Neural Architectures for Boolean Function Synthesis

Aditya Kanade, Chiranjib Bhattacharyya, Deepak D'Souza, Ravi Raja, and Stanly Samuel

***Abstract*— Boolean Function Synthesis is a fundamental problem in computer science with lots of different applications. The state of the art tool is able to solvel 356 out of 609 benchmarks, leaving room for improvement. We propose to use a specialized Neural Network called Gated Continuous Logic Network to synthesize the formulae that satisfies the given specification. Our expectation from using a Neural Network is two fold: 1. To beat the state-of-the-art tools and 2. To study whether a Neural Network can capture the underlying semantics of the Boolean Formula Specification.**

I. Introduction

The problem of Boolean Function Synthesis is a well known problem in computer science and has been in the limelight for past decade. Recently, synthesis of Boolean functions has found its applications in wide range of areas which includes reactive strategy synthesis [1], certified QBF-SAT solving [3], automated program synthesis ([9], [10]), circuit repair and debugging [6] and the likes. This has motivated the community to develop practically efficient algorithms for synthesizing Boolean functions. Latest tool Manthan [8] claims to have beaten all the other state of the art tools by a margin of 76 benchmarks. Manthan uses a Decision Tree based Learning approach to generate the skolem functions satisfying the given specification.

Problem Statement: Formally, given a Boolean specification $F(X, Y)$ between set of inputs $X = x_1, ..., x_n$ and vector of outputs $Y = \langle y_1, ..., y_m \rangle$, the problem of Skolem function synthesis is to synthesize a function vector $\psi(X) = \langle \psi_1(X), ..., \psi_m(X) \rangle$ such that $\exists Y F(X, Y) \equiv F(X, \psi(X))$. We refer to ψ as the Skolem function vector and ψ_i as the Skolem function for y_i.

Gated Continuous Logic Network or GCLN has been earlier used in learning Non Linear loop invariants [5]. We leverage its power for our problem such that it generates the required skolem funciton.

Contribution: In this report, we discuss the details of how GCLN is employed to synthesize boolean formulae. We also discuss various approaches for solving Boolean Function Synthesis using Neural Network. The major topics are:

- Sampling Strategy.
- A CNF based GCLN architecture.
- 4 different problem formulations and their performance on toy examples
- Future Directions

Brainstormed under the helpful guidance of Aditya Kanade, C. Bhattacharyya, and D. D'Souza

Aditya Kanade, C. Bhattacharyya, D. D'Souza, R. Raja and S. Samuel are with IISc, Bengaluru, India.

II. Motivation

As the existing state of the art Manthan solves only 356 out of 609 benchmarks, we aim to beat it in terms of number of benchmarks solved. Apart from this we also aim to find out answer to the question: How accurately a Neural Network can understand the semantics of a Logical formula?

III. Background

Basic Fuzzy Logic: Basic Fuzzy Logic (BL) is a relaxation of first-order logic that operates on continuous truth values on the interval [0, 1] instead of on boolean values. BL uses a class of functions called *t-norms* ($\otimes$), which preserves the semantics of boolean conjunctions on continuous truth values. Formally, a t-norm is defined $\otimes : [0, 1] \times [0, 1] \to [0, 1]$ such that:

- $\otimes$ is consistent for any $t \in [0, 1]$:

$$t \otimes 1 = t \qquad t \otimes 0 = 0$$

- $\otimes$ is commutative and associative for any $t \in [0, 1]$:

$$t_1 \otimes t_2 = t_2 \otimes t_1 \quad t_1 \otimes (t_2 \otimes t_3) = (t_1 \otimes t_2) \otimes t_3$$

- $\otimes$ is monotonic (non decreasing) for any $t \in [0, 1]$:

$$t_1 \leq t_2 \implies t_1 \otimes t_3 \leq t_2 \otimes t_3$$

BL additionally requires that t-norms be continuous. *T-conorms* ($\oplus$) are derived from t-norms via DeMorgan's law and operate as disjunctions on continuous truth values, while negations are defined $\neg t := 1 - t$.
Three widely used t-norms that satisfy the requirements are the Lukaseiwicz t-norm [7], the Godel t-norm [2] and the product t-norm [4]. Each t-norm has a *t-conorm* associated with it (denoted $\oplus$), which can be

	Lukaseiwicz:	Godel:	Product:
tnorm ($\otimes$)	$max(0, t+u-1)$	$min(t,u)$	$t*u$
tconorm ($\oplus$)	$min(t+u,1)$	$max(t,u)$	$t+u-t*u$

TABLE I

considered as logical disjunction. Given a t-norm $\otimes$, the t-conorm can be derived with DeMorgan's law: $t \oplus u \triangleq \neg(\neg t \otimes \neg u)$. Table I shows the formule for tnorms and tconorms.

Gated t-norms and gated t-conorms: Given a classic t-norm $T(x,y) = x \otimes y$, we define its associated gated t-norm as

$$T_G(x,y;g_1,g_2) = (1*(1-g_1)+x*g_1) \otimes (1*(1-g_2)+y*g_2)$$

Here $g_1, g_2 \in [0,1]$ are gate parameters indicating if x and y are activated, respectively.

Gates g_1, g_2 are learnt from Neural Network.
Given a threshold T let,

$$g_i^{'} = \begin{cases} 1 & g_i > T \\ 0 & otherwise \end{cases}$$

$$T_G(x,y;g_1^{'},g_2^{'}) = \begin{cases} x \otimes y & g_1^{'}=1 \text{ and } g_2^{'}=1 \\ x & g_1^{'}=1 \text{ and } g_2^{'}=0 \\ y & g_1^{'}=0 \text{ and } g_2^{'}=1 \\ 1 & g_1^{'}=0 \text{ and } g_2^{'}=0 \end{cases}$$

$(1+g_1(x-1))$ gives a convex combination of 1 and x for the values of g_1 and $g_2 \in (0,1)$. Similarly, for $(1+g_1(y-1))$

Using DeMorgan's laws $x \otimes y = 1-((1-x) \otimes (1-y))$, we define gated t-conorms as

$$T_G^{'}(x,y;g_1,g_2) = 1-((1-g_1*x) \otimes (1-g_2*y))$$

,

and has following property -

$$T_G^{'}(x,y;g_1^{'},g_2^{'}) = \begin{cases} x \otimes y & g_1^{'}=1 \text{ and } g_2^{'}=1 \\ x & g_1^{'}=1 \text{ and } g_2^{'}=0 \\ y & g_1^{'}=0 \text{ and } g_2^{'}=1 \\ 0 & g_1^{'}=0 \text{ and } g_2^{'}=0 \end{cases}$$

Continuous Logic Network (CLN): CLN's are based on parametric relaxation Logical formulas that maps the logical formulation from boolean first order logic to BL. The model defines the operator $\mathcal{S}$. A quantifier-free boolean formula $F : X \to True, False$, $\mathcal{S}$ maps it to a continuous function $\mathcal{S}(F) : X \to [0,1]$. In order for the continuous model to be both usable in gradient-guided optimization while also preserving the semantics of boolean logic, it must fulfill three conditions:

1) It must preserve the meaning of the logic, such that the continuous truth values of a valid assignment are always greater than the value of an invalid assignment: $(F(x) = True \wedge F(x') = False) \implies \mathcal{S}(F)(x) > \mathcal{S}(F)(x')$
2) It must be must be continuous and smooth (i.e. differentiable almost everywhere) to facilitate training.
3) It must be strictly increasing as an unsatisfying assignment of terms approach satisfying the mapped formula, and strictly decreasing as a satisfying assignment of terms approach violating the formula.

$\mathcal{S}$ is constructed as follows to satisfy these requirements. The logical relations $\wedge, \vee, \neg$ are mapped to their continuous equivalents in BL:

$$Conjunction : \mathcal{S}(F_1 \wedge F_2) \triangleq \mathcal{S}(F_1) \otimes \mathcal{S}(F_2)$$

$$Disjunction : \mathcal{S}(F_1 \vee F_2) \triangleq \mathcal{S}(F_1) \otimes \mathcal{S}(F_2)$$

$$Negation : \mathcal{S}(\neg F) \triangleq 1 - \mathcal{S}(F)$$

For Gated CLN, we use gated t-norms and gated t-conorms.

IV. Theorems

Theorem 1: For any Boolean Formula F, there exists a CLN model M, such that

$$\forall x, 0 <= M(x) <= 1 \tag{1}$$

$$\forall, F(x) = True \iff M(x) = 1 \tag{2}$$

$$\forall, F(x) = False \iff M(x) = 0 \tag{3}$$

In Boolean Function Setting, formula F as defined as follows:

$$F :\to p | F_1 \vee F_2 | F_1 \wedge F_2$$

where p is either x or $\neg$ x.

We prove this theorem by structural induction on F assuming the model vanilla CLN.

a) Atomic Case: : F is simply a proposition p. CLN constructed for the atomic case would consist of only one neuron which would give us back the inputs.

$\therefore$ By construction, $\forall x F(x) = True \iff M(x) = 1$

Since M(x) is identity and p is either 0 or 1, $0 \leq M(x) \geq 1$

b) Disjunction Case: : $F = F_1 \vee F_2$. Assume that F_1 and F_2 hold the induction hypothesis.

$\therefore$ F_1 can be represented by M_1 and F_2 can be represented by M_2

s.t. (F_1, M_1) and (F_2, M_2) satisfies equations 1, 2, and 3.

Now let p_1 and p_2 are the ouput nodes of M_1 and M_2. We add a final output node $p = p_1 \oplus p_2$. So, $M(x) = M_1(x) \oplus M_2(x)$

since $\oplus$ is continuous and M_1, M_2 follows induction hypothesis, M(x) is continuous as well.

$\therefore 0 \leq M(x) \geq 1$

Forward Implication: If F(x) is True then either $F_1(x)$ or $F_2(x)$ is true. Without loss of generality let's assume $F_1(x) = True$.

So, from induction hypothesis we know that $M_1(x) = 1$.

Using property of tconorm $t \oplus u \leq t^{'} \oplus u$ for $t \geq 0$ and $t \leq t^{'}$

Also, from induction hypothesis, $M_2(x) \geq 0$

we get, $0 \oplus M_1(x) \leq M_2(x) \oplus M_1(x)$

$0 \oplus M_1(x) = M_1(x)$ [By definition of tconorm]

$\therefore M_1(x) \leq M(x) \leq 1$

and we know that M·1(x) = 1 by induction hypothesis

$$\therefore M(x) = 1$$

Backward Implication: We have M(x) = 1

$\iff M_1(x) \oplus M_2(x) = 1$

$\iff 1 - (1 - p_1) \otimes (1 - p_2) = 1$

$\iff (1 - p_1) \otimes (1 - p_2) = 0$

from induction hypothesis, we know that either M·1 = 1 or M·2 = 1. Without loss of generality, let $M_1 = 1$

$\iff p_1 = 1$

$\iff F_1(x) = True$

as $F(x) = F_1(x) \vee F_2(x)$, we get $F(x) = True$

Similarly, we can also prove for the Conjunction case.

V. Workflow

Figure 1 shows the general workflow of our approach towards Boolean Function Synthesis. 1. Get the continuous equivalent of boolean specification, 2. Perform Sampling over input and output variables to obtain the trainig data, 3. Train GCLN, and 4. Extract Boolean formula from the learnt Network

x:	y:	XOR(x, y) (thresholded value)
0.9	0.2	1
0.85	0.01	1
0.3	0.4	0

TABLE II
Example Samples for XOR(x, y) with threshold = 0.7 and Product t-norm

A. Sampling Strategy

Figure 2 describes the sampling pipeline that we have implemented to generate the training data.

Random Sampling Strategy I: In this we sample uniformly at random for input and output variables in the range $[0, 1]$. These random samples are then supplied to the continous mapping of relational specification (F). Output of F is then thresholded to get binary values. The ones that make F = 1 are taken to be positive samples.

Random Sampling Strategy II: In this we collect positive samples as explained above and then add equal number of negative samples (F = 0) as well in the dataset.

Correlated Sampling: In this strategy, we first sample the input variables. Output variables are conditioned on input variables. This may help to capture the correlation betweenn input and output variables. we keep both positive and negative samples.

Bias Sampling (Manthan): Takes $F(X, Y)$ as input and returns a subset of satisfying assignments of F. Once these samples are generated we add noise to the data with range 0 to 0.2 and 0.8 to 1, to generate fractional sampling for training.

Table II shows an example of random sampling.

B. GCLN Architecture

Figure 3 shows the generic architecture that we use for predicting the skolem function from relational specifications. It consists of 3 layers viz. Input Layer, Disjunction Layer, and Conjunction Layer. After each layer Gates are applied except for the final Conjunction Layer. These Gates are the trainable weights of the neural network. More details below:

Input Layer: This layer is of shape 2N x 1, where N is the number of input variables in the given specification. Along with positive variables, we also consider their negations and include them in the input vector. For e.g. if the input variables are i1 and i2 then the input vector would contain $[i1, i2, \neg i1, \neg i2]$.

Gates G1: G1 is of shape 2N x K (K = No. of clauses and 2N = Size of each clause). These Gates decides which input variables to be selected in each clause.

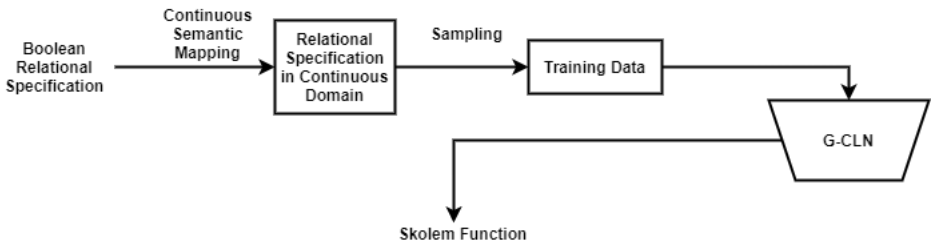

Fig. 1. Workflow

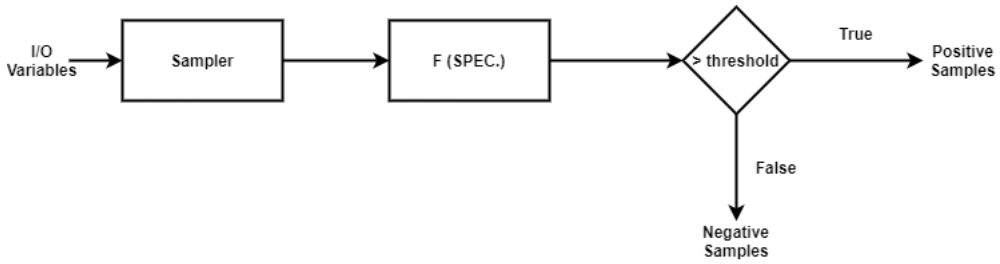

Fig. 2. Sampling Pipeline

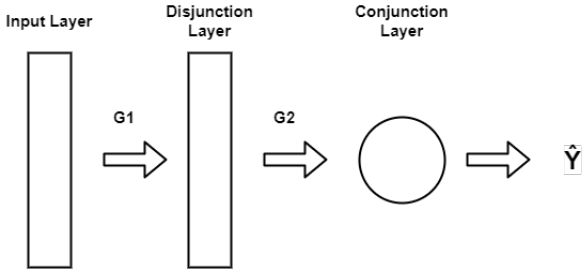

Fig. 3. GCLN Architecture

Disjunction Layer: This layer takes gated input variables. Shape of this layer is K x 1, where K is the maximum number of clauses that could possibly be present in the final solution. K is a hyperparameter and can be tuned.

Gates G2: G2 is of shape K x M, where M is the number of output variables. It selects the clauses for the skolem function.

Conjunction Layer: This layer takes input the gated output of the Disjunction Layer. It is of shape 1 x 1.

Due to the design of the network, the formula that we extract is in CNF form.

Running Example: Figure 4 shows a running example of XOR(i1, i2, i3), where i1 and i2 are input variable and i3 is the output variable.

VI. Proposed Solution

The problem of Boolean Function Synthesis can be posed as a learning problem. In an abstract sense, Observations O corresponds to boolean relational formula that serves as the specification for synthesizing skolem function.

let, F = Relational Specification (Observations)

ψ = Skolem Function Vector (Target)

And the aim is to learn a hypothesis function $'h'$

s.t. $h(F;\hat{\theta}) = \hat{\psi}(X)$, where θ are learneable parameters of the model and X denotes the set of input variables from the observations.

For a given dataset $\mathcal{D} = (f_i, \psi_i)|1 \leq i \leq N$, we want to minimize the expected loss between $\hat{\psi}(X) and \psi$

$$\text{Expected Loss: } E(l(\hat{\psi}(X), \psi)) \approx \frac{1}{N}\sum_{i=1}^{N} l(\hat{f}_i, \psi_i)$$

$$\hat{\theta} = \underset{\theta}{argmin}\frac{1}{N}\sum_{i=1}^{N} l(\hat{\psi}_i, \psi_i)$$

We model h using GCLN V-B. The parameters in this network are the Gates ($\theta = (G_1, G_2)$) which acts like a switch for the input variables (G_1) or clauses (G_2).

As ψ is not known to us, training a neural net over (F, ψ) is not possible. Therefore, we perform sampling over the boolean variables in formula F and mark the set of input variables (X) as the features and the set of output variables (Y) as the ground truth labels. Sampling

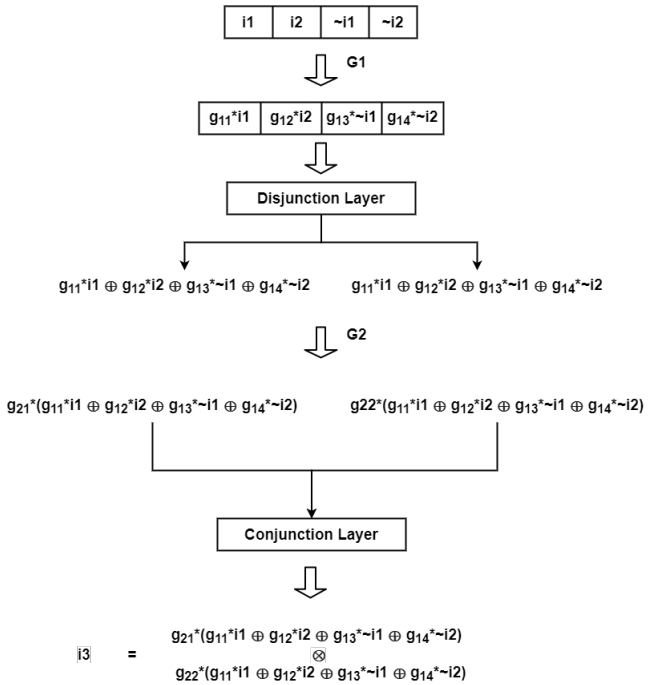

Fig. 4. Example run of GCLN over specification XOR(i1, i2, i3). Input Variable = i1, i2 and Output Variable = i3

strategies are discussed in detail in section V-A.

Sampling is done such that it captures the relation between X and Y. We exploit this fact and learn a function mapping $\psi : X \rightarrow Y$, wihch is our end goal.

We have implemented 4 different algorithms for solving this problem. We discuss these in the next section VII .

VII. Problem Formulations

As of now we have come up with 4 different formulations for training the network. In this section we discuss those formulations. We also discuss their performance over a small set of toy examples.

A. Regression

This uses Random Sampling Strategy I. Training data contains only positive samples. Output variables (Y) of the specification are the target variables while the input variables (X) are the features. As the target values are real, most intuitive thing is to do regression. The GCLN model regresses over the output variable. Loss function used here is mean square error loss.

Figure 5 pictorially describes the algorithm.

Intuitive Explanation for its Working: With the given set up, model learns to predict Y's given X's. That is it learn a function mapping from X to Y and this is the definition of the desired function to be synthesize. Now because we had sampled only samples which made the specification True, the function learnt for Y will output only those truth values of Y which would make the specification True.

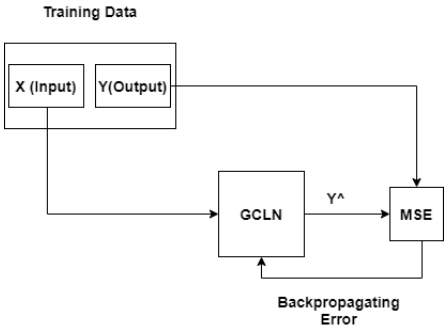

Fig. 5. Regression Formulation

B. Classification Problem - 1

Using Random Sampling Strategy I we sample the training data. In this case, the sampled real valued output variables (Y) are converted to binary (Y_{bin}) based on given threshold. Now we can consider Y_{bin} as the class labels and learn a classifer over them. Loss function used in this case is Binary Cross Entropy Loss.

Figure 6 describes it pictorially.

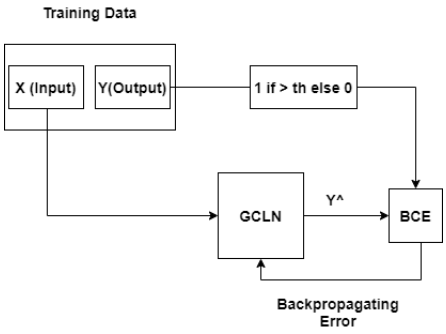

Fig. 6. Classification - 1

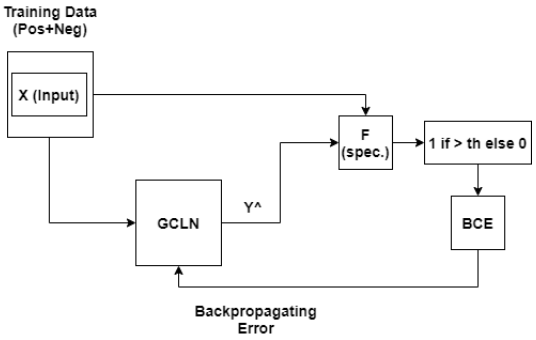

Fig. 7. Classification - 2

Intuitive Explanation for its Working: If the model learns a best fit separating hyperplane, it will predict class labels Y_{bin} for X. Which essentially means that the classifier is a mapping from X to Y_{bin}. As the data consists of only positive examples, the classifier would represent the required function.

C. Classification Problem - 2

We use Random Sampling Strategy II. Training data is (X, F_{out}) pairs, where, F_{out} is the output of F for the sampled input variables (X) and output variables (Y). i.e. X is features and F_{out} are the class labels. Once F_{out} is computed, Y is discarded. While training we take the output of GCLN model $\hat{Y}$ and the input variables X from trainig data and compute the value of $\hat{F_{out}}$ over them. In this set up, $\hat{Y}$ is the latent variable that the model learns to predict. Loss function used is Binary Cross Entropy Loss over $\hat{F_{out}}$ and F_{out} Figure 7 descibes this pictorially.

Intuitive Explanation for its Working: Here the model learns to predict correct F_{out}. For that it first learns the latent variable Y given only X as its input. So, the model would learn a classifier that tells which Y to output such that it correctly classifies the F_{out}. It means that model will act as a mapping from X to Y such that it matches F_{out} and $\hat{F_{out}}$. As the output of classifier mimics Y for given X, it should represent the required skolem function.

D. Classification Problem - 3

This is same as VII-C except the sampling strategy used here is correlated sampling explained in V-A.

VIII. Extraction Algorithm

Number of input variables = n, Number of output variables = m, Maximum clauses = K Gates in Disjunction Layer - G1 ($2n \times mK$) Gates in Conjunction Layer - G2 ($mK \times m$)

1. Input Vector I = [input variables, (input variables)] if there are n input variables then Input Vector I, would be of size $2n \times 1$
2. Apply gates G1 and construct mK clauses each of size 2n where m is the number of output variables.
3. Apply gates G2 and select suitable clauses for each output variables. First K clauses corresponds to the first output variable, second K clauses corresponds to the second output variable, and so on.
4. Finally, we have boolean formula for each output variable in terms of input variables.

IX. Results

The output variable in the specification is replaced with the extracted formula from the network and checked for validity using z3Py Solver. Figure 8 shows the preliminary results over 5 toy problems. The results are not very impressive at the moment but the hope is with intelligent sampling and better training procedure, this can be improved.

X. Future Directions

Following is a consolidation of the discussions held in previous meetings.

A. Sampling Strategy

Seed based Sampling: Instead of going for Random Sampling, we can use a few seed examples from the SAT solver. Once we have the seed examples, we can sample around these examples to get the training data. This would give more accurate training data faster.

Simulated Annealing: This is another way to avoid Random Sampling. Simulated Annealing would be faster and won't require the seed examples as well. This method is based on Energy Functions.

	i_0 ^ i_1 ^ i_2	(i_0 ^ i_1) ^ (i_1 & i_2)	(i_0 & i_1) | (i_2 ^ i_3)	i_0 ^ i_1 ^ i_2 ^ i_3	((i_0 | i_1) ^ (i_2 & i_3)) ^ i_4
	Y = i_2	Y = i_2	Y = i_3	Y = i_3	Y = i_4
Regression	Valid **i2 = ((i_0 | ~i_1) & (i_1 | ~i_0))**	Valid **i2 = ((i_0) & (i_1 | ~i_1))**	Valid **i3 = ((~i_2))**	Valid **i3 = ((i_0 | i_1 | ~i_2) & (i_0 | i_2 | ~i_1) & (i_1 | i_2 | ~i_0) & (~i_0 | ~i_1 | ~i_2))**	Not Valid **i4 = ((i_2 | ~i_0) & (i_2 | ~i_1) & (i_3 | ~i_0) & (i_3 | ~i_1))**
Classification-1	Valid **i2 = ((i_0 | ~i_1) & (i_1 | ~i_0))**	Valid **i2 = ((i_0 | i_1) & (i_0) & (i_1))**	Valid **i3 = ((~i_0 | ~i_1 | ~i_2) & (~i_0 | ~i_1) & (~i_2))**	Not Valid **i3 = ((i_0 | i_1 | ~i_2) & (~i_0 | ~i_1 | ~i_2))**	Not Valid **i4 = ((i_2 | i_3 | ~i_0 | ~i_1) & (i_2 | ~i_0) & (i_2 | ~i_1) & (i_3 | ~i_0) & (i_3 | ~i_1))**
Classification-2	Valid **i2 = ((i_0 | i_1 | ~i_0 | ~i_1) & (i_0 | i_1 | ~i_0) & (i_0 | ~i_1) & (i_1 | ~i_0 | ~i_1) & (i_1 | ~i_0))**	Valid **i2 = ((i_0) & (i_1 | ~i_0) & (i_1))**	Not Valid **i3 = ((i_0 | i_1 | i_2) & (~i_0 | ~i_1 | ~i_2))**	Valid **i3 = ((i_0 | i_1 | ~i_0 | ~i_2) & (i_0 | i_1 | ~i_2) & (i_0 | i_2 | ~i_0 | ~i_1) & (i_0 | i_2 | ~i_0 | ~i_2) & (i_0 | i_2 | ~i_1) & (i_1 | i_2 | ~i_0) & (~i_0 | ~i_1 | ~i_2))**	Not Valid **i4 = ((i_0 | i_1 | ~i_2 | ~i_3) & (i_2 | ~i_0 | ~i_1) & (i_3 | ~i_0 | ~i_1) & (i_3 | ~i_0) & (~i_0 | ~i_1 | ~i_2))**
Classification-3	Not Valid **i2 = ((i_0 | i_1 | ~i_0 | ~i_1))**	Not Valid **i2 = None**	Not Valid **i3 = ((i_0 | i_2 | ~i_1) & (i_1 | i_2 | ~i_0) & (i_2))**	Not Valid **= ((i_0 | i_1 | i_2 | ~i_0 | ~i_1) (i_0 | i_1 | ~i_0 | ~i_1 | ~i_2) & (i_0 | i_1 | ~i_0 | ~i_1) & (i_1 | i_2 | ~i_1 | ~i_2) & (i_2 | ~i_2))**	Not Valid **i4 = ((i_0 | i_1 | ~i_2 | ~i_3) & (i_0 | i_1 | ~i_2) & (i_0 | i_1 | ~i_3) & (i_0 | ~i_2 | ~i_3) & (i_1 | ~i_2 | ~i_3))**

Fig. 8. Results on 5 toy Problems

B. Training

Counter Example Guided Training: If the first output of the model doesn't give us the intended result, we can generate countere examples using a SAT solver and feed that back to the Network through a feedback loop. This will tell exactly where the model is failing.

Loss Function: Instead of computing the loss over model's numerical output, can we compute loss over final formula extracted from the network?
As we don't have final formulae in our dataset, this idea doesn't seem to work as of now.

Hyperparameter Search: Instead of trying out different hyperparameter values manually, we can automate this process. An efficient way to do this is through Hyperparameter Search.

REFERENCES

[1] Alur, R., Madhusudan, P. and Nam, W. Symbolic computational techniques for solving games. *Int J Softw Tools Technol Transfer*, 118–128 (2005).

[2] Baaz, Matthias. Infinite-valued Gödel logics with 0-1-projections and relativizations. In *Gödel'96: Logical foundations of mathematics, computer science and physics—Kurt Gödel's legacy, Brno, Czech Republic, August 1996, proceedings*, pages 23–33. Association for Symbolic Logic, 1996.

[3] Balabanov, V., Jiang, JH.R. Unified QBF certification and its applications. *Form Methods Syst Des*, 2012.

[4] Hájek, Petr and Godo, Lluís and Esteva, Francesc. A complete many-valued logic with product-conjunction. *Archive for mathematical logic*, 35(3):191–208, 1996.

[5] Jianan Yao and Gabriel Ryan and Justin Wong and Suman Jana and Ronghui Gu. Learning Nonlinear Loop Invariants with Gated Continuous Logic Networks. *CoRR*, abs/2003.07959, 2020.

[6] Jo, Satoshi and Matsumoto, Takeshi and Fujita, Masahiro. SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions. In *2012 IEEE 21st Asian Test Symposium*, pages 19–24, 2012.

[7] Łukasiewicz, J. and Tarski, A. *Untersuchungen über den Aussagenkalkül*. 1930.

[8] Priyanka Golia and Subhajit Roy and Kuldeep S. Meel. Manthan: A Data Driven Approach for Boolean Function Synthesis. *CoRR*, abs/2005.06922, 2020.

[9] Solar-Lezama, A. Program sketching. *Int J Softw Tools Technol Transfer*, 15(475–495), 2013.

[10] Srivastava, S., Gulwani, S. and Foster, J.S. Template-based program verification and program synthesis. *Int J Softw Tools Technol Transfer*, 15(497–518), 2013.

XI. Network architectures to represent CNF formulae

In this section, we design three architectures for the boolean function synthesis problem. Each of these architectures are used to learn formulae in conjunctive normal form (CNF). The training phase is guided using an oracle that provides counterexamples to the formulae synthesized using the network. In our setting, the oracle is a verifier for the boolean function synthesis problem. In this problem, for a given function, there could be multiple solutions possible and hence multiple local minima. The choice of minima depends on the counterexamples generated by the counterexample loop.

A. Architecture 1: Single output GCLN

Let the CNF formula be F as shown below:

$$F = \bigwedge_{j=1}^{m} (\bigvee_{i=1}^{2n} (x_i \ . \ g_{ij}) \ . \ g_j)$$

where,
n = number of input variables (i.e., $|V_{in}|$),
m = number of clauses and is bounded by 2^{2n}
x_i = set of variables and their negations (i.e., $2|V_{in}|$), .
g_{ij} = 1 iff the i^{th} variable is used in the j^{th} clause.
g_j = 1 iff the j^{th} clause is used for the output.

The GCLN architecture for $m = 3$ and $V_{in} = \{x\}$ is as follows:

Enter diagram here

The network has three layers where the middle (disjunct) layer is variable in size.

Although this network has a single output, we may be able to synthesize functions for multiple outputs by:
(a) sampling intelligently,
(b)using a dependency guided loop after every function synthesized (not sure how as we do not know how to extract a single correct function separately)

B. Architecture 2: Multiple output GCLN with a shared layer

In this setting, for each of the l output variables in the boolean function synthesis problem, we wish to synthesize a formula F_k such that:

$$F_k = \bigwedge_{j=1}^{m} (\bigvee_{i=1}^{2n} (x_i \ . \ g_{ij}) \ . \ g_{jk}), \quad k \in \{1, ..., l\}$$

where,

l = number of output variables (i.e., $|V_{out}|$),
g_{jk} = 1 iff the j^{th} clause is used for the k^{th} output.
and the rest are as before.

The GCLN architecture for $m = 3$, $V_{in} = \{x\}$, and $V_{out} = \{y_1, y_2\}$ is as follows:

Enter diagram here

As observed, the middle layer is shared between all the l outputs.

C. Architecture 3: Multiple output GCLN without a shared layer

In this setting, we extend the above architecture and assign m nodes to each of the l outputs. Thus, we now have ml nodes in the middle layer.

Thus, F_k is now:

$$F_k = \bigwedge_{j=mk-m-1}^{mk} (\bigvee_{i=1}^{2n} (x_i \ . \ g_{ij}) \ . \ g_{jk}), \quad k \in \{1, ..., l\}$$

where a block B_k for an output k ranges from $mk - m - 1$ to mk, for a given output k, for m clauses.

The GCLN architecture for $m = 3$, $V_{in} = \{x\}$, and $V_{out} = \{y_1, y_2\}$ is as follows:

Enter diagram here

XII. Network architecture to represent DNF formulae

The architectures used to synthesize CNF formulae can also be used to learn formulae in disjunctive normal form (DNF). In such a case, the middle layer would now consist of T-conorms instead of T-norms.

XIII. Possible advantages of our approach over Manthan

- Manthan is constrained to use DNF formulae only whereas we can use both CNF and DNF as the size of the formula is within our reach. $SAT \leq_P 3 - SAT$ tells us that for any formula there exists an equisatisfiable formula in CNF of polynomial size. No such result is known for DNF. All such reductions give an exponentially sized DNF formulae. Can we somehow expect, using this information, that we can generate smaller formulae?