Learning Subject to Constraints via Abstract Gradient Descent

Shiwen Yu, Wanwei Liu, Zengyu Liu, Liqian Chen, Ting Wang YUSHIWEN14@NUDT.EDU.CN
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology, China

Naijun Zhan NJZHAN@PKU.EDU.CN
School of Computer Science and Key Laboratory of High Confidence Software Technology, Peking University & Zhong Guan Cun Lab, China

Ji Wang[*] WJ@NUDT.EDU.CN
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology, China

Editors: G. Pappas, P. Ravikumar, S. A. Seshia

Abstract

Deep learning has made significant advancements and has been successfully used in different areas. However, current deep learning models are based on theories of probability and statistics, thus suffer from adhering to constraints and ensuring assurances. In contrast, symbolic methods inherently own rigidity but with low efficiency and high cost. In this paper, we propose a novel symbolic learning architecture which can deal with data fitting and the logic satisfaction uniformly. The key idea is to treat logic formulas as discrete functions that can be optimized through Abstract Gradient Descent, a discrete optimization method based on backward abstract interpretation. The efficiency of our approach is illustrated by training a symbolic neural network with 8,542 parameters that can accurately recognize 70,000 handwritten digits. Experiments indicate that the trained model not only fits the data well but also adheres to key properties such as robustness, invariance to zooming, translation, resistance to noise background, and so on.

Keywords: Deep Learning, Formal Method, Abstract Interpretation, Neurosymbolic, Optimization

1. Introduction

Deep learning, as a pivotal innovation in artificial intelligence, has significantly altered the landscape of computational learning as well as data processing and is evident through groundbreaking accomplishments in diverse areas. Despite these big successes, deep learning still faces many intrinsic challenges, e.g., predictability, consistency, susceptibility to min-max attacks, and generalization, just to name a few.

The classical deep learning problem aims at data-fitting tasks. Given a function (i.e., a neural network) $F(x, p)$ with parameters p and input x, as well as a data set D, where each record is denoted as $\langle x, y\rangle \in D$ with y as the label, the task of data fitting is to find p^* such that the trained model fits as many records $\langle x, y\rangle \in D$ as possible. It can be typically formalized as the following optimization problem:

$$\min_p \sum_{\langle x,y\rangle \in D} loss(F(x,p), y) \tag{1}$$

[*] Corresponding Author.

where the function $loss(a, b)$ measures the 'difference' between a and b, the bigger the more different, and $loss(a, b) = 0$ indicates $a = b$. A typical instance of $loss(a, b)$ is the mean square loss.

In safety or mission-critical applications, the learning components are required to not only fit the data as accurately as possible but also satisfy certain critical properties often represented by a first-order logic formula, such as $\forall x.\ \ C(x, p)$. By introducing the constraint $\forall x.\ \ C(x, p)$ on the parameters, the learning task becomes the following constraint optimization problem:

$$\min_{p\ s.t.\ \forall x.\ \ C(x,p)} \sum_{\langle x,y\rangle \in D} loss(F(x, p), y) \tag{2}$$

The above learning problem plays an important role in trustworthy machine learning. For example, the robustness property can be specified as $\forall x.\ (||x, x_0|| < t) \to (F(x, p) = F(x_0, p))$, where $||\cdot||$ stands for the L_2 Norm, x_0 for the original input, and x for the actual input subject to a disturbance within the norm ball centered at x_0.

Solving the problem in Equation (2) is challenging and unsolvable in general. Existing approaches (e.g., Petersen et al. (2022); Li et al. (2023)) use *relaxation* to transform the logic formula C into a continuous loss function L_c such that $C(p)$ is satisfied iff $L_c(p) = 0$. Hence, the problem in Equation (2) can be reduced to the following unconstrained optimization problem:

$$\min_{p} \sum_{\langle x,y\rangle \in D} loss(F(x, p), y) + L_c(p) \tag{3}$$

However, the above relaxation of discrete logic formulas to continuous loss functions gives rise to many problems, such as approximation error and non-convexity, making gradient descent powerless. Consequently, the logic constraints seem to play auxiliary enhancements of the generalization and anti-attacking abilities of the trained models rather than the assurance.

To avoid the aforementioned problems, we utilize *abstract gradient descent* (AGD) Yu et al. (2024) to tackle optimization of discrete functions. It combines efficient gradient descent with formal search in an backward abstract interpretation way Peled et al. (2023). Instead of relaxation, we can transform Equation (2) as a min-max optimization problem as follows:

$$\min_{p} \max_{x} \left(\neg C(p, x) + \sum_{\langle x',y\rangle \in D} loss(F(x', p), y)\right) \tag{4}$$

where $\neg C(p, x)$ in this equation is treated as a discrete boolean function with p and x as input parameter. The optimum of the objective function being 0 implies that there exists p^* not only satisfying the constraint but also perfectly fitting the dataset.

However, if the optimum is not 0, $C(p)$ is not guaranteed to be satisfied. In this paper, we argue that satisfying the logic constraint is more critical than fitting the dataset. Therefore, we designed a trade-off optimization strategy that iteratively find the 'legal sub-optimal' values of the parameters p.

Instead of being caught up in cumbersome details (which is in Section 4.3), we use an example to illustrate this process.

Example 1 (A Motivating Example) *Consider the constraint optimization problem:*

$$\min_{p\ s.t.\ \forall x.\ (0.4<x<0.75)\to(0.5<x*p<1)} (\frac{2}{p} + p) \tag{5}$$

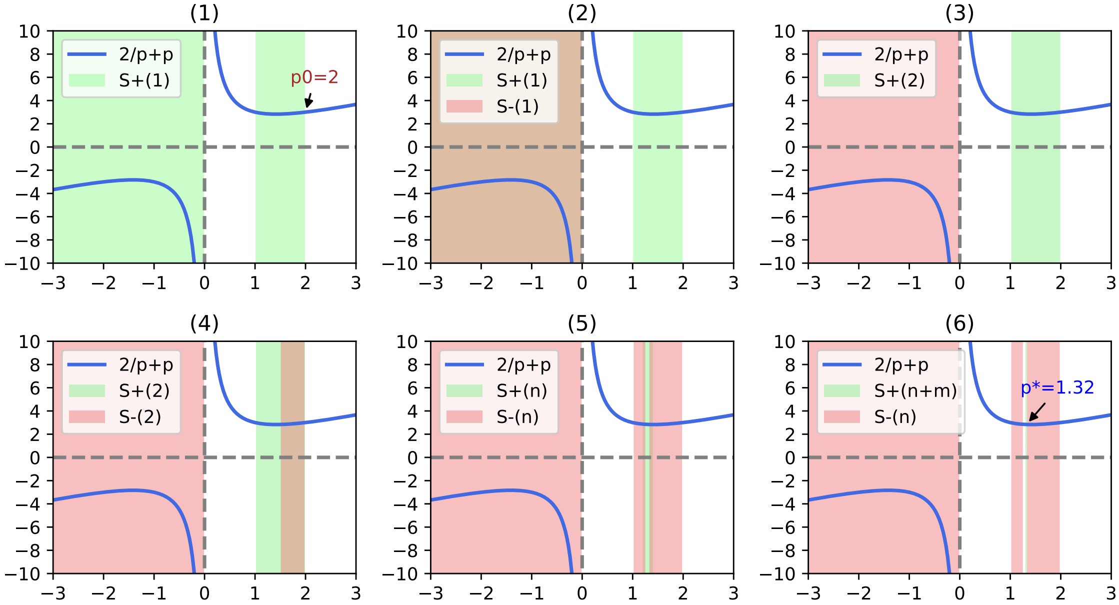


Figure 1: The min-max optimization process of the motivating example, where the pink range represents S^{-}, the green range represents S^{+}, the X-axis represents the value of p, and the Y-axis denotes the minimization objective value.

It can be transformed to the following discrete min-max optimization using our approach:

$$\min_{p} \max_{x} (\frac{2}{p} + p) + (\neg(0.4 < x < 0.75 \ \rightarrow \ 0.5 < x * p < 1)) \tag{6}$$

We show how to solve it using our approach step by step as follows (also see Figure 1):

1. At the beginning step, we set an empty set $S^{-}_{(0)}$ to include the values that p is forbidden to be, at step i. Suppose the initial value of p is $p = 2$, the initial objective value of the left part $(\frac{2}{p} + p)$ is: 3.

2. The minimization objective $(\frac{2}{p} + p)$ can be broken into a sequence of descending steps. The next step, solved by AGD Yu et al. (2024), is $p \in [-\infty, -0.01] \cup [1.01, 1.99]$ (one can see this through the Figure 1.(1)). We use a set $S^{+}_{(1)}$ to record $[-\infty, -0.01] \cup [1.01, 1.99]$, which includes the values that we wish p to be in the next step. For any $p \in S^{+}_{(1)}$, we have $\frac{2}{p} + p < 3$, descending the objective.

3. The maximization of the discrete constraint $\neg(0.4 < x < 0.75 \rightarrow 0.5 < x * p < 1)$ results in $x \in [0.41, 0.74] \wedge p \in [-\infty, -0.01]$ by AGD (i.e. any x and p in this range can make the target value maximum). We let $S^{-}_{(1)} = S^{-}_{(0)} \cup [-\infty, -0.01]$, since we do not wish p in this range.

4. We redo the minimization by AGD with p not be in $S^{-}_{(1)} = [-\infty, -0.01]$, and obtain $S^{+}_{(2)} = [1.02, 1.98]$ (Figure 1.(3)). Then, we redo the maximization with p be in $S^{+}_{(2)}$, and obtain $x \in [0.67, 0.74] \wedge p \in [1.51, 1.98]$. So, $S^{-}_{(2)}$ is assigned as $[-\infty, -0.01] \cup [1.51, 1.98]$ (Figure 1.(4)).

5. We repeat the above process till the optimum of $\max_x(\neg(0.4 < x < 0.75 \ \rightarrow \ 0.5 < x * p < 1))$ when $p \in S^{+}_{(n)}$ is 0, where $S^{-}_{(n)} = [-\infty, -0.01] \cup [1.02, 1.25] \cup [1.33, 1.98]$ (Figure 1.(5)).

6. We continue $\min_p (\frac{2}{p} + p)$ with $p \in S^+_{(n)}$ till the sub-optimum: $p^* = 1.32$ (Figure 1.(6)).

As a comparison, if we apply gradient descent initialized with $p = 2$, we will end up with $p = \sqrt{2}$, which fails to satisfy the constraint.

With AGD, the objective function F does not require continuity, which gives more flexible architecture for the model, e.g., allowing logic connectives and predicates, unifying the neural network structure of F and the symbolic expression, and so on. This largely generalizes deep neural networks and their training methods. We name the proposed general learning framework as *Symbolic Neural Network* (SNN). Its overall architecture is depicted in Figure 2.

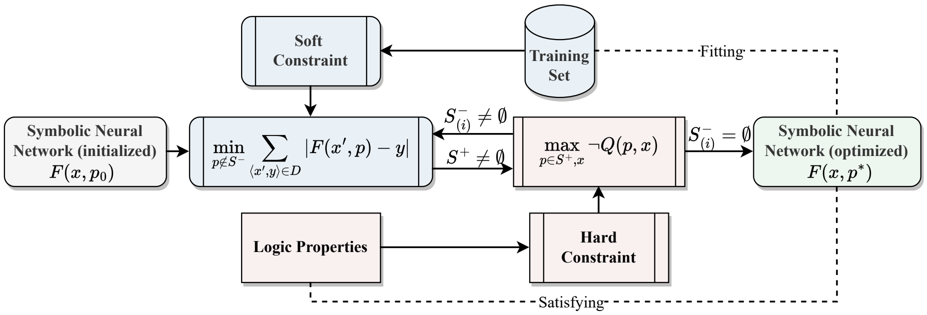

Soft Constraint
Training Set
Fitting
Symbolic Neural Network (initialized) $F(x, p_0)$
$\min_{p \notin S^-} \sum_{\langle x', y \rangle \in D} |F(x', p) - y|$
$S^-_{(i)} \neq \emptyset$
$S^+ \neq \emptyset$
$\max_{p \in S^+, x} \neg Q(p, x)$
$S^-_{(i)} = \emptyset$
Symbolic Neural Network (optimized) $F(x, p^*)$
Logic Properties
Hard Constraint
Satisfying

Figure 2: The first step is to construct a parameterized symbolic neural network $F(x, p_0)$. The dataset fitting task is transformed into a soft constraint optimization, and the logical constraint is transformed into a hard constraint optimization. The two optimizations proceed interactively until we can only find an empty range of x to negate the hard constraint, we obtain the optimal value of p in the range and output the resulting symbolic neural network $F(x, p^*)$.

The rest of this paper is organized as follows: Section 2 introduces the related work; Section 3 recaps basic notions and gives a brief introduction of AGD. Section 4 presents the novel symbolic learning framework of SNN. Section 5 reports experiments. We conclude this work in Section 6.

2. Related Work

The verification of deep learning models Gehr et al. (2018); Wang et al. (2021); Katz et al. (2017); Wong and Kolter (2018) becomes increasingly vital as these models are applied in more and more safety-critical domains. A seminar work in Gehr et al. (2018) proposes a scalable approach using abstract interpretation to certify neural network safety. Wang et al. introduces an efficient method for robustness verification, addressing scalability challenges Wang et al. (2021). Additionally, Katz et al. presents an SMT-based verification for neural networks with ReLU activations, which is a significant step towards handling the verification complexity of nonlinear models Katz et al. (2017). Despite the progress in verification of the trained models, the state of the art needs to scale up for practical deep learning models.

Since the verification of the trained model is difficult, recent works Petersen et al. (2022); Cropper and Dumancic (2022); Wong and Kolter (2018); Li et al. (2023); Giannini et al. (2019); van Krieken et al. (2022) investigate to integrate verification into training process as an alternative. The basic idea is to construct an extra target other than the original fitting target, and integrate them as one single gradient descent optimization task. Wong et al. presents how to construct a robust loss function by building a convex outer approximation Wong and Kolter (2018). Furthermore, some recent works consider more general constraints specified in first order logic language. For example, to exploit fuzzy logic to relax the logic constraint to a differential function and combine it with the original data fitting loss function van Krieken et al. (2022); Li et al. (2023). However, relaxation error and non-convexity keep them from learning verified models. Thus, these integrated methods seem more like a constraint-enhancement to improve the ability of trained models for generalization and anti-attacking. In the experiments, we will compare our method with these methods proposed in Wong and Kolter (2018); Li et al. (2023).

There are other optimization algorithms, known as solving algorithms or search algorithms Barrett et al. (2016, 2021); Krishnan et al. (2020); Cai et al. (2023). SMT solving Barrett et al. (2016) is able to tackle this optimization but with an extremely expensive computing cost, especially for the cases of nonlinear operations and a large number of variables (both are necessary for general learning tasks). As a result, SMT solving is not suitable for this optimization. Moreover, random search methods, such as Monte Carlo Tree Descent Zhai and Gao (2022), try to use a random input sampling process to find the descent direction in a black box style. In contrast, AGD optimization proposed in this work is a white-box method based on abstraction interpretation, that provides the guarantee of the given logical constraints.

Abstraction is an efficient strategy for handling a large amount of concrete states. A famous theoretic framework is abstract interpretation Cousot (1996); Cousot and Cousot (1977); Bertrane et al. (2015); Yin et al. (2020); Miné (2014), which aims to approximate the program semantics in a general way. In order to make the inference process determinable, as well as balance the precision of abstraction and computing cost, a number of abstract domains have been proposed, such as Intervals Brauer et al. (2010), Boxes Gurfinkel and Chaki (2010), Octagons Miné (2006); Chen et al. (2014), and Polyhedra Sankaranarayanan et al. (2006), and so on. In this paper, we leverage Boxes as the abstract domain with lower/upper forward/backward rules that are implemented with GPU support.

3. Preliminaries

3.1. Notations

We consider three simple data types in this paper, i.e., Boolean $\mathbb{B}$, Integer $\mathbb{I}$, and Real $\mathbb{R}$. Conventionally, we treat Boolean type $\mathbb{B}$ as a subtype of Integer $\mathbb{I}$, i.e. only with 0 and 1, and Integer type $\mathbb{I}$ as a subtype of Real $\mathbb{R}$. Further, we use $\mathbb{X}$ to represent one of the three types: $\mathbb{X} \in \{\mathbb{B}, \mathbb{I}, \mathbb{R}\}$.

For a vector $x \in \mathbb{X}^n$, $x[i] \in \mathbb{X}$ represents its ith component. Projecting a set $V \in 2^{\mathbb{X}^n}$ onto the ith dimension is denoted by $V[i:] \in 2^{\mathbb{X}}$, i.e., $V[i:] = \{x[i] | x \in V\}$. Similarly, for a vector of function $f : \mathbb{X}^n \to \mathbb{X}^m$, projecting f onto the ith dimension is denoted as $f_i : \mathbb{X}^n \to \mathbb{X}$, i.e., $\forall x \in \mathbb{X}^n.\ f(x)[i] = f_i(x)$.

3.2. Abstract Gradient Descent

Like gradient descent, *abstract gradient descent* (AGD) can be regarded as a local-search-based optimization algorithm. This subsection introduces the basic idea of AGD, and we refer readers to Yu et al. (2024) for more detailed introduction.

Basically, AGD is trying to calculate the preimage (aka. inverse image) of a given function f with a given image:

Definition 1 *For a function $f : \mathbb{X}^n \to \mathbb{X}$ with input $\alpha \in \mathbb{X}^n$, the abstract gradient is a boolean function $\hat{\nabla} f : \mathbb{X}^n \to 2^{\mathbb{X}^n}$ such that*

$$\hat{\nabla} f(\alpha) = \{\Delta\alpha | f(\alpha - \Delta\alpha) < f(\alpha)\} \tag{7}$$

In other words, AGD tries to find all possible descending directions ($\Delta\alpha$) to the minimization of F. However, computing the exact abstract gradient is generally infeasible. Therefore, we use *lower* $\hat{\nabla}^{\blacktriangledown}$ (*upper* $\hat{\nabla}^{\blacktriangle}$) abstract gradients instead. In practice, the former is used for efficiency and the latter for guarantees. Luckily, computing lower or upper abstract gradients can be carried out efficiently based on rules in an abstract domain, with the help of backward abstraction interpretation Miné (2014). We listed an example of how backward abstraction interpretation works to calculate the abstract gradient in Appendix 6.1.

3.3. The Abstract Domain of Boxes

The abstract domain of Boxes (boxes) has been widely used in abstract interpretation for program analysis. The advantages of boxes lie in: i) most operators, such as union or intersection of sets, are closed in boxes; ii) Elements in boxes can be represented in matrix, thus operations over them can be done efficiently on GPU.

Definition 2 *A set $V \in 2^{\mathbb{X}^n}$ is a* box-set *if there exists $l_1, ..., l_n, u_1, ..., u_n \in \mathbb{X} \cup \{-\infty, +\infty\}$ such that $\forall x \in \mathbb{X}^n. \, x \in V \leftrightarrow \bigwedge_{i=1}^{n}(l_i \leq x[i] \leq u_i)$, where $-\infty$ and $+\infty$ stand for negative and positive infinity, respectively.*

*A set $V \in 2^{\mathbb{X}^n}$ is called a k-*boxes-set *if there exists k box-sets V_1, V_2,...,V_k such that $V = \bigcup_{j=1}^{k} V_j$. Conventionally, we denote a one dimension boxes-set as $[a, b] = \{a \leq x \leq b\}$ and two dimension boxes-set as $[a, b] \times [c, d] = \{a \leq x[0] \leq b \wedge c \leq x[1] \leq d\}$, and so on.*

4. Our Method

The symbolic learning framework we propose is an iterative step-by-step optimization process between learning and verification. Its main framework includes 3 phases: i) Construction of the symbolic neural network (SNN) (i.e., the F function in Equation (4)) based on the composition of primitive functions; ii) Generation of the min-max discrete optimization problem from the training set D and the constraint C; iii) The abstract gradient descent process for this min-max optimization.

4.1. Primitive Functions

We only consider the primitive functions, predicates, and logic connectives occurring in NIA (Nonlinear Integer Arithmetic) and NRA (Nonlinear Real Arithmetic), which can be well solved by SMT solvers Barrett et al. (2016). They are summarized in Table 1.

Similar to neural networks, we can construct a composed function F as a symbolic neural network in terms of the listed primitive functions. For example, an affine layer of the traditional neural network, $f(x) = w \cdot x + b$, where $w \in \mathbb{R}^n, x \in \mathbb{R}^n, b \in \mathbb{R}$, can be written as: $f(x) = \mathit{Add}(\mathit{Add}(...\mathit{Add}(\mathit{Mul}(w[0], x[0]), \mathit{Mul}(w[1], x[1]))...), b)$, where $x[i], w[i] \in \mathbb{R}$. SNN does not require the F to be differential anymore. Therefore, we can build more intuitive structures with predicates and logic connectives in the symbolic neural network with much fewer parameters to achieve the data fitting task (LeNet5 LeCun et al. (1989), which contains 7 layers with 59,654 learnable

Symbol	Discrete Function	Domain	Description
$\neg$	*Not*	$\mathbb{B} \to \mathbb{B}$	Logic negation.
$\wedge$	*And*	$\mathbb{B}^2 \to \mathbb{B}$	Logic conjunction.
$\vee$	*Or*	$\mathbb{B}^2 \to \mathbb{B}$	Logic disjunction.
$\cdot ? \cdot : \cdot$	*Ite*	$\mathbb{B} \times \mathbb{X}^2 \to \mathbb{X}$	If-Then-Else operation
$\leq$	*Le*	$\mathbb{X}^2 \to \mathbb{B}$	Linear order: 'less than or equal to'.
$=$	*Eq*	$\mathbb{X}^2 \to \mathbb{B}$	Linear order: 'equal to'.
$<$	*Lt*	$\mathbb{X}^2 \to \mathbb{B}$	Linear order: 'less than'.
$+$	*Add*	$\mathbb{X}^2 \to \mathbb{X}$	Addition.
$\times$	*Mul*	$\mathbb{X}^2 \to \mathbb{X}$	Multiplication.
$\div$	*Div*	$\mathbb{X} \times \mathbb{X}/0 \to \mathbb{X}$	Division.
$\%$	*Mod*	$\mathbb{I} \times \mathbb{I}^+ \to \mathbb{I}^+$	Modulo operation.
$[\cdot]$	*Rnd*	$\mathbb{X} \to \mathbb{I}$	Rounding.

Table 1: The involved operations in a mixed theory of NIA and NRA.

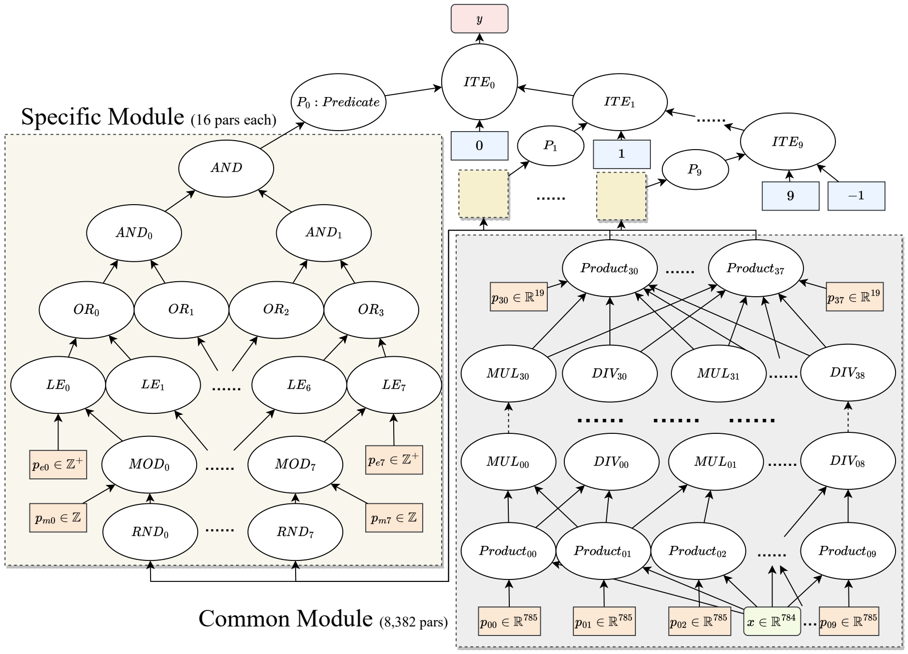


Figure 3: The Symbolic Neural Network for Hand-written Digit Recognition.

parameters, can achieve 99. 9% fitting accuracy on MNIST. For comparison, the inner construction of the symbolic neural network F for this task, depicted in Figure 3, contains only 8,542 parameters and achieves a fitting error of 99. 8%).

4.2. Architecture Design of SNN for MNIST

Figure 3 depicts the SNN model to fulfill the hand-written digit recognition task. The outermost layer of function is a series of 10 *Ite* functions, where the last *Ite* function can output a special label: -1, denoting that the input picture is predicted to be non-digit. The former 9 *Ite* functions are composed with a Boolean function (i.e. a predicate denoted as P_i), a constant function (0 to 9), and a succeeding *Ite* function.

Each P_i is constructed with two modules: a common module to extract features of the pixel matrix, and a specific module expected to run certain logic decisions with the extracted feature to predict the specific label. The common module is shared by the 10 *Ite* functions, while the 10 specific modules are monopoly to each *Ite* function.

The common module is formed of 4 layers, each of which consists a linear layer and an nonlinear layer. The linear layers are the same to traditional neural network: 784×10, 18×10, 18×10,

18×8 affine layers. The nonlinear layers multiplies and divides the adjacent pairs of the 10 output of linear layers, producing 18 outcomes.

The specific module first rounds the 8 real values output from the common module to integers, then apply *Mod* function with learnable parameters p_{mj} on these integers. The resulting value will be compared with other learnable parameters p_{ej}, resulting in 8 Boolean values. The last three layers are constructed as a Conjunctive Normal Form to produce the final Boolean value, deciding whether to predict this input as the corresponding label or not. The whole model contains 8,542 parameters, with an inference time of less than 100ms.

Besides the symbolic neural network, we can also specify constraints uniformly. For example, $\forall x.\ F(x,p) = F(-x,p)$ can be represented as $\forall x.\ \mathit{Eq}(F(x,p), F(\mathit{Mul}(x,-1),p))$.

4.3. Min-max Optimization

We use f_{soft} and f_{hard} to denote the soft constraint $\sum_{\langle x',y\rangle\in D} \mathit{loss}(F(x',p),y)$ and the hard constraint $\neg C(p,x)$ in Equation (4), respectively, i.e.,

$$f_{\mathit{soft}}(p) = \sum_{\langle x,y\rangle\in D} \mathit{Ite}(\mathit{Eq}(F(x,p),y),0,1) \quad \text{and} \quad f_{\mathit{hard}}(x,p) = C(x,p) \tag{8}$$

where $\sum$ stands for the composition of a sequence of *Add* functions; $C(x,p)$ is a quantifier-free first-order logic formula only with free variables p and x, the output of $F(x,p)$ and y are scalars.

Because of the non-convexity and discontinuity of C and F, we do not expect to find a ground-optimized p^*, which is impractical. The primary target is to find the value of p such that $C(x,p)$ is valid (i.e. $C(x,p) \geq 1$), and the second target is to minimize the fitting error. Thus, the above goal is formalized as the following min-max optimization:

$$\min_{p\notin S^-} f_{\mathit{soft}}(p) \qquad \max_{p\in S^+,x} \mathit{Not}(f_{\mathit{hard}}(x,p)) \tag{9}$$

Let $p_i \in \mathbb{X}^n$ be the values of parameters p after the $(i-1)$-th iteration (it can be any sampled value in $S^+_{(i-1)}$), then $S^-_{(i)}$ and $S^+_{(i)}$ are:

$$\begin{aligned} S^+_{(i)} &= \hat{\nabla}^{\blacktriangledown} f_{\mathit{soft}}(p_i) \quad \text{s.t.} \quad S^+_{(i)} \cap \bigcup_{j=1}^{i-1} S^-_{(j)} = \emptyset \\ S^-_{(i)} &= \hat{\nabla}^{\blacktriangledown} (f_{\mathit{hard}} = 1)[p:] \quad \text{s.t.} \quad S^-_{(i)} \cap S^+_{(i)} \neq \emptyset \end{aligned} \tag{10}$$

where $\hat{\nabla}^{\blacktriangledown}(f_{\mathit{hard}} = 1)$ denotes the lower approximated preimage of f_{hard} with the imgae $\{0\}$; $V[p:]$ is the projection of the set V on the dimensions of parameters p; and $\bigcup_{j=1}^{i} S^-_{(j)}$ represents the union of all S^- in previous iterations.

The min-max optimization works as follows:

1. Initialize $S^-_{(0)}$ as an empty set, and $i := 1$.

2. Compute $S^+_{(i)}$ according to (10).

   If $S^+_{(i)}$ is not empty, go to step 3.

   If $S^+_{(i)}$ is empty, set $S^+_{(i)} = (\bigcup_{j=1}^{i-1} S^-_{(j)})^c$ (that is to give up the soft optimization target for this time). If $S^+_{(i)}$ is not empty, then go to step 3. If $S^+_{(i)}$ is still empty, then it indicates that the hard constraint is unsatisfiable.

3. Compute $S^{-}_{(i)}$ according to (10).

   If $S^{-}_{(i)}$ is not empty, go to step 2.

   If $S^{-}_{(i)}$ is empty, go to step 4.

4. Compute $\hat{\nabla}^{\blacktriangle}(f_{hard} = 1)$ with $p_i \in S^{+}_{(i)}$(the upper approximated preimage of f_{hard} with the imgae $\{0\}$).

   If $\hat{\nabla}^{\blacktriangle}(f_{hard} = 1) \neq \emptyset$, then go to step 3.

   If $\hat{\nabla}^{\blacktriangle}(f_{hard} = 1) = \emptyset$, then the optimization terminates with $S^{-} = (S^{+}_{(i)})^{c}$, and repeat computing S^{+}_{i+1}, S^{+}_{i+2},..., until S^{+}_{k} is empty.

Upon termination, any $p \in S^{+}_{k}$ provides an optimal solution $p*$.

The above min-max optimization is sound but incomplete. If it terminates, we can get the p^{*} that both satisfies the hard constraint and fits the dataset well. However, it may not terminate due to precision loss of abstraction. In the experiments, if the number of iterations between the two optimizations is more than 50, we give up the hard constraint and only run optimization on the soft constraint. If the refinement iterations between step 3 and step 4 are more than 100, we also give up the hard constraint.

5. Evaluation

This section evaluates the effectiveness of our method on the task of hand-written digits recognition Dataset MNIST Deng (2012). We are interested in whether our method can produce a model that can not only recognize the hand-written digits, but also satisfy logical constraint such as norm robustness, zooming, translation, rotation, fading, frame annotation, contrast of pixel, thresholding, and symmetry. We compare our method with two SOTA methods, named Logic Constraint Learning Li et al. (2023) (*LogicCL*) and Robust Learning Wong and Kolter (2018) (*RobustL*), and a data augmented deep learning (*Data+L*) as the baseline.

A brief introduction of the dataset (soft constraint), together with the formal definition of the interested properties (hard constraint), and the experiment settings, can all be checked in Appendix 6.2[1].

We summarize the results in Table 2. As can be seen, 9 out of the 11 hard constraints are successfully satisfied by the learned SNN. The attack errors of SNN on the 9 hard constraints are all 0. The unsatisfied properties are both about rotation, in the optimization process of which we drop the hard constraints after 50 interactions in the min-max learning. Further, if SNN fails to satisfy a hard constraint, it will be as vulnerable as the data-augmented learning to the attacking. On the contrary, once a constraint is verified, SNN is immune to attacks from this property.

The cost of SNN to satisfy a hard constraint is intuitive. SNN cannot fit the soft constraint (the training data) as well as other learning methods, as it has the highest training error in most properties' training sets. Specifically, SNN has a training error as high as 10.4% in the property Symmetry's training set, one order of magnitude higher than the other two methods. Another drawback of SNN is that it takes extensive training time cost. The other three methods can finish 100 epochs of training within 5 minutes, while SNN has to take 5 to 12 hours to finish one epoch of

[1] The repository is at https://doi.org/10.5281/zenodo.15425225

Property (Verified?)	Training Error %				Test Error %				Attack Error %			
	Data+L	*LogicCL*	*RobustL*	*SNN*	*Data+L*	*LogicCL*	*RobustL*	*SNN*	*Data+L*	*LogicCL*	*RobustL*	*SNN*
Fading (✓)	**0.6**	0.8	-	1.5	3.1	2.6	-	**1.5**	12.6	10.4	-	**0**
Zooming (✓)	**1.1**	1.9	-	5.2	7.1	6.5	-	**5.0**	41.6	20.2	-	**0**
Translation (✓)	**2.5**	3.4	-	5.3	4.8	**4.1**	-	5.9	11.9	8.2	-	**0**
Rotation (✗)	**2.9**	3.3	-	3.5	8.1	6.2	-	**3.7**	46.2	**33.1**	-	37.4
Frame (✓)	**0.1**	0.4	-	0.5	1.7	0.9	-	**0.5**	7.7	2.1	-	**0**
Contrast Pixel (✓)	1.4	1.4	-	**1.1**	5.8	4.0	-	**1.1**	18.6	11.5	-	**0**
Threshold (✓)	**0.0**	0.2	-	0.4	0.9	0.8	-	**0.5**	2.2	1.8	-	**0**
Symmetry (✓)	**0.7**	1.8	-	10.4	**2.5**	2.7	-	10.5	10.4	8.3	-	**0**
180° Rotation (✗)	**0.2**	0.5	-	1.7	2.7	**1.2**	-	2.2	10.6	**2.3**	-	8.1
Robust(ϵ0.1)(✓)	**0.0**	0.3	0.1	2.1	**1.1**	1.4	1.8	2.2	8.1	4.5	3.9	**0**
Robust(ϵ1)(✓)	**0.4**	1.8	0.7	4.1	4.8	2.5	**2.1**	4.7	20.9	8.0	9.3	**0**

Table 2: The first column lists the 11 properties (hard constraints). The mark within the parentheses indicates if this hard constraint is verified successfully by SNN. The second column shows the *Training Error*, the rate of unfitted data on the training set. The third column shows the error rate on the original test set, named *Test Error*. Test Error indicates the usual performance of models. The last column shows the predict error of models on the attack set, generated by applying the constraint on correctly predicted instance in the test set. The last error reveals the anti-attack ability of models.

training. The inference process, on the other hand, is equally fast, as SNN is a tiny model compared to others.

Nevertheless, we notice that the training errors and the test error of SNN on all properties are very close, while the other methods' training error are always distinctively lower than the test error. This indicates that SNN does not over-fit the training set. The reason behind is that SNN has only 8,542 parameters, comparing to the 59,654 parameters in LeNet5. And the much longer and exquisite learning process also keeps SNN from over-fitting, although with the risk of under-fitting.

In summary, SNN learning can gain the guarantee of certain properties, which are ever lacking in current deep learning methods, with the cost of much longer learning time and the risk of under-fitting.

6. Conclusion

This paper presents a novel neurosymbolic framework, named Symbolic Neural Network, for guarantee needed learning tasks, in which the desired properties of a learned model are represented in logic constraint, which can be regarded as a discrete function.

While current works aim to approximate the logic constraint as a continuous function and add it into the loss function, problems such as approximation error and local optimum keep them from giving sound guarantees. In a different view, SNN adopts abstract gradient descent with the abstract domain of Boxes to directly optimize discrete functions. It regards the target of data fitting as a soft constraint and treats the logical constraint as a hard constraint. By carrying an min-max optimization scheme, SNN tries to train a learning model that not only satisfies the constraints but also fits the data as well as possible. The soundness of this method is given by theory of AGD, and the computational cost on abstraction inference can be accelerated by GPU power. Experiments on the MNIST task also demonstrate its advantages in generalization-ability, anti-attacking, and the ability to give assurance, which is previously lacking in current deep learning models.

Acknowledgments

This work is supported by the NSFC grant under Grant No. 62032024.

References

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021. doi: 10.3233/FAIA201017. URL https://doi.org/10.3233/FAIA201017.

Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Static analysis and verification of aerospace software by abstract interpretation. *Found. Trends Program. Lang.*, 2(2-3):71–190, 2015. doi: 10.1561/2500000002. URL https://doi.org/10.1561/2500000002.

Jörg Brauer, Thomas Noll, and Bastian Schlich. Interval analysis of microcontroller code using abstract interpretation of hardware and software. In *Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems*, pages 1–10, 2010.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res.*, 21: 42:1–42:39, 2020. URL http://jmlr.org/papers/v21/19-468.html.

Shaowei Cai, Bohan Li, and Xindi Zhang. Local search for satisfiability modulo integer arithmetic theories. *ACM Trans. Comput. Logic*, 24(4), jul 2023. ISSN 1529-3785. doi: 10.1145/3597495. URL https://doi.org/10.1145/3597495.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.49. URL https://doi.org/10.1109/SP.2017.49.

Liqian Chen, Jiangchao Liu, Antoine Miné, Deepak Kapur, and Ji Wang. An abstract domain to infer octagonal constraints with absolute value. In Markus Müller-Olm and Helmut Seidl, editors, *Static Analysis*, pages 101–117, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10936-7.

Patrick Cousot. Abstract interpretation. *ACM Computing Surveys (CSUR)*, 28(2):324–328, 1996.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

Andrew Cropper and Sebastijan Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022. doi: 10.1613/JAIR.1.13507. URL https://doi.org/10.1613/jair.1.13507.

Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477. URL https://doi.org/10.1109/MSP.2012.2211477.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18. IEEE Computer Society, 2018. doi: 10.1109/SP.2018.00058. URL https://doi.org/10.1109/SP.2018.00058.

Francesco Giannini, Giuseppe Marra, Michelangelo Diligenti, Marco Maggini, and Marco Gori. On the relation between loss functions and t-norms. In Dimitar Kazakov and Can Erten, editors, *Inductive Logic Programming - 29th International Conference, ILP 2019, Plovdiv, Bulgaria, September 3-5, 2019, Proceedings*, volume 11770 of *Lecture Notes in Computer Science*, pages 36–45. Springer, 2019. doi: 10.1007/978-3-030-49210-6_4. URL https://doi.org/10.1007/978-3-030-49210-6_4.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

Arie Gurfinkel and Sagar Chaki. Boxes: A symbolic abstract domain of boxes. In Radhia Cousot and Matthieu Martel, editors, *Static Analysis - 17th International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010. Proceedings*, volume 6337 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2010. doi: 10.1007/978-3-642-15769-1_18. URL https://doi.org/10.1007/978-3-642-15769-1_18.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2017. doi: 10.1007/978-3-319-63387-9_5. URL https://doi.org/10.1007/978-3-319-63387-9_5.

Hari Govind Vediramana Krishnan, Yuting Chen, Sharon Shoham, and Arie Gurfinkel. Global guidance for local generalization in model checking. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 101–125. Springer, 2020. doi: 10.1007/978-3-030-53291-8_7. URL https://doi.org/10.1007/978-3-030-53291-8_7.

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 396–404. Morgan Kaufmann, 1989. URL http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.

Zenan Li, Zehua Liu, Yuan Yao, Jingwei Xu, Taolue Chen, Xiaoxing Ma, and Jian Lü. Learning with logical constraints but without shortcut satisfaction. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=M2unceRvqhh.

Antoine Miné. The octagon abstract domain. *Higher-order and symbolic computation*, 19:31–100, 2006.

Antoine Miné. Backward under-approximations in numeric abstract domains to automatically infer sufficient program conditions. *Sci. Comput. Program.*, 93:154–182, 2014. doi: 10.1016/j.scico.2013.09.014. URL https://doi.org/10.1016/j.scico.2013.09.014.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.282. URL https://doi.org/10.1109/CVPR.2016.282.

Matan Peled, Bat-Chen Rothenberg, and Shachar Itzhaky. SMT sampling via model-guided approximation. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2023. doi: 10.1007/978-3-031-27481-7_6. URL https://doi.org/10.1007/978-3-031-27481-7_6.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/0d3496dd0cec77a999c98d35003203ca-Abstract-Conference.html.

Sriram Sankaranarayanan, Michael A Colón, Henny Sipma, and Zohar Manna. Efficient strongly relational polyhedral analysis. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 111–125. Springer, 2006.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artif. Intell.*, 302:103602, 2022. doi: 10.1016/J.ARTINT.2021.103602. URL https://doi.org/10.1016/j.artint.2021.103602.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29909–29921, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html.

Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th In-*

ternational Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 5283–5292. PMLR, 2018. URL http://proceedings.mlr.press/v80/wong18a.html.

Banghu Yin, Liqian Chen, Jiangchao Liu, and Ji Wang. Hierarchical analysis of loops with relaxed abstract transformers. *IEEE Transactions on Reliability*, 69(1):203–215, 2020. doi: 10.1109/TR.2019.2925037.

Shiwen Yu, Zengyu Liu, Ting Wang, and Ji Wang. Neural solving uninterpreted predicates with abstract gradient descent. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024. ISSN 1049-331X. doi: 10.1145/3675394. URL https://doi.org/10.1145/3675394.

Yaoguang Zhai and Sicun Gao. Monte carlo tree descent for black-box optimization. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/5185aa776fd64ae3b4c6dae1af1066b1-Abstract-Conference.html.

Appendix

6.1. An example of Abstract Gradient Calculation on boxes

In this section, we take the rule of *Add* as an example to illustrate the inference process of primitive functions. And the abstractions of a compositional function on boxes can be constructed from the abstractions of its component primitive functions.

Definition 3 *An abstraction rule $\mathcal{R}_f$ for a function $y = f(x)$ is in a form of $\frac{C_s(x)}{E_s(y)}$, where C_s and E_s are logic predicates with s as the coefficients and x and y as the variables. C_s is called* condition *and E_s is called* conclusion *respectively. If C_s and E_s are in the form of a box-set, we say $\mathcal{R}_f$ is a rule on* boxes *domain.*

The abstraction rules for all primitive functions in Table 1 are proposed by Yu et al. (2024).

Rule *[*Add*]*

$$\frac{(s_l \leq x_1 \leq s_u) \wedge (l - s_l \leq x_2 \leq u - s_u)}{l \leq \text{Add}(x_1, x_2) \leq u} \tag{11}$$

The variables other than x, such as s_l, s_u, l, and u, are *coefficients*. It is easy to see this rule is sound: $((s_l \leq x_1 \leq s_u) \wedge (l - s_l \leq x_2 \leq u - s_u)) \Rightarrow (l \leq \mathit{Add}(x_1, x_2) \leq u)$. This rule is also complete: once there exist a pair of (x_1, x_2) such that $l \leq x_1 + x_2 \leq u$, the condition $(s_l \leq x_1 \leq s_u) \wedge (l - s_l \leq x_2 \leq u - s_u)$ is satisfiable. Further, this rule is tight under the condition $(-\infty < l \leq u < \infty)$ since no other box-set abstraction is more precise than (super set of lower/ subset of upper) the constructed ones.

Now, consider we have an image set $Y = [3, 6]$, and we need calculate its lower approximated preimage (denoted as $\overleftarrow{Add}^{\blacktriangledown}(Y)$). The solving query is:

$$\exists x \in \mathbb{R}^2. \qquad [3, 6] = [l, u] \wedge ((s_l \leq x[0] \leq s_u) \wedge (l - s_l \leq x[1] \leq u - s_u))$$

It is easy to see that the above query is a set of linear inequalities, which can be solved by linear programming. One valid answer is $l = 3, u = 6, s_l = 2, s_u = 4$, and hence $\overleftarrow{Add}^{\blacktriangledown}(Y) = [2, 4] \times [1, 2]$.

6.1.1. ABSTRACTION PROPAGATION

The above example shows how to produce backward or forward abstractions for one primitive function. For an arbitrarily composed function $f : \mathbb{X}^n \rightarrow \mathbb{X}^m$, we first decompose it into w layers of functions: $f = g^{(w)} \circ g^{(w-1)} \circ ... \circ g^{(1)}$, where $g^{(j)} : \mathbb{X}^{d_{j-1}} \rightarrow \mathbb{X}^{d_j}$ and $d_0 = n$, $d_w = m$. Each layer function $g^{(j)}$'s each projection function $g_i^{(j)}$ is a primitive function in Table 1. We denote the abstraction inference rule for a primitive function as $\mathcal{R}_{g_i^{(j)}} : \frac{C_{g_i^{(j)}}}{E_{g_i^{(j)}}}$, with the condition as $C_{g_i^{(j)}}$ and conclusion as $E_{g_i^{(j)}}$.

It is easy to see that we can construct a rule for the layer function g as: $\mathcal{R}_{g^{(j)}} : \frac{\bigwedge_{i=1}^{d_j} C_{g_i^{(j)}}}{\bigwedge_{i=1}^{d_j} E_{g_i^{(j)}}}$. Coefficients in different rules are independent.

Three things should be noted:

1. If each $C_{g_i^{(j)}}$ and $E_{g_i^{(j)}}$ are in the form of box-set, then the constructed $\mathcal{R}_{g^{(j)}}$'s condition and conclusion are also in the form of box-set.

2. If the solving query of coefficients of each $C_{g_i^{(j)}}$ and $E_{g_i^{(j)}}$ during inference is a disjunctive of linear inequalities, then the solving query of coefficients of $\mathcal{R}_{g^{(j)}}$ during inference is also a disjunctive of linear inequalities.

3. If each $\mathcal{R}_{g_i^{(j)}}$ is sound and complete, then $\mathcal{R}_{g^{(j)}}$ constructed above is also sound and complete.

The propagation process is similar to back-propagation of gradient: calculate the outmost layer function and then propagate to the inner layers. We refer readers to Yu et al. (2024) for more detailed introduction.

6.2. Detail of Experiments

6.2.1. TASK AND DATASET

The MNIST dataset contains 60,000 instances of hand-written digits for training and 10,000 instances for testing. In the experiment, we will build a soft constraint learning target function only on the 60,000 instances in the training set. The rest 10,000 instances will be unseen to the learned model, and are used to test the model's performance.

Each instance can be regarded as an input-output pairs: $\langle x, y \rangle$, where $x \in \mathbb{R}^{784}$ and $y \in \{0, 1, ..., 9\}$. This task is challenging to traditional formal methods in two aspects: First, the variable space is a very high-dimensional space, where most solving techniques such as SMT methods

(DPLL-T Barrett et al. (2021) and CDCL-T Krishnan et al. (2020)), are exponential to the number of dimensions. In addition, although the high dimensional space seems not a problem for gradient-descent-based deep learning methods, it will bring great trouble for verification of certain properties for a learned deep learning model, where the computing cost is also exponential (e.g., Branch-and-Bound methods Bunel et al. (2020)). Second, the digits recognition task needs complex nonlinear operations to tackle, which not only bring complexity but also further increase the computing cost.

6.2.2. CONSTRAINTS

Although the existing deep learning models for the MNIST dataset have already reached a predication accuracy as 99.9%, there are still many bottlenecks keeping the current methods from safe-critical applications. For example, one can still easily find min-max counterexample to misleading the prediction of deep learning models that have a perfect accuracy on the test set Goodfellow et al. (2015); Carlini and Wagner (2017); Moosavi-Dezfooli et al. (2016).

We present SNN to provide the guarantee of properties, including but not limited to robustness, zooming, translation, rotation, fading, frame annotation, contrast of pixel, thresholding, and symmetry. We illustrate the properties in Table 3.

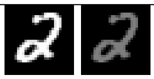
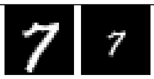
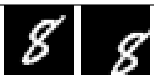
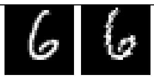
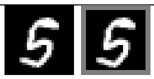
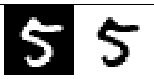
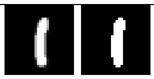
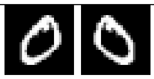
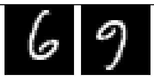
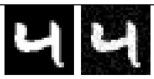
Name	Hard Constraint	Example
Fading	$\forall x.\ F(x,p) = F(x/2,p)$	
Zooming	$\forall x.\ F(x,p) = F(Zoom(x,0.5),p)$	
Translation	$\forall x.\ F(x,p) = F(Shift(x,5,5),p)$	
Rotation	$\forall x.\ F(x,p) = F(Rotate(x,15^\circ),p)$	
Frame Annotation	$\forall x.\ F(x,p) = F(Frame(x),p)$	
Contrast of Pixel	$\forall x.\ F(x,p) = F(255-x,p)$	
Thresholding	$\forall x.\ F(x,p) = F(Threshold(x),p)$	
Symmetry	$\forall x.\ F(x,p) \in \{0,1,8\}$ $\leftrightarrow (F(x,p) = F(Flip(x),p))$	
180° Rotation	$\forall x.\ F(x,p) = 6$ $\leftrightarrow F(Rotate(x,180^\circ),p) = 9$	
Norm Robust	$\forall x.\ \|\|x - x_0\|\|_\infty < 30 \rightarrow F(x) = F(x_0)$	

Table 3: Properties as Hard Constraints of the Symbolic NN model. The implementation of the functions $Zoom$, $Shift$, $Rotate$, $Frame$, $Threshold$, and $Flip$ can be checked in Equation (12). $||\cdot||_\infty$ is the infinite norm.

$$
\begin{aligned}
&Zoom(x,0.5)[i][j] = (\sum_{t=0}^{t=1}\sum_{k=o}^{k=1} x[i*2+t][j*2+k])/4 \\
&Shift(x,a,b)[i][j] = \begin{cases} x[i-a][j-b] & (i-a \geq 0 \wedge j-b \geq 0) \\ 0 & \text{else} \end{cases} \\
&Rotate(x,\theta)[i][j] = x[[\cos(-\theta)*i+\sin(-\theta)*j],[\cos(-\theta)*j-\sin(-\theta)*i]] \\
&Frame(x)[i][j] = \begin{cases} x[i][j] & (4 \leq i \leq 22 \wedge 4 \leq j \leq 22) \\ 100 & \text{else} \end{cases} \\
&Threshold(x)[i][j] = \begin{cases} 255 & x[i][j] \geq 128 \\ 0 & \text{else} \end{cases} \\
&Flip(x)[i][j] = x[27-i][j]
\end{aligned}
\tag{12}
$$

6.2.3. EXPERIMENT SETTINGS

To evaluate the performance of our method, we designed a series of experiments, assessing the accuracy, the training time cost, the property satisfiability for single or combinations of constraints, and the ability of anti-attacking. Our method is compared with the SOTA works Wong and Kolter (2018); Li et al. (2023), named Logic Constraint Learning Li et al. (2023) (*LogicCL*) and Robust LearningWong and Kolter (2018) (*RobustL*), and a data augmented deep learning (*Data+L*) as the baseline. *RobustL* is only applicable to the Norm Robust property, and the other two methods are applicable to all properties. They all use LeNet5 LeCun et al. (1989) as the learning model, containing 7 layers with 59,654 learnable parameters including: 2 convolution layers, 2 max-pooling layers, 2 fully connected layers, and 1 soft-max output layer. As a comparison, the inner construction of the symbolic neural network F for this task, depicted in Figure 3, contains only 8,542 parameters.

To compare with the data augmented learning, we apply the hard constraint as a data augmentation method to the training set. Each instance produces one augmented instance if the hard constraint is applicable. The performance of model will be tested on two test sets. The first is the original 10,000 instances in the test sets. The second is an *attack* set produced by using the hard constraint as an augmentation on the correctly predicted instances in the original test set.

The rules used in both forward and backward can all be vectorized and implemented on GPU (we implement the vector computation by the ArrayFire C++ library[2]). Hence, we can support batch. The batch size on constructing the soft constraint is set as 4096. The upper abstract gradient $\overleftarrow{p}^{\blacktriangle}$ is propagated through batches, while the lower abstract gradient $\overleftarrow{p}^{\blacktriangledown}$ is dropped between batches. We only train SNN on one epoch as more epochs brings no improvement to the abstract gradient descent. We set an epoch of 100 and a batch size of 512 for the traditional gradient-descent methods.

Also, in the SNN learning, if we failed to pass the min-max optimization on x for the hard constraint for 50 interactions, we will drop the hard constraint and run soft constraint optimization only to get a trained model.

The computing environment of the experiments is: CPU i9-13900KF, GPU RTX-4090, and Memory 256G. All experiments are carried 5 times, and we average the results.

[2] https://arrayfire.com/