Boolean Function Synthesis using Gated Continuous Logic Network

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 formula F(X, Y) specifying a desired relation between inputs X and outputs Y, we wish to synthesize each output in Y as a function of inputs X such that F(X, Y) is satisfied, whenever possible.

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
- Important meeting discussions

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 considered as logical disjunction. Given a t-norm $\otimes$, the t-conorm can be derived with DeMorgan's law: $t \oplus u \triangleq$

	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

$\neg(\neg t \otimes \neg u)$. 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

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

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

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

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:

$$\begin{aligned} \text{(F(x) = True } \wedge F(x') = False) \\ \implies \mathcal{S}(F)(x) \\ > \mathcal{S}(F)(x') \end{aligned} \quad (1)$$

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.

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

$$Conjunction : S(F_1 \wedge F_2) \triangleq SF_1 \otimes F_2$$

$$Disjunction : S(F_1 \vee_2) \triangleq SF_1 \otimes F_2$$

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

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

IV. 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

A. Sampling Strategy

Figure 2 describes the sampling pipeline that we have implemented to generate the training data. Currently we are using Random Sampling Strategy in which, we sample uniformly at random for input and output variables in the range $[0, 1]$. These random samples are then applied with the continous mapping of relational specification (F). Output of F is then thresholded to get the positive samples.

Apart from Random Sampling, we also use Correlated Sampling Strategy for one of the problem formulation. In this strategy, we first sample the input variables and the output variables are conditioned on input variables.

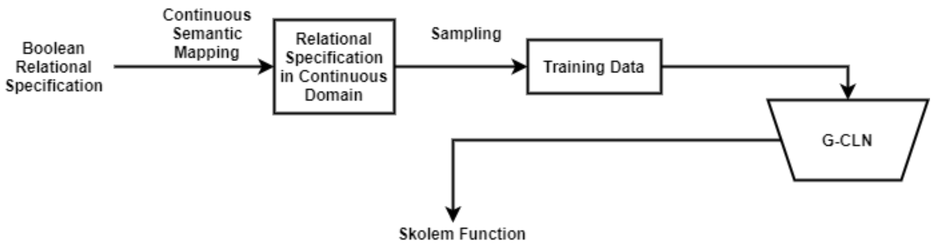

Boolean Relational Specification
Continuous Semantic Mapping
Relational Specification in Continuous Domain
Sampling
Training Data
G-CLN
Skolem Function

Fig. 1. Workflow

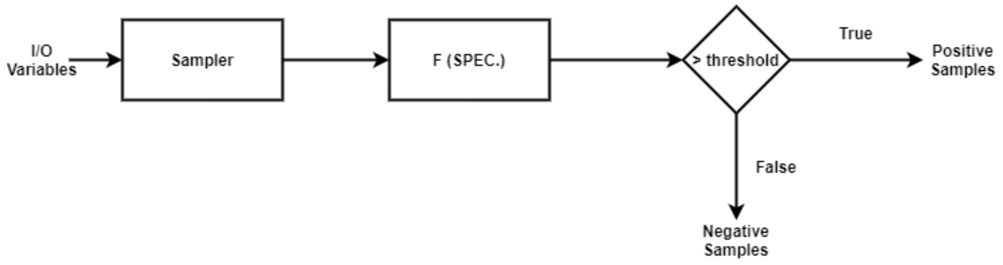


Fig. 2. Sampling Pipeline

x:	y:	XOR(x, y)
0.9	0.2	0.72
0.85	0.01	0.84

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

This may help to capture the correlation betweenn input and output variables.

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.

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 1 i.e. each node represents one of the clause. Here we decide upon which clause to keep as part of the solution 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.

V. 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. Formulating it as Regression Problem

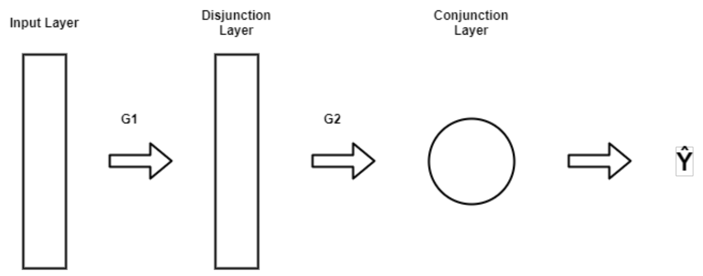
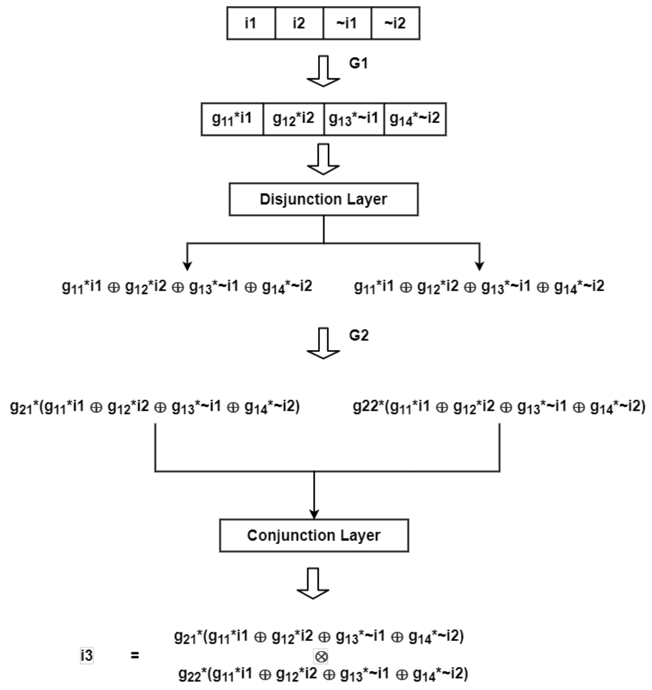


Fig. 3. GCLN Architecture



i1
i2
~i1
~i2
G1
g11*i1
g12*i2
g13*~i1
g14*~i2
Disjunction Layer
g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2
g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2
G2
g21*(g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2)
g22*(g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2)
Conjunction Layer
i3 = g21*(g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2) ⊗ g22*(g11*i1 ⊕ g12*i2 ⊕ g13*~i1 ⊕ g14*~i2)

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

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.