

# Logical Neural Networks: Opening the black box

## Logical Normal Form Network

Daniel Braithwaite

May 23, 2017

Given a simple feed forward network using neurons resembling NAND gates is unable to learn any given boolean formula we switch our interest to something else. We know that any boolean expression can be represented in Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF). So it seems prudent to ask whether it's possible to construct a feedforward neural network which can learn these underlying representations. Findings [1] seem to suggest that it is possible however there is limited justification for some claims and because of this is difficult to reproduce. We will take this general concept and reproduce the research in an attempt to develop a better understanding.

## 1 Noisy Neurons

During a previous investigation of Logical Neural Networks the concept of Noisy-OR and Noisy-AND neurons were derived [2]. Here we will simply state them as.

**Definition 1.1.** A **Noisy-OR** neuron is a perceptron with activation  $a = 1 - e^{-z}$  where  $W$  is our weights,  $X$  is our inputs,  $b$  is our bias term and  $z = WX + b$ . We constrain each  $w_i \in W$  and  $b$  to be in the interval  $[0, \text{inf}]$

**Definition 1.2.** A **Noisy-AND** neuron is a perceptron with activation  $a = e^{-z}$  where  $W$  is our weights,  $X$  is our inputs,  $b$  is our bias term and  $z = WX + b$ . We constrain each  $w_i \in W$  and  $b$  to be in the interval  $[0, \text{inf}]$

## 2 Logical Normal Form Networks

A Logical Normal Form Network is a neural net which satisfies one of the following definitions

**Definition 2.1.** CNF-Network 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 2.2.** DNF-Network 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.

It is worth noting that CNF and DNF form can have the notes of atoms in there clauses, a simple way to account for this is to double the number of inputs where one represents the atom and the next represents the negation of that atom.

## 2.1 Learnability Of Boolean Gates

Theoretically it makes sense for these networks to be able to learn the CNF and DNF representations of various boolean expressions, however before we attempt arbitrary boolean functions we would like to start with something simple, namely expressions such as NOT, AND, NOR, NAND, XOR and IMPLIES. Results of which were promising, we were able to achieve a low error and from inspecting the weights we could see that the networks were in fact learning the correct CNF and DNF representations.

## 2.2 Learnability Of Interesting Expressions

We now wish to see if we can use these networks to learn more interesting boolean formula, starting with expressions of 3 variables. While we are able to achieve a small error there is now some noise (i.e. small non zero weights for inputs that are irrelevant). While a small amount of noise is okay and can be pruned out after training we could run into issues if the amount of noise increases as the number of inputs does.

One other interesting observation to be made is that both the CNF and DNF Networks have trouble learning the boolean expression  $(a \text{ XOR } b) \text{ AND } c$ . They can't achieve an error lower than 1. Individually we can learn an XOR gate and an AND gate but somehow combining the two results in something which is unlearnable.

During the investigation of this a more sinister issue was uncovered, namely that these LNF networks are unable to learn boolean expressions of 2 variables when given 3. I.e. we give the network all values for three inputs, a, b and c but we only want to learn  $a \text{ OR } b$ . This is a fundamental issue as in practice most problems will be made up of boolean expressions which don't rely on all inputs.

It turns out this problem was not caused by a problem with our LNF Networks but with the weight initializations, namely they were currently initialised to 0, changing the weights to be randomly distributed over the interval  $[0,1]$  fixed this problem and allows the LNF Networks to solve all attempted problems so far, along with their weight representations being interpretable. However the

networks seem quite sensitive to their initial conditions, an investigation of best ways to initialise LNF networks would be prudent.

### 3 LNF Network Performance

We wish to compare the CNF and DNF networks against each other but also against standard perceptron networks. We will take 5 randomly chosen boolean expressions of  $n$  inputs for  $n$  between 2 and 10. For each we will train CNF, DNF and Perceptron networks 5 separate times and use the datapoints to perform significance tests between the three.

### 4 LNF Network Generalization

Say we are trying to learn a boolean function of  $n$  inputs, we know for a fact there are  $2^n$  total input patterns in total for this boolean function. We also know there are  $2^n$  total possible boolean functions with  $n$  inputs. So an important question to ask is once we start to remove some of the training examples what happens to our network accuracy? The usefulness of standard neural networks is in part because they are able to take a sample of the total data set and then generalise well to unseen examples, can we achieve the same with LNF networks?

We will take one boolean expression with  $n = 4$  and slowly remove from the pool of training examples, training a fresh network each time. This will allow us to see a trend of network accuracy as we remove more and more from the training pool.

### 5 LNF Network Rule Extraction

The motivation for this project is to be able to train these networks and essentially extract rules from them. However as the networks get larger we see that the networks pick up noise so we must investigate how to remove this noise and then extract rules from the network

#### 5.1 LNF Network Pruning

A nice property of LNF Networks is that their weights directly represent how relevant they are to the network. The larger the weight the more important that particular input is. Our goal here is to find weights less than some threshold and set them to 0, removing noise and allowing us to uncover the CNF or DNF expression (or something close to it).

The question now becomes how to we decide on a threshold. We could investigate trained networks for which we know what the representation should

look like and decide on what an appropriate threshold should be. From my investigation I have found that in all cases weights less than 1 represent noise, so this would be a good place to start. We will call this method relevance pruning.

### 5.1.1.1 Relevance Pruning

---

**Algorithm 1:** Algorithm for performing Relevance Pruning on some LNF Network

---

```

1 Function: Relevance Prune (hidden, output, threshold)
   Input : The weights for the hidden nodes and output nodes finally a
           threshold for the weights
   Output: Pruned weights
2  $H \leftarrow \emptyset$   $O \leftarrow \emptyset$ 
3 for  $i \leftarrow 1$  to  $|H|$  do
4   if  $output_i < threshold$  then
5      $H_i \leftarrow 0, \dots, 0$   $O_i \leftarrow 0$ 
6   end
7   else
8      $H_i \leftarrow PRUNE(hidden_i, threshold)$  /* PRUNE simply sets any
           value less than threshold to 0 */
9      $O_i \leftarrow output_i$ 
10  end
11  return  $H, O$ 
12 end

```

---

It should be noted that this algorithm doesn't account for biases being in the weight vectors, if they are some indices will need to be shifted

## References

- [1] HERRMANN, C., AND THIER, A. Backpropagation for neural dnf-and cnf-networks. *Knowledge Representation in Neural Networks, S* (1996), 63–72.
- [2] LAURENSEN, J. Learning logical activations in neural networks, 2016.