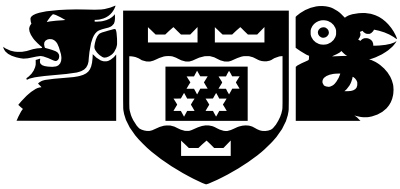
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



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

A short description of the project goes here.

Contents

Chapter 1

Introduction

Neural Networks (NN'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.

There are a number of motivations for wanting to solve the black-box problem. If a NN is able to provide an explanation for its output a deeper understanding of the problem can be developed, the rules or patterns learnt by an NN could represent some knowledge in the data which has not yet been identified. Another possibility is that the NN is being implemented to operate a critical systems which involves the safety of humans, a situation which is becoming more common place. In the context of safety critical systems being able to inspect the NN is a necessary part of ensuring the system is safe, because a NN could take potentially dangerous actions for situations where the action must be extrapolated [1].

By restricting the function set that each neuron can perform is it possible to create a more interpretable network? Restricted the functions for each neuron to be the function set taking some subset of inputs and perform a pre determined logical function, after training to identify the function each neuron is performing on its inputs only the subset of inputs considered must be identified as the operation is fixed.

This report develops a class of networks in which the function space of each neuron is restricted to be a predefined operation on a subset of its inputs, ideally it would be possible to consider this operation as a Boolean function, consequently each neuron can be interpreted as logical operation on 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. Instead of using discrete boolean functions this report makes use of Noisy-OR and Noisy-AND neurons [4], which are generalized continuous parametrisations of OR and AND gates, meaning neurons are restricted to performing operations on a subset of their inputs which correspond to a logical OR or AND. Noisy neurons are placed in specific configurations which allow learning Conjunctive or Disjunctive Normal Form expressions, they are called Logical Normal Form Networks (LNFN's).

By the design of LNFN's it is possible to extract logical rules from the network once trained, by inspecting the weights of each Noisy neuron it is possible to determine the relationship between the inputs and output. Rule extraction algorithms all ready exist as a method to extract knowledge from NN's [1], some aim to discover rules which replace the NN and others extract rules which are combined with the NN to improve performance.

Rule extraction algorithms are generally split into three categories. The **Decompositional Approach** extracts rules by analysing the activations and weights in the hidden layers. The **Pedagogical Approach** works by creating a mapping of the relationship between inputs and outputs. Finally The **Eclectic Approach** combines the previous two approaches [1].

The LNFN's developed in this report present an intuitive eclectic method for extracting rules from such a network. 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 development of a new rule extraction approach this report will identify and explore the issues introduced by the restriction placed on neurons to determine where the rule extraction algorithm can be used.

When provided a 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 equal performances as a MLPN when given incomplete truth tables.

Chapter 2

Proposal Review

Originally proposed was to restrict the function set of neurons to the NAND ($\uparrow$) operations, and construct feedfoward networks with these restricted neurons, however this idea is flawed. Consider the expression p implies q, $p \implies q \iff p \uparrow (q \uparrow q)$.

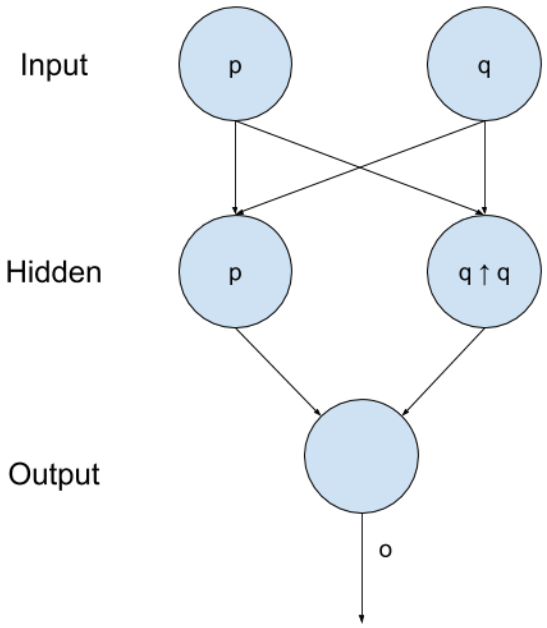


Figure 2.1

Figure 2.1 demonstrates the structure a feed-forward network attempting to represent implies would look like. Two inputs and one output, defined by the problem. Two hidden neurons as implies is the NAND of two values, p and $q \uparrow q \iff \neg q$, consequently the output neuron is computing the NAND of two hidden units. For o to be the output required then one node in the hidden layer would have to act as an identity, passing the input p to the output neuron. NAND cant act as an identity.

Chapter 3

Background Survey

3.1 Core Concepts

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

3.1.2 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.

3.2 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 network, to be sure 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 [6].

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.

The paper "Backpropagation for Neural DNF- and CNF-Networks" presents an approach which relies on a special NN architecture. The neurons in these networks have a restricted function space, they are only able to perform a OR or AND on a subset of their inputs. By restricting the degrees of freedom in the network it is possible to understand the actions each neuron is taking. Rules can simply be extracted from the trained network by inspecting these neurons [2]. The conjunctive and disjunctive neurons presented, while making sense mathematically are cumbersome to implement. A perfected way to describe logical neurons will be to use Noisy-OR and Noisy-AND gates [4], derived from the Noisy-OR relation which was developed by Judea Pearl [5], a concept in Bayesian Networks.

A Bayesian Network represents the conditional dependencies between random variables in the form of a directed acyclic graph.

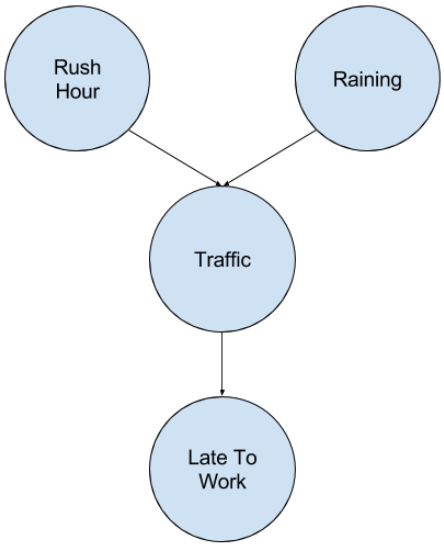


Figure 3.1

Figure 3.1 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 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.

$$P(D = 1|S_1 = 1, ..., S_n = 1) = 1 - \prod_{i=1}^{n} \epsilon_i \tag{3.1}$$

In the context of a neuron, the inputs $x_1, ..., x_n$ represent the probability that inputs $1, ..., n$ are true. 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 [4]. (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 3.2.1. 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.

$$a = 1 - \prod_{i=1}^{p} (\epsilon_i^{x_i}) \cdot \epsilon_b \tag{3.2}$$

Definition 3.2.2. A **Noisy-AND** Neuron has weights $\epsilon_1, ..., \epsilon_n \in [0, 1]$ which represent the irielevence of corresponding inputs $x_1, ..., x_n \in (0, 1]$. The activation of a Noisy-AND Neurons is.

$$a = \prod_{i=1}^{p} (\epsilon_i^{1-x_i}) \cdot \epsilon_b \tag{3.3}$$

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

While the concept presented in [2] is the foundation for the work presented in this report, the approach presented is different that what has been done. A different approach to disjunctive and conjunctive neurons is taken, along with this more investigation is carried out in terms of the capabilities of these networks (when compared to a standard perceptron) and the rule extraction method.

Chapter 4

Logical Normal Form Networks

Before considering the case of continuous variable assume the problem we are trying to learn can be represented by a boolean formula. Then it is known that this formula must have a Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF). This section explores networks which learn the CNF or DNF representation of any underlying boolean expression.

The paper "Backpropagation for Neural DNF and CNF Networks" [2] proposes networks which can solve this task, however the paper does not provide justification, consequently it is difficult to understand and reproduce. This report takes this concept but rederives it using the Noisy-OR and Noisy-AND gates [4] and further investigates the properties of such networks.

Definitions for CNF and DNF networks are now given.

Definition 4.0.1. A **CNF-Network** is a three layer network where neurons in the hidden layer consist soley of Noisy-OR's and the output layer is a single Noisy-AND.

Definition 4.0.2. A **DNF-Network** is a three layer network where neurons in the hidden layer consist soley of Noisy-AND's and the output layer is a single Noisy-OR.

Definition 4.0.3. 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, i.e. the inputs will be all the atoms and negations of thoughts atoms.

It must also be determined how many hidden units to have the the network, 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.

Theorem 4.0.1. Let T be the complete truth table for the boolean formula B. Let L be an LNF network, if L has 2^n hidden units then there always exists a weight assignment to L which always 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 LNF network, 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 corospnding 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

$$\lim_{Z\to 0} Out(o) = 1 - \prod_{m=1}^{2^n} \mu_m^{h_m} \qquad = 1 - 0 = 1 \tag{4.1}$$

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 consider that any relevant neuron (i.e. cosponsoring $\mu \neq 1$) will have some input weight pair of i_m ϵ_m such that $\epsilon_m^{i_m} = 0$. Moreover it can be said that for all m $\mu_m^{h_m} = \mu_m^0 = 1$, consequently 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 $\neg(i_{r_1} \wedge ... \wedge i_{r_k}) = False$, 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 relevent inputs, at least one relevant input is present i_r. Consequently $i_{r_1} \vee ... \vee i_{r_k} = True$, therefore $\neg(i_{r_1} \wedge ... \wedge i_{r_k}) = True$, a contradiction as then $B(x_1, ..., x_n)$ is True.

□

Theorem 4.0.1 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.

4.1 Noisy Gate Parametrisation

The parametrisation of Noisy gates require weight clipping, an expensive operation. A new parametrisation is derived that implisitley 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.

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

From a similar derivation we get the activation for a Noisy-AND, concisely giving equations 4.2, 4.3

$$a_{and}(X) = e^{W^{'} \cdot (1-X) + b^{'}} \tag{4.2}$$

$$a_{or}(X) = 1 - e^{-(W^{'} \cdot X + b^{'})} \tag{4.3}$$

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

4.2 Training LNF Networks

Using equations 4.3 and 4.2 for the Noisy-OR, Noisy-AND activations retrospectively allows the networks to be trained without explicit clipping. The ADAM Optimizer is used for

training 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-training) learning [3], which LNF Networks respond well to.

Preliminary testing showed that LNF Networks 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.

4.3 LNF Network Performance

How do LNF Networks perform against standard perceptron networks which we know to be universal function approximators. Two different perceptron networks will be used as a benchmark, one with the same configuration as the LNF Network, the other with less hidden 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 4.1 shows a comparison between all 4 of the networks and figure 4.2 shows just the LNF Networks.

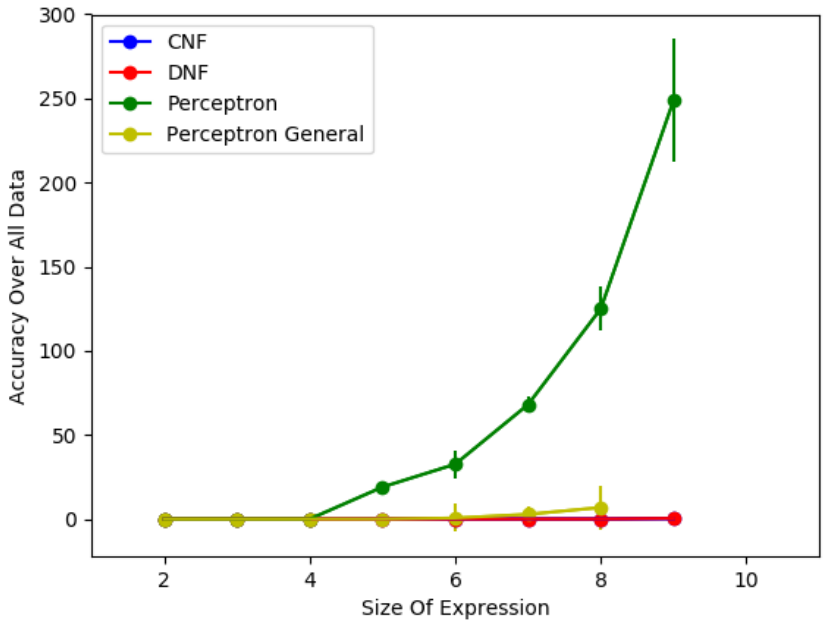


Figure 4.1

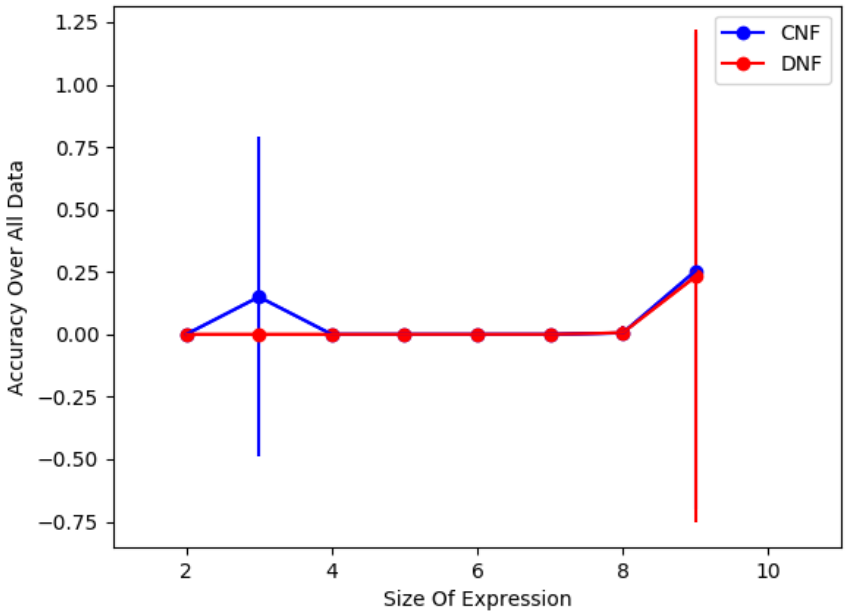


Figure 4.2

Figure 4.1 shows that neither of the perceptron networks perform as well as the LNF Networks as n increases. Figure 4.2 shows on average there are no statistically significant differences between the CNF or DNF networks. What is not present in 4.2 is that at $n = 9$ sometimes the CNF network far out performs the DNF and visa versa, theoretically both should be able to learn any boolean expression.

4.4 LNF Network Rule Extraction

The goal of this report is to learn rules which can be extracted from the network, consequently a method must be developed to extract the rules from LNF Network's. Take the weights of a trained LNFN, these weights can be converted back into ϵ_i's by apply the sigmoid function to each w_i.

As $\epsilon_i \to 0$ then x_i becomes relevant and as $\epsilon_i \to 1$ then x_i becomes irrelevant. If the network has learnt the correct DNF or CNF representation the for every neuron if input i is relevant then $w_i \to -\infty$ and therefore $\epsilon_i \to 0$, otherwise x_i is irrelevant and $w_i \to \infty$ meaning $\epsilon_i \to 1$.

Consequently in ϵ form the network is binary and rules can be easily extracted. Many of the formulas extracted contain redundant terms, i.e. clauses that are a tautology or a duplicate of another, filtering these out is not an expensive operation.

When training an LNF network over the entire truth table for a boolean expression, when a low error is achieved it is possible to extract boolean formula from the network which gets can generate the original truth table. This is a necessary first step but a more important question is, can formula still be extracted from the network when the LNF network is not trained with the entire truth table?

4.5 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. Perceptron networks are so widely used because of their ability to generalize, for LNF Networks to be useful they must also be able to generalize.

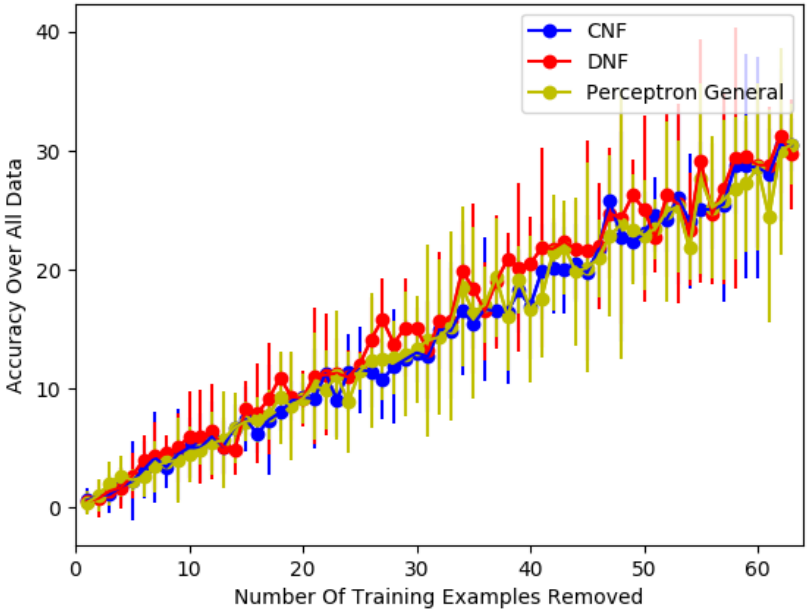


Figure 4.3

Figure **??** 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.

Chapter 5

Future Plan

1. Investigate why the DNF network learns better than CNF in some ocations and visa versa. **2 Weeks**
2. Compare performance and intepretability on some benchmark problems which have boolean inputs. **2 Weeks**
3. Investigate applying LNF Networks to problems with continuous inputs, will the inputs be descretized? **4 Weeks**
4. Compare against other rule extraction methods, using criteria laid out in [1]. This would include and investigate the generalization of the extracted rules as the training set is reduced. It has been shown that LNF Networks are able to generalize **4 Weeks**

This plan accounts for other work during semester 2, and aims to have the work finished before the final report submission to allow a couple of weeks for final report writing.

Chapter 6

Request For Feedback

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

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

[4] LAURENSON, J. Learning logical activations in neural networks, 2016.

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

[6] 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.