Logical Neural Networks: Opening the black box

Logical Normal Form Network

Daniel Braithwaite

May 26, 2017

Given a simple feed foward network using neurons resembleing 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 wether its possible to construct a feedfoward netural netowrk which can learn these underlying representations. Findings [1] seem to sugest that it is possible however there is limited justification for some claims and because of this is diffcult 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 where 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 byas term and $z = WX + b$. We contrain each $w_i \in W$ and b to be in the interval $[0, \text{inf}]$

Definition 1.2. A **Noisy-OR** neuron is a perceptron with activation $a = e^{-z}$ where W is our weights, X is our inputs, b is our byas term and $z = WX + b$. We contrain 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 stasfys one of the folowing definitions

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

It is worth noteing that CNF and DNF form can have the nots 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

Theoretiaclly it makes sense for these networks to be able to learn the CNF and DNF representations of various boolean expressions, however before we attempt arbatary boolean functions we would like to start with something simple, namely expressions such as NOT, AND, NOR, NAND, XOR and IMPLIES. Results of which where promising, we where able to achieve a low error and from inspecting the weights we could see that the networks where infact 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 ammount of noise is okay and can be pruned out after training we could run into issues if the ammount of noise increases as the number of inputs does.

One other interesting observation to be made is that both the CNF and DNF Netoworks have trouble learning the boolean expression $(a\ XOR\ b)\ AND\ c$. They can't achevice an error lower than 1. Indivually 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 netowrks 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 $a\ OR\ b$. This is a fundimental issue as in practice most problems will be made up of boolean expressions which dont rely on all inputs.

It turns out this problem was not caused by a problem with our LNF Networks but with the weight initilizations, namely they where currently initilised to 0, changing the weights to be randomly distrubuted over the interval [0,1] fixed this problem and allows the LNF Networks to solve all attempted problems so far, along with there weight representations being intepretable. However the

networks seem quite sensitive to there intial conditions, an investgation of best ways to initilise LNF networks would be prudent.

3 LNF Network Learning Issues

Before we can hope to compare peformance, pruning or generalization we must address an issue with learning boolean functions of size 7 or greater.

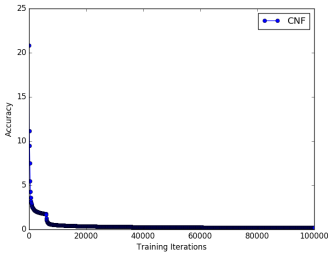

Figure 1: Boolean Formula of size 6

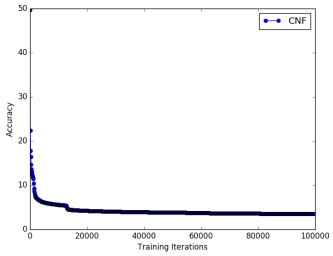

Figure 2: Boolean Formula of size 7

The above graphs demonstraite a CNF Network learning boolean formulas of size 6 and 7. As we move from 6 to 7 we are unable to achieve an error close enough to 0. As we further increase N this only gets worse. Here we will explore possible options for how to fix this

3.1 Weight Initilization

Currently the weights are initilized from the uniform distrubution $[0, 1]$, however one noticible feature of trained networks of lower inputs is that the weights which noise (i.e. once removed reveal the CNF or DNF of the formula) are in the interval $[0, 1]$. Trying different intervals for initilizing the weights could result in better peformance.

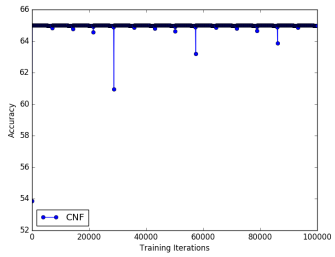

Figure 3: Weight initilization from uniform in [1.0, 3.0]

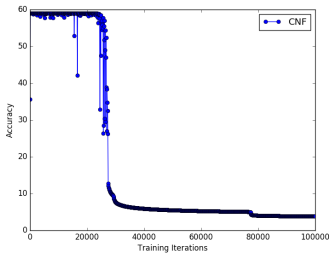

Figure 4: Weight initilization from uniform in [0.5, 1.5]

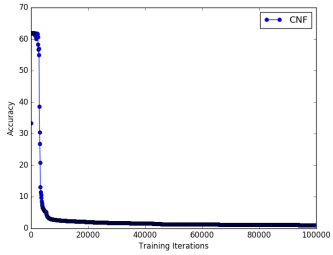

Figure 5: Weight initilization from uniform in [0.0, 2.0]

This significantly improves the peformance of learning boolean functions of 7 inputs for learning all boolean functions of less that 7 inputs aswell. Does further increasing the upper bound on this initilization range keep increasing the peformance? Also does this change in initilization range fix the similar problems with boolean functions of size greater than 7?

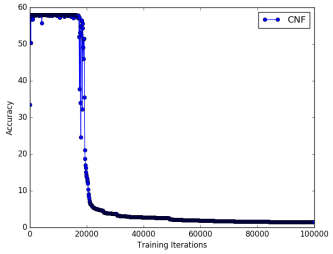

Figure 6: Weight initilization from uniform in [0.0, 3.0]

So further increasing the range does not help peformance. Also our current change dosnt benifit when we move to functions of size 8

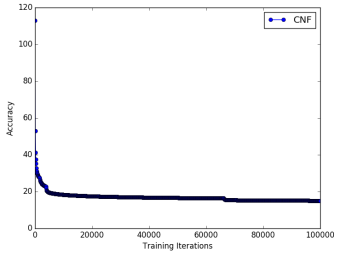

Figure 7: Size 8 Weight initilization from uniform in [0.0, 1.0]

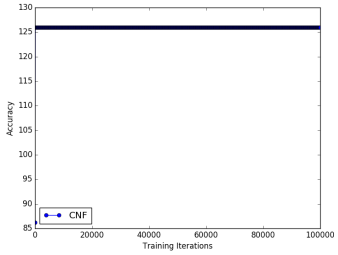

Figure 8: Weight initilization from uniform in [0.0, 2.0]

3.2 Different Optimizers

Currently we are using Gradient Descent to optimize our paramaters, however there are other optimization techequics that could be more suitible our networks. Such as Adam or RMSProp.

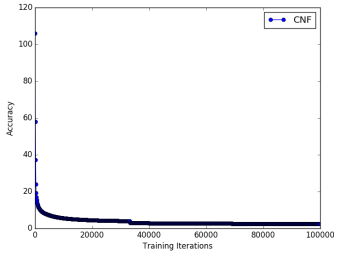

Figure 9: Size 8 With ADAM optimizer

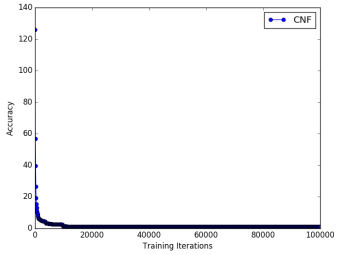

Figure 10: Size 8 With RMS optimizer

RMSProp gives us signigicant improvement at 8 inputs, but are we able to maintain this improvement while we scale up the number of inputs? From experementation as we increase the number of inputs we need a smaller learning rate to achieve a smalle error. This means the larger the number of inputs the larger the training time required, as demonstraited below

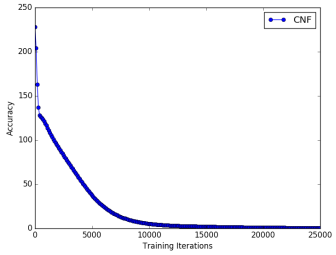

Figure 11: RMSProp with size 9, learning rate 0.0005 and total iterations 20000

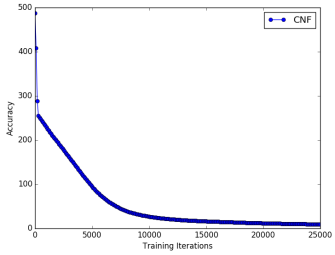

Figure 12: RMSProp with size 10, learning rate 0.0005 and total iterations 25000

4 LNF Network Peformance

We wish to compare the CNF and DNF networks against each other but also against standard perceptron networks. We will take 5 randomly choosen 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 peform significance tests between the three.

5 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 traning 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 slowley remove from the pool of traning 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.

6 LNF Network Rule Extraction

The motivation for this project is to beable to train these networks and esentially extract rules from them. How ever as the networks get larger we see that the networks pickup noise so we must investigate how to remove this noise and then extract rules from the network

6.1 LNF Network Pruning

A nice property of LNF Networks is that there weights directly represent how relevant they are to the network. The larger the weight the more important that perticular 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 respresentation should look like and decide on what an approaite threshold should be. From my investigation I have found that in allcases weights less than 1 represent noise, so this would be a good place to start. We will call this method relevance pruning.

6.1.1 Relevance Pruning

Algorithm 1: Algorythm for peforming 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 ← ∅ O ← ∅
3  for i ← 1 to |H| do
4  |  if output_i < threshold then
5  |  |  H_i ← 0, ..., 0 O_i ← 0
6  |  end
7  |  else
8  |  |  H_i ← PRUNE(hidden_i, threshold) /* PRUNE simply sets any
   |  |     value less than threshold to 0                    */
9  |  |  O_i ← output_i
10 |  end
11 |  return H, O
12 end
```

It should be noted that this algorythm dosnt account for byases being in the weight vectors, if they are some indicies 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] Laurenson, J. Learning logical activations in neural networks, 2016.