDs.

- 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: $(\neg a \wedge \neg b \wedge c \wedge \neg d) \vee (\neg a \wedge d \wedge e) \vee \neg f$

These DNF formula do not correspond to the CNF give above but this is to be expected, despite the fact that collectivity all the problem instances span the space of features once they have been converted to the boolean feature form this fact is no longer true. there is a way to combine the knowledge from both the CNF and DNF formula to create resultant formula which perform better.

One possible issue that could arrive here is that the CNFN is now attempting to learn three CNF expressions with the same number of hidden neurons. The fact that meaningful rules where able to be extracted in this case could of been a coincidence, intuitively if we are learning a problem with k classes then we could need $k \cdot 2^n$ hidden neurons which is becoming impractical.

4.1.2 Features with Continuous Domains

The inputs to an LNFN are allowed to be continuous but must be in the rage $[0, 1]$, would it still be possible to extract meaningful rules from the network if the inputs are continuous? Here there are two things to investigate. What can be achieved by training an LNFN on problems with continuous features? Secondly what can be achieved by discretizing the continuous inputs and then using an LNFN to learn this new boolean problem. A simple benchmark to use is the Iris problem [7]

LNFNs and Continuous Features

If the features are continuous it no longer makes sense to extract rules but it could still be possible to see what inputs are considered in making a decision about the class. When training an LNFN on the Iris problem (over all problem instances) the network converges to a solution with a loss of 96.932, poor performance when compared to a perceptron network that can achieve an accuracy of 0.0.

Inspecting the class prediction of each reveals that for a problem instance that has a true class of Iris-virginica or Iris-versicolor then LNFN sometimes predicts multiple classes, this leads to the belief that these networks have issues with learning problems that are not linearly separable as the two classes which an LNFN has trouble differentiating between are not linearly separable.

The LNFN is able to learn XOR so not all problems which are not linearly separable are out of reach of LNFNs. Section 4.1.2 investigates whether it is possible to descretize the variables in such a way that makes this problem learnable by LNFNs.

LNFNs and Discretized Continuous Features

There are a number of method for discretizing continuous features, there are many algorithms for performing such an operation on some data [8], to many for all to be tested. Two simple methods for descretization are Equal width or frequency binning, these methods are very naive and prone to outliers, also the number of bins must be chosen before hand so this requires experimentation. A supervised partitioning method will also be tested, namely Recursive Minimal Entropy Partitioning.

Results from training LNFNs with descretized data results in the same issue as before, the network has problems with classifying classes that are not linearly separable.

Discussion of Application to Continuous Domains

The ideas explored in Sections 4.1.2 & 4.1.2 demonstrate that applying LNFNs to problems with continuous inputs is not viable.

Another thing to consider is that it becomes difficult to justify what the input features mean in the context of Noisy gates. In terms of the iris problem each feature represents a length, sure its possible to normalize the features to $[0, 1]$ and train an LNFN anyway but essentially the lengths are being forced to be some sort of truth value.

Chapter 5

Logical Neural Networks

The algorithm shown in Figure 3.4 presents a method for extracting rules from LNFNs, there is no reason why this algorithm cant be applied to Logical Neural Networks (LNNs) [6] which can take on any configuration.

An issue with LNFNs is that the number of hidden units allows for the possibility to memorise the input data, using LNNs consisting of more layers with a smaller width could force the learning of better rules.

Before investigating this 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.

Despite the issues outlined above being able to implement LNNs with layers that are not required to be 2^n hidden units wide would be a lot more practical, in practice the number of features could be upwards of 100 and 2^{100} is a huge number of hidden neurons.

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, a more complicated but more elegant solution 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.

An LNN, consisting of Noisy-OR hidden units and Noisy-AND outputs, with without any modification was shown to achieve a classification accuracy of 8.67 on the MNIST data set, it was also conjectured that using an LNN with an AND layer followed by OR does not learn well [6].

Before running any tests on the MNIST dataset LNNs with and without not operations will be compared.

5.1 Modified Logical Neural Network Structure

5.1.1 Connections Between Layers & Parameters

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

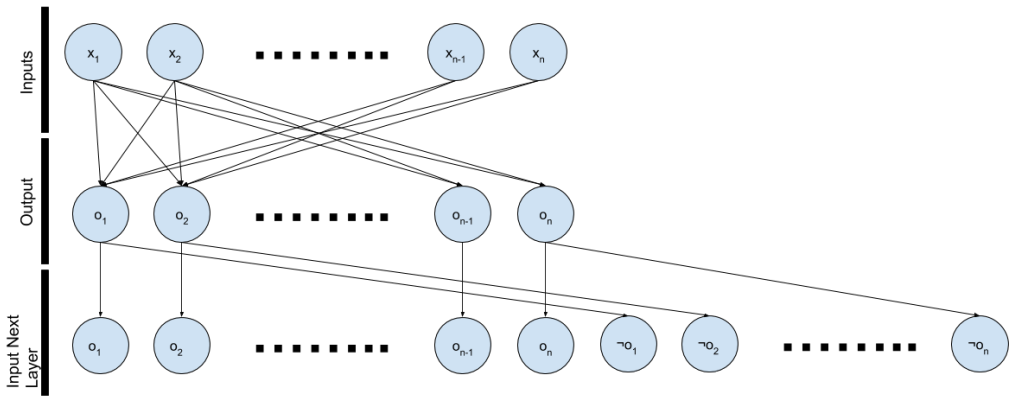


Figure 5.1

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 old LNN structure does not include a Softmax layer for the output. On the one hand a traditional Softmax layer would not be effective as the neuron outputs are limited to the range $[0, 1]$, so a different method must be employed to ensure the mutual exclusivity of the classes.

Consider the following vector of probabilities $P = \{p_1, ..., p_n\}$ where p_i is the probability the given example belongs to class i. 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.

5.2 Weight Initialization

This new structure is a super set of LNFNs, it should be possible to create and train an CNFN or DNFN with the modified LNN, however this is not the case. There are many more weights in the network now, this specifically causes an issue when the output neuron is a Noisy-AND, the current weight initialization technequic causes the output to saturate and nothing is able to be learnt when there exist to many neurons in the network. Before proceeding to experiment with the modified LNN a method to initialize the weights must be developed.

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 5.2.1. 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 $var(w_i)$.

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

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

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

Consequently the variance can be found by the following

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

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 $[0, \infty]$. Potential distributions are Beta Prime, Log Normal.

Fitting To Beta Prime Consider the Beta Prime distribution which as two parameters α and β which must be solved for given the mean and variance.

$$
\begin{aligned}
E[w_i] &= \frac{2}{n} \\
&= \frac{\alpha}{\beta - 1} \\
\alpha &= (\beta - 1) \cdot \frac{2}{n}
\end{aligned}
$$

$$
\begin{aligned}
var(w_i) &= \frac{3}{n} \\
&= \frac{\alpha(\alpha+\beta-1)}{(\beta-2)(\beta-1)^2} \\
&= \frac{(\beta-1)\cdot\frac{2}{n}((\beta-1)\cdot\frac{2}{n}+\beta-1)}{(\beta-2)(\beta-1)^2} \\
&= \frac{\frac{2}{n}((\frac{2}{n}+1))}{(\beta-2)} \\
(\beta-2) &= \frac{4+2n}{3n} \\
\beta &= \frac{4+8n}{3n}
\end{aligned}
$$

Finally giving $\alpha = \frac{10n+8}{3n^2}$.

Fitting To Log Normal **ADD MATH HERE**

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

Weight Initialization for LNNs 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 5.2: Weight Initialization Algorithm for LNNs

5.3 Conjectures Concerning Performance of LNNs

Based on experimental evidence the following Conjectures 5.3.1 & 5.3.2 can be made. Ideally formal arguments would be given, however this is out of scope for this project.

Conjecture 5.3.1. If the given network structure is unable to learn the formula it gets stuck.

Conjecture 5.3.2. If the loss is small enough then similarly to LNFNs rules can be extracted directly from the weights.

The LNN structure is significant when it comes to the learnability of problems so conjectures 5.3.1 & 5.3.2 allow for determining whether a network configuration is correct.

Chapter 6

Evaluation Of Logical Neural Networks

6.1 Tic Tac Toe

This problem involves classifying tic-tac-toe endgame boards and determining whether 'x' can win [7]. There are 9 categorical attributes, representing each cell of the board, each cell can have 'x', 'o' or 'b' for blank.

This gives a total of 27 attributes, if using an LNFN then the hidden layer would consist of 134217728 neurons, an intractable number, if the computer did not run out of memory then computing the gradients would be very slow.

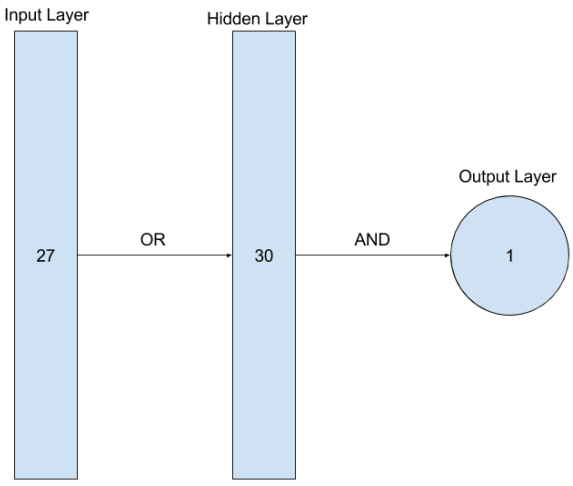


Figure 6.1

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 is able to achieve an error rate of 0.1% on the training set and 0.3% on the test set. The rules extracted from the network have an error rate of 1.7% on the training set and 6.3% on the test set.

What about the performance of the same network but swap around the activations? It was conjectured that networks of this form have poor learning [6] but with these new LNNs it might not be true any more.

Changing the configuration so that the AND activation is before the OR obtains a result which is better than the previous. The network has the same performance but the rule set extracted has a 0% error rate on the training set and a 2% error on the test.

It can be concluded that the conjecture saying that an LNN with an AND-OR configuration does not learn no longer holds.

6.1.1 Evaluation of LNN Rules

The rule extraction algorithm given in Figure 3.4 is an electic algorithm as this category describes algorithms that examine the network at the unit level but extract rules which represent the network as a whole. The algorithm is certainly 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 [1].

1. **Accurate:** As demonstrated experimentally the extracted rules are able to generalise well to unseen examples.
2. Fedelity: The experiments also show that the rules perform very similar to the network they where extracted from.
3. Consistency: **TEST THIS**

6.2 Application to MNIST

To determine if this new LNN structure can out perform the old it will be tested on the MNIST problem. The old LNN structure was able to achieve a classification accuracy of 8.67% on a test set.

There are 784 features per example and each will lie in the range $[0, 1]$. In Section 4.1 it was discussed why these networks should not be applied to continuous domains, however in this case the inputs still make sense, each feature can be interpreted as the probability that the corresponding pixel in the image is white.

Given that the features are continuous the rules no longer make sense but it is still possible to visualise the weight to get an understanding of what the network has learnt. A number of different architectures are trained and inspected, below is a list of each along with a justification as to why it could be effective.

1. **OR-AND Original LNN**, this is because the Noisy parametrisation has been changed and while it should be equivalent it is possible that it is not.
2. **OR-AND Modified LNN**, to compare the modified structure with the old.
3. **AND-AND Modified LNN**, a logical way to consider a digit is as an AND of features where each feature is an AND of pixels, however a problem with this approach could be that the border of each feature is fuzzy due to the different positioning and shape of the digits.
4. **AND-OR-AND Modified LNN**,

6.2.1 OR-AND Modified LNN

An OR-AND LNN with 30 hidden neurons and the modified structure is able to achieve an 4% error rate on the training data along with a 4.5% error rate on the test set, this is a significant improvement on previous results making LNNs a more feasible option.

The intepretability of these weights is also something which needs to be assessed. There are two things to consider, first the individual weights but also the combinations of hidden features which make up each digit.

Figures 6.2 & 6.3 & 6.4 & 6.5 show the weights of 4 distinct neurons in the LNN model. Each image, while sparse, is not necessarily any more interpretable, it is not obvious what each represents.

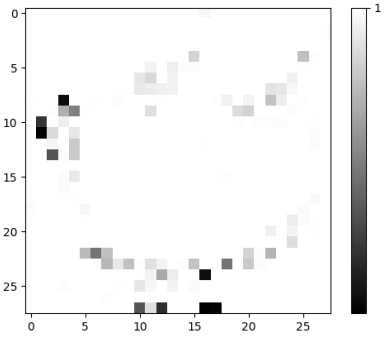


Figure 6.2

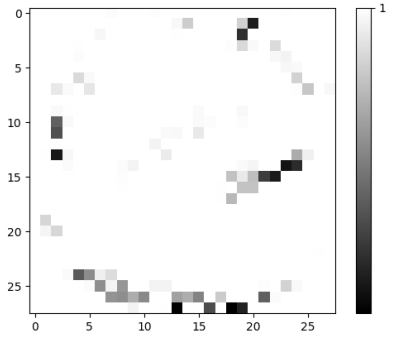


Figure 6.3

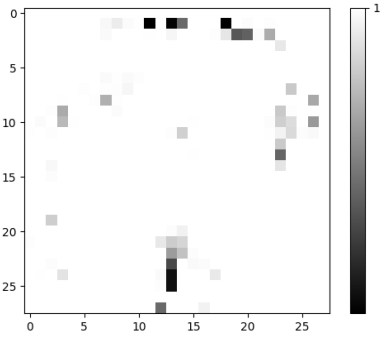


Figure 6.4

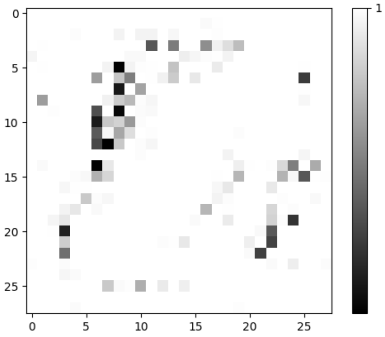


Figure 6.5

In terms of the pixels considered in deciding between digits, Figures 6.6 & 6.7 & 6.8 & 6.9 show the pixel weightings with respect to digit 2,3,5,8 retrospectively. The results are promising as there are cerntialy noticeable similarity with the digits they are attempting to represent, inpeticular the outlines of each of the digits are certainly present. These visualisations indicate that this trained network is using the outlines of the numbers to make its determination.

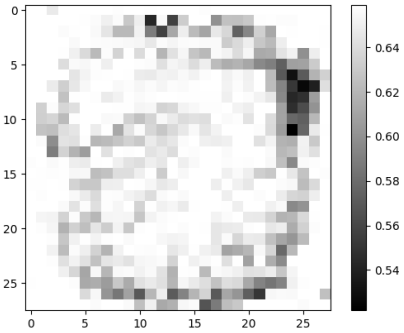


Figure 6.6

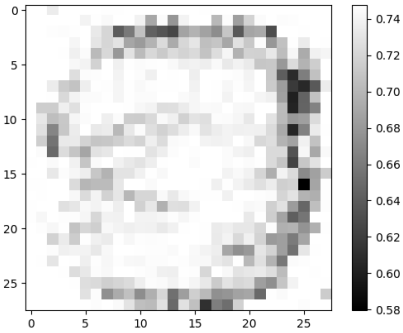


Figure 6.7

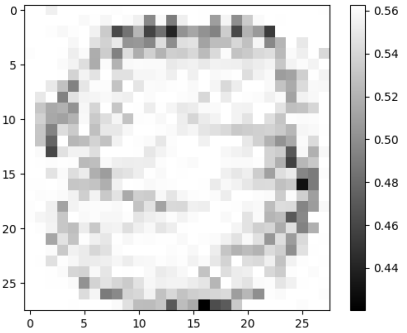


Figure 6.8

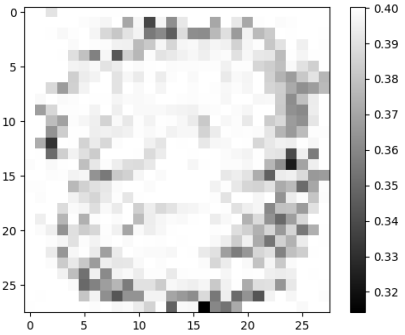


Figure 6.9

The model is certainly interpretable, however only once the features have been combined do they make sense. In an ideal world each of the individual features would correspond to something which represents part of a digit. In an effort to make the hidden features more interpretable another OR-AND model is learnt but with fewer hidden neurons.

6.2.2 AND-OR Modified LNN

This model required a lower learning rate and thus a longer training time.

6.2.3 AND-AND Modified LNN

Training a Modified LNN with 30 hidden neurons using an AND-AND structure was able to achieve a 3.7% error rate on the training set and 4.1% error rate on the test set. Similarly to the OR-AND model the hidden features are sparse but not necessary interpretable, as shown by Figures 6.10 & 6.11 & 6.12 & 6.13

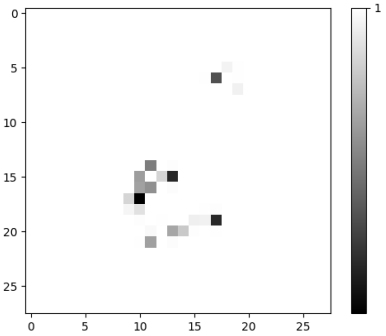


Figure 6.10

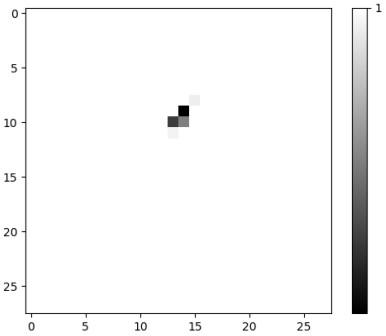


Figure 6.11

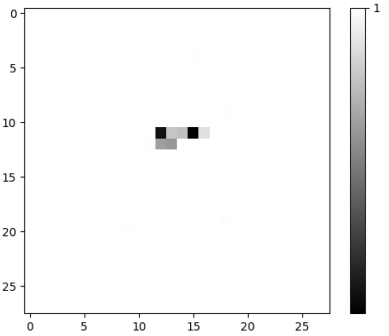


Figure 6.12

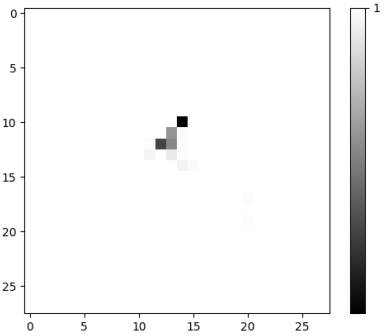


Figure 6.13

Figures 6.14 & 6.15 & 6.16 & 6.17 show the image pixels considered for the output neurons that classify 2,3,5,8 retrospectively. While obtaining a comparable performance to the OR-AND structure these weight patterns are much less interpretable, there is no distinctive shape in any which indicate there corresponding number. The only determination which can be made is that these appear to consider the filling of each digit as opposed to the boundary, which intuitively makes sense, the boundary will be inherently noisy (as each digit is drawn slightly differently) and AND of pixels will be heavily penalised for anything missing. This is evidence that while using ANDs in a first hidden layer might provide good performance it does not lead to interpretable results.

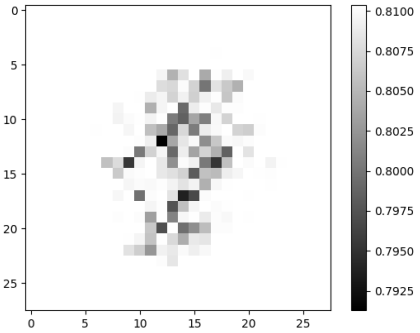


Figure 6.14

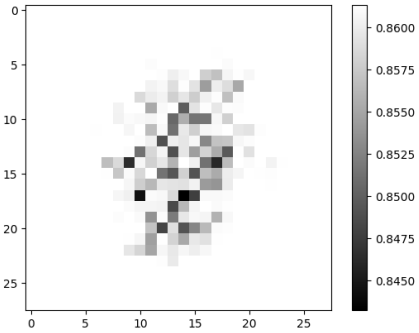


Figure 6.15

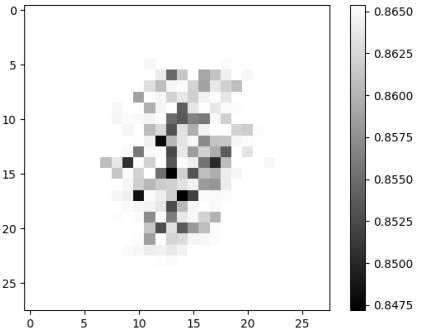


Figure 6.16

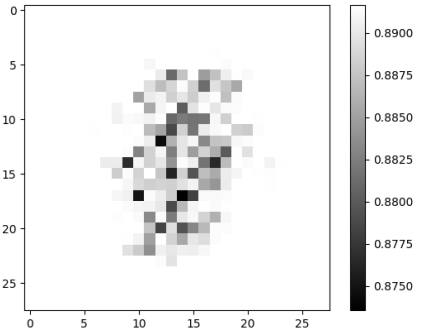


Figure 6.17

6.2.4 Discussion Of Results

Chapter 7

Application to *

This will hopefully be LNNs applied to something cool!

Bibliography

[1] 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.

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

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

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

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

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

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

[8] LIU, H., HUSSAIN, F., TAN, C. L., AND DASH, M. Discretization: An enabling technique. *Data mining and knowledge discovery 6*, 4 (2002), 393–423.

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

[10] 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.

[11] TSUKIMOTO, H. Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks 11*, 2 (2000), 377–389.

Appendices

Appendix A

Tables

a	b	c	d	e	f	Rule Out	Expected
1	0	0	0	0	0	(0, 0, 1)	(0, 0, 1)
1	0	0	0	0	1	(0, 1, 0)	(0, 1, 0)
1	0	0	0	1	0	(0, 0, 1)	(0, 0, 1)
1	0	0	0	1	1	(1, 0, 0)	(1, 0, 0)
1	0	0	1	0	0	(0, 0, 1)	(0, 0, 1)
1	0	0	1	0	1	(0, 1, 0)	(0, 1, 0)
1	0	0	1	1	0	(0, 0, 1)	(0, 0, 1)
1	0	0	1	1	1	(1, 0, 0)	(1, 0, 0)
0	1	0	0	0	0	(0, 0, 1)	(0, 0, 1)
0	1	0	0	0	1	(0, 1, 0)	(0, 1, 0)
0	1	0	0	1	0	(0, 0, 1)	(0, 0, 1)
0	1	0	0	1	1	(1, 0, 0)	(1, 0, 0)
0	1	0	1	0	0	(0, 0, 1)	(0, 0, 1)
0	1	0	1	0	1	(0, 1, 0)	(0, 1, 0)
0	1	0	1	1	0	(0, 0, 1)	(0, 0, 1)
0	1	0	1	1	1	(0, 0, 0)	(0, 0, 1)
0	0	1	0	0	0	(0, 0, 1)	(0, 0, 1)
0	0	1	0	0	1	(0, 0, 1)	(0, 0, 1)
0	0	1	0	1	0	(0, 0, 1)	(0, 0, 1)
0	0	1	0	1	1	(1, 0, 0)	(1, 0, 0)
0	0	1	1	0	0	(0, 0, 1)	(0, 0, 1)
0	0	1	1	0	1	(0, 1, 0)	(0, 1, 0)
0	0	1	1	1	0	(0, 0, 1)	(0, 0, 1)
0	0	1	1	1	1	(0, 0, 0)	(0, 0, 1)

Table A.1: Rules extracted from CNF applyed to Lenses problem