DeepLogic: Joint Learning of Neural Perception and Logical Reasoning

Xuguang Duan,Xin Wang, *Member, IEEE,* Peilin Zhao,Guangyao Shen,Wenwu Zhu, *Fellow, IEEE*

Abstract—Neural-symbolic learning, aiming to combine the perceiving power of neural perception and the reasoning power of symbolic logic together, has drawn increasing research attention. However, existing works simply cascade the two components together and optimize them isolatedly, failing to utilize the mutual enhancing information between them. To address this problem, we propose **DeepLogic**, a framework with joint learning of neural perception and logical reasoning, such that these two components are jointly optimized through mutual supervision signals. In particular, the proposed **DeepLogic** framework contains a deep-logic module that is capable of representing complex first-order-logic formulas in a tree structure with basic logic operators. We then theoretically quantify the mutual supervision signals and propose the deep&logic optimization algorithm for joint optimization. We further prove the convergence of **DeepLogic** and conduct extensive experiments on model performance, convergence, and generalization, as well as its extension to the continuous domain. The experimental results show that through jointly learning both perceptual ability and logic formulas in a weakly supervised manner, our proposed **DeepLogic** framework can significantly outperform DNN-based baselines by a great margin and beat other strong baselines without out-of-box tools.

Index Terms—Connectionism and neural nets; Perceptual reasoning

1 Introduction

NEURAL—symbolic learning, which targets combining the perception ability of deep neural networks (DNNs) and the reasoning ability of symbolic reasoning systems, has attracted increasing attention in the research community. On the one hand, although current DNNs have achieved promising results in various domains such as computer vision [1] and natural language processing [2], they can hardly handle logical reasoning tasks directly [3], [4]. On the other hand, though traditional symbolic logical reasoning approaches have concrete theories and rich applications in dealing with discrete logic [5], [6], they are not designed to deal with semantic data such as raw images and text.

Therefore, some preliminary works have been trying to combine neural perception with symbolic logical reasoning [7], [8] . Existing works cascade symbolic systems to deep neural models [9], constructing DNN structures with logic constrains [10] or developing differentiable logic learning methods [11], [12], [13]. However, they optimize the two parts isolatedly, failing to utilize the correlation between the two cascaded parts, thus would be hard to reach the global optimal.

In this work, we explore the joint learning of neural perception and logical reasoning, where the neural perception component provides guidance to learn logic rules while the logic formulas obtained from the logical reasoning component, in turn, supervise the neural perception learning, *i.e.*, these two components can mutually enhance each other. Nevertheless, achieving this purpose is challenging given the essential differences between continuous feature space used by neural perception models and discrete symbolic space adopted by logical reasoning algorithms.

To tackle the challenge, we propose a joint learning framework **DeepLogic** for neural-symbolic reasoning, which contains a deep-logic module (DLM) and a deep&logic optimization (DLO) algorithm. In particular, DLM is a learnable formula tree derived from first-order logic (FOL) and can automatically represent logic rules with FOL formulas. DLO is a joint-optimization algorithm that can mutually enhance neural perception and logical reasoning through iteratively quantifying the mutual supervision signals between the two parts. With our proposed DeepLogic framework, we can jointly learn both perception ability and the logic formula with a 1-bit supervision signal indicating whether the semantic inputs satisfy the given formula or not, as shown in Fig. 1. We conduct extensive experiments to demonstrate several advantages of our proposed DeepLogic Framework. To evaluate our methods, we conduct extensive experiments include:

1) Evaluating model convergence and performance with a *number addition* task based on the MNIST dataset;
2) Testing the generalization ability of DeepLogic with multiple rules and multiple attributes;
3) Relaxing DLM with differentiable logic operators [14] on the well-known RAVEN dataset [15].

To summarize, this paper makes the following contributions:

- We propose the **DeepLogic** framework with a theoretical convergence guarantee, which conducts the joint learning of neural perception and logical reasoning such that they can mutually enhance each other to boost the performance and explainability of neural-symbolic reasoning.

- *Xuguang Duan, Xin Wang, Guangyao Shen and Wenwu Zhu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. Peilin Zhao is with Teccent AI Lab.*
- *Xin Wang and Wenwu Zhu are corresponding authors.*
  E-mail: duan_xg@outlook.com, {xin_wang, wwzhu}@tsinghua.edu.cn, thusgy2012@gmail.com, masonzhao@tencent.com

- We propose the deep-logic module (DLM) which derives from first-order logic, capable of constructing and learning logic formulas from basic logic operators.
- We propose the deep&logic optimization (DLO) algorithm to guarantee the joint learning of neural perception and logical reasoning through theoretically quantifying the mutual supervision signals between them.

The remainder of this paper is organized as follows. We review related works in Sec. 2 and present an overview of our proposed DeepLogic framework in Sec. 3. We describe the deep-logic module (DLM) and deep&logic optimization (DLO) algorithm in detail in Sec. 4 and Sec. 5, respectively, followed by our extensive experiment in Sec. 6 to validate the superiority of our proposed approach in both model performance and explainability. Finally, we conclude the whole paper in Sec. 7.

2 Related Works

With the development of deep learning, the researchers find that pure DNN methods perform quite poor when it comes to those tasks requiring complex reasoning and generalizing ability [4], [7], [16], [17], and thus the communities try to seek help from traditional symbolic methods, which, however, could not handle semantic data [18], [19], [20].

On the one hand, numerous efforts have been done to integrate reasoning into deep neural networks. One line of work is to design new structures to enable DNN with reasoning ability [10], [16], [21], [22], [23]. Besides, in neural modular networks [24], the fully blackbox neural networks are reorganized in a rational way with logical supervision. The modularization endows each module with a specific function and the whole structure can thus be parsed into human-readable symbolic structures (e.g., programs [25] or parsing tree [24], [26]). Though these works can achieve a certain level of reasoning and generalization, they are still blackbox models in essence and suffer from the limitation of requiring extra annotations.

On the other hand, treating deep neural networks as a powerful component of the symbolic system also draws attention from the community [11]. Along with this work, designing a differentiable version of the current symbolic system attracts lots of research interests. In particular, [11] extends Prolog [27] by employing DNN as a new Prolog predicate to deal with those non-symbolic inputs. [13] proposes a differentiable Forth interpreter and [12] proposes the ∂ILP system that learns first-order-logic clauses with a differentiable SAT solving strategy.

Further, letting the neural network unleash its strength to be a powerful perception model and employing the complex reasoning part to handle the symbolic system has gradually become a new consensus [7], [8], [28]. One straightforward way is to *cascade the two systems in a way that the neural network detects objects from semantic inputs and the symbolic executor reason over the symbolic representations of objects to find the final answer* [7], [8]. Other works [9], [29] follow this line of research and solve the Visual Question Answering (VQA) tasks on CLEVR dataset [30], where a pre-trained MASK-RCNN [31] is used as the perception model to detect objects, and a predefined program is used to reason over the detected objects to seek the final answer. Such types of methods either rely heavily on the accuracy of the pre-trained neural network or require extra annotations for the intermediate symbol representation. [7] and [28] ease the extra annotation costs through their proposed solutions. [7] uses an external Prolog [27] program to reason over the "pseudo-labels" generated by the neural network, and fixes those "inconsistency" labels with a zero-order optimization [32] method. [28] further introduces a grammar model to supervise the training of the perception model and gives a formal view of what the "supervision signal" could be. However, [7] and [28] rely on a predefined out-of-box Prolog program or a grammar model, which makes them difficult to generalize to other tasks.

In this paper, we take one step further, requiring that the symbolic system should also be learned rather than predefined. This strategy will be much more challenging while more attractive when it comes to new scenarios. We propose to jointly learn the neural network and the logic system by iteratively utilizing their mutual correlation.

3 The DeepLogic Framework

Neural-symbolic learning targets the problem of simultaneous perception and reasoning, where the inputs are usually semantic data and the desired output is a complex relationship of inputs previously unknown to the algorithm (e.g., some logical relationship between a certain attribute). We note that the symbolic attributes of the semantic inputs to be learned should not be given. Otherwise, this task would be fully disentangled into two separate parts, which would degenerate into two simple sub-tasks. In this section, we will mathematically describe the problem formulation and modeling of our proposed DeepLogic framework, followed by a brief introduction of our proposed Deep&Logic Optimization (DLO) algorithm for joint learning the neural perception and symbolic reasoning. In the following, a bold letter (*e.g.*, $\mathbf{z}$) represents a vector, and a normal letter (*e.g.*, z) represents a scalar.

3.1 Formulation

Formally, given a semantic input $\mathbf{x}$ (*e.g.*, an image) and a label $y \in \{0, 1\}$ indicating whether the symbolic attributes $\mathbf{z}$ of $\mathbf{x}$ meet a certain latent logical concept $\Delta^\star$ or not. The goal of neural-symbolic learning is to simultaneously learn $p(y|\mathbf{x})$ as well as *explicitly* discover the correct symbolic concept $\Delta^\star$ and the attribute mapping functions from $\mathbf{x}$ to $\mathbf{z}$. In particular, we decouple $p(y|\mathbf{x})$ by introducing symbolic attributes $\mathbf{z}$ as follows:

$$p(y|\mathbf{x}) = \sum_{\mathbf{z}\in\mathcal{Z}} \underbrace{p_\theta(\mathbf{z}|\mathbf{x})}_{neural} \cdot \underbrace{p_\phi(y|\{z_i\})}_{symbolic}, \quad (1)$$

where the attribute mapping $p_\theta(\mathbf{z}|\mathbf{x})$ is a neural network while the logical module $p_\phi(y|\mathbf{z})$ is modeled with a symbolic system Δ_ϕ. We rewrite $\mathbf{z}$ as $\{z_i\}$ here to emphasize the role of each symbol z_i.

For example, to solve VQA tasks, $\mathbf{x}$ could be the input image and question, $\mathbf{z}$ is a hidden program, and y is the final answer [9], [29].

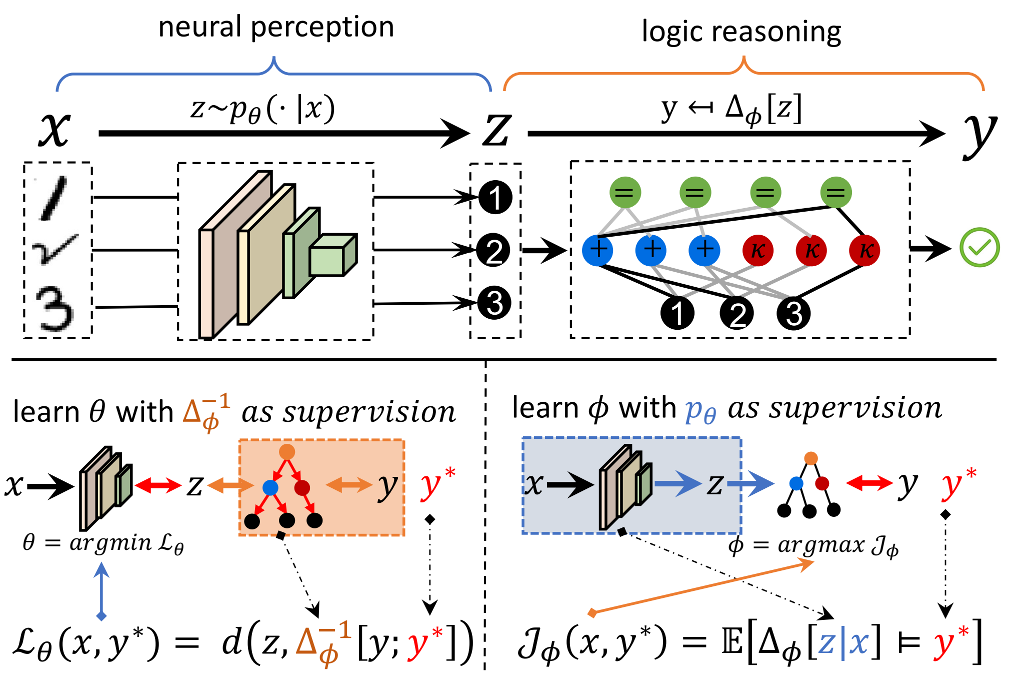
neural perception
logic reasoning
$z \sim p_\theta(\cdot|x)$
$y \leftarrow \Delta_\phi[z]$
x
z
y
learn θ with Δ_ϕ^{-1} as supervision
learn ϕ with p_θ as supervision
$\theta = argmin\, \mathcal{L}_\theta$
$\phi = argmax\, \mathcal{J}_\phi$
$\mathcal{L}_\theta(x, y^*) = d(z, \Delta_\phi^{-1}[y; y^*])$
$\mathcal{J}_\phi(x, y^*) = \mathbb{E}[\Delta_\phi[z|x] \vDash y^*]$

Fig. 1: The proposed DeepLogic framework. The forward pass (top) is processed sequentially from semantic input $\mathbf{x}$ through intermediate symbolic attribute $\mathbf{z}$ to the final deductive label y. For example, to reason on the relationship between , and . First, the system recognize these images into symbols as:❶, ❷ and ❸ with the neural perception model. Then, the logic reasoning model reasons the relationship between ❶, ❷ and ❸, and concludes that they meet the logic formula: "❶ `add` ❷ `equals` ❸". In the backward pass (bottom left / bottom right), the parameters of perception model θ and symbolic system ϕ are iteratively optimized with the other one as supervision, respectively.

3.2 Modeling the Neural Perception (p_θ)

The best choice for p_θ is a set of neural network $nn_\theta(\cdot)$ to predicate hidden symbolic attributes $\mathbf{z}$ from $\mathbf{x}$:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \prod_i \sigma\left(nn_{\theta,i}(\mathbf{x})\right)_{z_i} = \prod_i \frac{exp\left(nn_{\theta,i}(\mathbf{x})_{z_i}\right)}{\sum_{z' \in Z_i} exp\left(nn_{\theta,i}(\mathbf{x})_{z'}\right)}, \tag{2}$$

where $\sigma(nn_{\theta,i}(\mathbf{x}))$ maps $\mathbf{x}$ into a distribution over the symbol set Z_i for the i-th attribute, where $nn_{\theta,i}(\mathbf{x})$ is a DNN that has $|Z_i|$ outputs while σ is the softmax function that normalizes the DNN output. $nn_{\theta,i}(\mathbf{x})_{z_i}$ and $\sigma(nn_{\theta,i}(\mathbf{x}))_{z_i}$ are the output value of raw DNN output and the normalized probability.

3.3 Modeling the Logical Reasoning (p_ϕ)

p_ϕ is designed to conduct logical reasoning with explicit logic formulas. Denote Δ_ϕ as such a symbolic system, $p_\phi(y|\mathbf{z})$ is given as:

$$p_\phi(y|\mathbf{z}) = \begin{cases} 1, & \Delta_\phi[\{z_i\}] \vDash y \\ 0, & otherwise \end{cases}. \tag{3}$$

$\Delta_\phi[\{z_i\}] \vDash y$ means $\Delta_\phi[\{z_i\}]$ is consistent with y. If y is a boolean constant, then $\Delta_\phi[\{z_i\}] \vDash y$ is equivalent to $\Delta_\phi[\{z_i\}] = y$.

Dai et al. [7] model Δ_ϕ as a predefined Prolog [27] program, while Li et al. [28] regard Δ_ϕ as the python `eval` function to conduct `arithmetic computation`. In this work, our proposed DeepLogic framework differs from them through learning a "symbolic system" from scratch instead of utilizing a predefined symbolic system. We design a novel learnable **deep-logic** module to achieve this purpose, which is able to conduct logical reasoning with the learned explicit logic formulas.

3.4 Joint Learning of Neural Perception and Logical Reasoning

To complete the joint learning of neural perception and logical reasoning, we further propose a joint-optimization algorithm, i.e., Deep&Logic Optimization (DLO), to jointly learn the perception model p_θ and symbolic system p_ϕ. As shown in Fig. 1, the optimization of neural perception and logical reasoning are conducted iteratively. When optimizing the neural network p_θ, the symbolic system Δ_ϕ together with the true label y^* act as the supervision to provided pseudo label z^*. When optimizing the symbolic system Δ_ϕ, the neural network works as a sampler to sample z from prior x to judge the performance of Δ_ϕ. More details of this optimization procedure can be found in Sec. 5.

4 The Deep-Logic Module (DLM)

In this section, we discuss our proposed deep-logic module (DLM) which is capable of modeling neural perception and logical reasoning. In particular, the proposed DLM benefits in the following advantages:

- DLM depends on no external knowledge and is easy to implement;
- DLM can adaptively fit for various scenarios through stacking the logic layers from shallow to deep;
- DLM is able to utilize supervised information to optimize both p_θ and p_ϕ, ensuring the joint learning of neural perception and logical reasoning.

This section is organized as follows: in Sec 4.1, we provide preliminary first order logic (FOL) concepts used in this work, followed by our introduction in Sec 4.2 on the proposed **deep-logic** module capable of implementing all the components described in Sec 4.1 and preserving differentiability.

4.1 Preliminaries on FOL

4.1.1 Terms and Formulas

First order logic (FOL), also known as predicate logic [1], is a formal language that enables us to quantify and reason on relationships between entities. For example, "father's father is grandfather" could be formulated using relationships `Father(X,Y)`, `GrandFather(X,Y)` as:"`GrandFather(X,Y)` $\leftarrow$ $\exists$ `Z Father(X,Z)` $\wedge$ `Father(Z,Y)`". Moreover, if we replace the variable `X,Y` with `John` and `Nash`, then the formula could be used to determine whether the two "entities" `John` and `Nash` meet the relation `GrandFather(·,·)` or not.

Formally, those entities are called `Terms`, which could be predefined or derived from other terms. Those relationships are represented with `Formulas`, which could be a relationship between entities or the logical combination of several relationships. A FOL model is defined as $(\mathcal{T}, \mathcal{F})$ where $\mathcal{T}$ and $\mathcal{F}$ are the set of all possible terms and formulas, respectively. To make this paper self-contained, we formally define several concepts mathematically as follows:

Definition 1 (Terms $\mathcal{T}$)**.** Terms are defined recursively:

1) Constants and Variables are (atomic) terms, where a constant z is a symbol representing a concrete object and Z is a variable representing an indeterminate object.
2) Given r terms $t_1, t_2, ...t_r$ and an r-artriy operation $f : \mathcal{T}^r \mapsto \mathcal{T}$, then $f(t_1, t_2, ...t_r)$ is also a term.

Definition 2 (Formulas $\mathcal{F}$)**.** Formulas are also defined recursively:

1) Given an r-artiy predicate $p : \mathcal{T}^r \mapsto \{$`True, False`$\}$, and r terms $t_1, t_2, ...t_r$, then $p(t_1, t_2, t_3, ...t_r)$ is an (atomic) formula.
2) Given formula P, Q, and connective Ξ $(\{\wedge, \vee, \neg\})$, $P \Xi Q$ is also a formula.

Definition 3 (Logic Grounding)**.** The value of a formula could be `True`, `False` or `indeterminate`. By replacing the variable within a formula with a constant, we are able to eliminate the state of `indeterminate` and obtain a determined value to indicate whether this constant fits the formula or not. We denote this process as $\Delta[\{z_1/Z_1, z_2/Z_2, ..\}]$, which indicates "grounding variable Z_i to a constant z_i", where Z and z are the set of variables and constants, respectively. Sometimes Z can be omitted, and therefore the grounding process is denoted as $\Delta[\{z_1, z_2, \cdots\}]$.

1. See [33, pp.3/pp.15] for more details.

Definition 4 (Degree of Term and Formula)**.** The degree, i.e., $\aleph$, is used to model the complexity of a term or formula, where an atomic term has degree 0, and the degree of other terms and formulas are defined as the maximum degree of their children plus one:

$$\aleph\left(f(s_1, s_2, ...s_r)\right) = \underset{i \in [1,r]}{max}\, \aleph(s_i) + 1. \tag{4}$$

4.1.2 Recursive Definition of A FOL Model

Given the definition of terms, formulas, and other related concepts shown above, we could formally define a FOL model recursively.

Specifically, the atomic terms are (Def. 1):

$$\mathcal{T}_0 = \{z_i\} \cup \{Z_i\}, \tag{5}$$

where z_i, Z_i are constant and variable respectively, then we have $\mathcal{T} = \bigcup_{k<\infty} \mathcal{T}_k$, and

$$\mathcal{T}_k = \{f(t_1, t_2, ...t_{r_f}) | f \in F, t_i \in \mathcal{T}_{<k}\}. \tag{6}$$

Here F is the set of operations, r_f is the arity of operation f, and $\mathcal{T}_{<k} \leq \bigcup_{j<k} \mathcal{T}_j$.

Similarly, formulas are:

$$\mathcal{F}_0 = \{p(t_1, t_2, ...t_{r_f}) | p \in P, t_i \in \mathcal{T}\}, \tag{7}$$

$$\mathcal{F}_k = \{\Xi(f_i, f_j) | \Xi \in \{\wedge, \vee, \neg\}, f_i, f_j \in \mathcal{F}_{k-1}\}, \tag{8}$$

where P is the set of predicates, $\{\wedge, \vee, \neg\}$ is the set of connectives, and $\mathcal{F}_0$ is the set of atomic formulas.

We further unify the definition of terms and formulas, given that they have the same format of recursion. For each step of the recursion, we denote the input (term/formula) set as $S^{(i)}$, and the operation/relationship/connectives set as Ψ_s, then the output set can be formulated as:

$$\mathcal{S}^{(o)} = \{\psi(t_1, t_2, ...t_{r_f}) | \psi \in \Psi_s, t_i \in \mathcal{S}^{(i)}\}, \tag{9}$$

Example. *We give a full example of "arithmetic" FOL model to illustrate the aforementioned recursive definition. Let* $\mathcal{T}_0 = \{0, 1, X, Y\}$ *where* 0 *and* 1 *are constants representing* ⓿ *(zero) and* ❶ *(one),* X *and* Y *are variables;* $F = \{+, \times\}$ *where "+" and "×" are the 2-arity operators* `add` *and* `multiple`*, respectively;* $P = \{=\}$ *where "=" is the 2-arity predicate* `equal` *and define "*$= (1, +(0, 1)) \mapsto True$*". Based on Eq. (5)-(8):*

$$\begin{aligned}
\mathcal{T}_0 =& \{0, 1, X, Y\}, \\
\mathcal{T}_1 =& \{+(0,0), +(0,1), +(1,1), +(X,Y), +(X,1), \\
& \times(0,0), \times(0,1), \times(1,1), \times(X,Y), \times(X,1) \cdots\} \\
\mathcal{T}_2 =& \{+\left(0, +(0,1)\right), +\left(+(0,1), +(X,1)\right) \cdots\}, \\
& \cdots \\
\mathcal{F}_0 =& \{= (0,1), = (1, +(0,X)), \cdots\} \\
\mathcal{F}_1 =& \{(= (0,1)) \wedge (= (1, +(0,X))), \cdots\} \\
& \cdots
\end{aligned}$$

With the above recursive definition, we could find that $\mathcal{T} = \bigcup_{k<\infty} \mathcal{T}_k$ *would represent the whole set of integers* ($\mathbb{N}$)*, where "*$+(1,1)$*" represents* ❷ *(two), "*$+(1, +(1,1))$*" represents* ❸ *(three), respectively. Besides, the using of variables would enable the representation of an indeterminate entity, e.g., '*$+(X, 1)$*" represents "X plus one" for some integer* X*.*

With the definition of "=", we could find that the formula "$= (0,1)$*" is* `False`*, while "*$= (+(1, +(0,1)), +(1,1))$*" is*

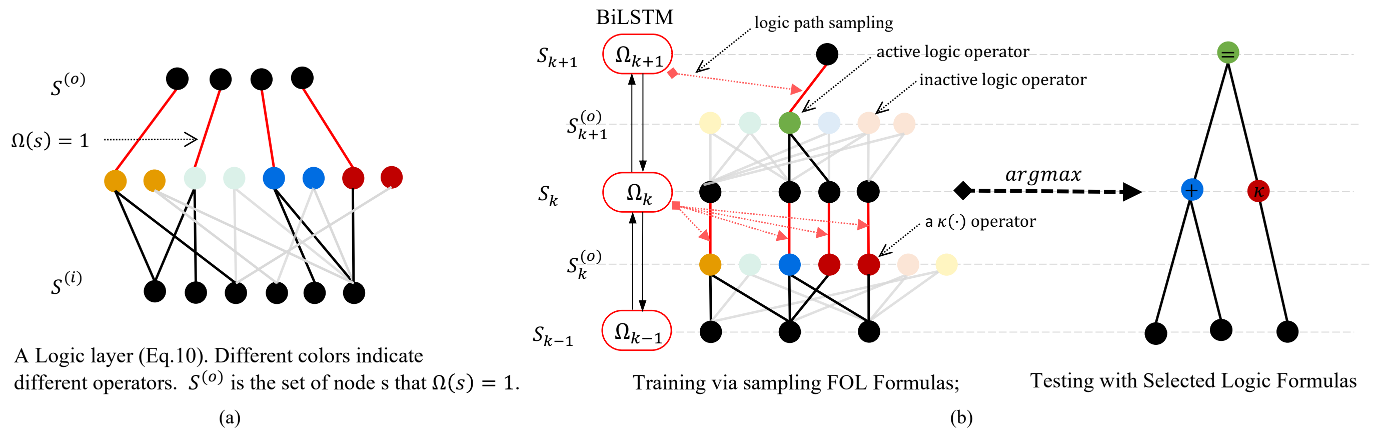
$S^{(o)}$
$\Omega(s)=1$
$S^{(i)}$
A Logic layer (Eq.10). Different colors indicate different operators. $S^{(o)}$ is the set of node s that $\Omega(s)=1$.
(a)
BiLSTM
logic path sampling
active logic operator
inactive logic operator
S_{k+1}
Ω_{k+1}
$S_{k+1}^{(o)}$
S_k
Ω_k
$S_k^{(o)}$
S_{k-1}
Ω_{k-1}
argmax
a $\kappa(\cdot)$ operator
=
+
κ
Training via sampling FOL Formulas;
Testing with Selected Logic Formulas
(b)

Fig. 2: (a) A single Logic layer as defined in Eq. 10; (b) The illustration of deep-logic module (DLM) as is presented in Sec. 4.2. After the learning process, those edges with the biggest weights are preserved, and we obtain the formula *"$Eq(Add(Z_1, Z_2), Z_3)$"*.

True if we apply the grounded formula "$=(1,+(0,1))$" to replace "$+(0,1)$" with "1". And the correctness of formula "$=(1,+(0,X))$" depends on the value of X.
This recursive definition could finally define arithmetic over the entire natural number $\mathbb{N}$. Moreover, if we introduce abbreviated notations such as "$2 \triangleq +(1,1), 3 \triangleq +(1,+(1,1)), \cdots$".

Remark 4.1.1 (redundancy of the recursive definition)**.** *The definition in Eq. (6), (8) would result in some redundant definitions with respect to terms for the reason that different layers would define the same term (Fig 3(c)). In this paper, we remove the redundancy by constrain that the every symbol in $\mathcal{S}_k^{(o)}$ should have degree k. Formally: $\mathcal{S}_k = \{\psi(s_1, s_2, ...s_{r_f}) | \psi \in \Psi, s_i \in \mathcal{S}_{<k}, \aleph(f(s_1, s_2, ...s_{r_f})) = k\}$. The condition $\aleph(f(s_1, s_2, ...s_{r_f})) = k$ indicates that at least one input term should have degree $k-1$, i.e., come from $\mathcal{S}_{k-1}$. This constraint could efficiently remove the duplication.*

Remark 4.1.2 (complexity of the FOL model)**.** *The definitions of terms and formulas depend on infinitely recursions, which is intractable in real-world applications. Therefore, in this paper, we configure the max-recursion steps of terms and formulas in the same manner as other works [10], [12]. Based on the complexity of tasks, the max-recursion can be adaptively changed to fit for different complexities.*

4.2 The Definition of the Deep-logic Module

With the definition in Eq. (5)-(9), it's possible to define a FOL model with a set of operations and predicates. However, it would be very resource-consuming if we directly apply this definition to enumerate all the possible items in the FOL model (As in [12]). In this paper, we solve this problem with the basic hypothesis that *not all symbols are useful in the FOL model, especially when our goal is to discover a specific formula hidden in the data.*

In the following, we introduce the **deep-logic** module which realizes all the features discussed in Sec. 4.1, while keeping the model complexity trackable simultaneously.

4.2.1 A logic Layer

From the definitions of terms and formulas (Eq. (5)-(8)), we are able to find that both of them can be organized as a triplet $(\mathcal{S}^{(i)}, \Omega, \mathcal{S}^{(o)})$ (Eq. (9)). Moreover, in Remark 4.1.1, we show that these constraints on the degree of symbol can remove redundancy. In this section, we introduce two extra techniques to address Remark 4.1.2 gracefully:

- Firstly, we introduce a new operator `keep` $\kappa(s_1, s_2, ..) \triangleq s_1$, and let $\bar{P} = P \cup \{\kappa(\cdot)\}$, $\bar{F} = F \cup \{\kappa(\cdot)\}$, where $\kappa(\cdot)$ is the auxiliary operator (predicate or connective) that copies the symbols from the previous layer. $\kappa(\cdot)$ disentangles the direct dependency between disjoint layers. With this auxiliary operator, $\mathcal{S}_k^{(i)}$ could be $\mathcal{S}_{k-1}^{(o)}$ rather than $\bigcup_{j<k} \mathcal{S}_j^{(o)}$, *i.e.*, the output set of the previous layer is exactly the input set of the current layer, which is quite useful to construct the **deep-logic** module layer by layer.
- Secondly, with the hypothesis that *not all the symbols are useful*, we do not need to compute the full output set. In this work, we set a hyperparameter L_k and select L_k symbols to conduct the computations, generating the new output as a subset of the full output symbols: $\bar{\mathcal{S}}_k^{(o)} \subset \mathcal{S}_k^{(o)}$ where $|\bar{\mathcal{S}}_k^{(o)}| = L_k$. During training, we sample a subset of symbols as the output and optimize the sampling distribution. During testing, the best sampling path is picked out, which then acts as a determinate formula.

Through these two techniques, the k-th logic layer can be modified as $\left(\mathcal{S}_k^{(i)}, \bar{\Phi}_k, \Omega_k\right)$ where Ω_k is the sampler determining which output symbols should be computed. $\bar{\mathcal{S}}_k^{(o)}$ can be represented as:

$$\bar{\mathcal{S}}_k^{(o)} = \{s \in \mathcal{S}^{(o)} : \Omega_k(s) = 1\}, \tag{10}$$

where $\Omega_k(\cdot) \in \{0,1\}$ and $\sum_{\omega \in \Omega_k} \omega = L_k$.

An illustration of the logic layer could be found in Fig. 2(a) where different colors represent different operators and the black lines indicate the symbols selected to construct $S^{(o)}$, and the gray lines refer to those unselected symbols.

4.2.2 The Implementation of the Deep-logic Module

Sec. 4.2.1 illustrates the structure of a single logic layer, which is designed to construct the **deep-logic** module through being stacked multiple times. As remark 4.1.2 states, we conduct finite recursion for terms and formulas based on the task complexity wiht M term layers, and $N+1$ formula layers given that $\mathcal{F}_0$ based on $\mathcal{T}$ is always included.

When learning the structure parameter $\Lambda = \{\Omega_k\}_{k=1}^{M+N+1}$, it is impossible to learn the parameters of different layers separately because they are coupled together. Therefore, we model these coupled structure parameters through a Recurrent Neural Network, which is a common practice for the Neural Architecture Search [34]. Formally, we define a sampler p_ϕ, to sample Λ as follows:

$$\begin{aligned} &\mathbf{h}_k = \texttt{BiLSTM}_\phi\left(\{\mathbf{h}_j\}_{j<k},\ \{\mathbf{h}_j\}_{j>k}\right), d \\ &\Omega_k \sim Multinomial(\cdot\,; \mathbf{h}_k), \end{aligned} \tag{11}$$

where $\texttt{BiLSTM}_\phi$ is a bidirectional LSTM cell that models dependencies between different layers. The model after sampling is denoted as Δ_ϕ to when concerning the sampling parameter ϕ, and denoted as Δ_Λ when concerning the sampled structure Λ.

Training. Suppose we have a training dataset $\mathcal{D}$, and our goal is to optimize ϕ that Δ_ϕ could best model the logic underlying $\mathcal{D}$. To achieve this, we sample structure variable Λ from p_ϕ as in Eq. (11), and optimize the expectation performance of Δ_Λ as follows:

$$\phi^\star = \arg\max_\phi \underset{\Lambda\sim p_\phi}{\mathbf{E}}\left[\sum_{(z,y)\in\mathcal{D}} \mathbf{1}\left(\Delta_\Lambda[\mathbf{z}] \models y\right)\right], \tag{12}$$

where $\Delta_\Lambda[\mathbf{z}] \models y$ means that $\Delta_\Lambda[\mathbf{z}]$ is consistent with label y (logic entailment).

Testing. Given ϕ^*, the best Λ can be selected as $\Lambda^* = \arg\max_{\Lambda\ \sim q_{\phi^*}} \sum_{(\mathbf{z},y)\in\mathcal{D}} \mathbf{1}\left(\Delta_\Lambda[\mathbf{z}] \models y\right)$, and Δ_{Λ^*} is then used as a fixed FOL formula.

As shown in Fig. 2 (b), in the training phase, those active nodes are selected based on Ω_k to form a logic tree. During testing, the optimal path is selected to form a new logic tree that best describes the underlying logic.

Compared with existing works, this parameterized model in our proposed deep-logic module benefits in serving as a learnable logic system that can learn formulas from data rather than heavily depend on hand-crafted formulas. Moreover, our learning process is less time-consuming than fully enumerating methods [12] and independent of extra tools such as Prolog [11], Inductive Logic Programming [12], and Forth language [13], *etc.*

5 The Deep&logic Optimization (DLO)

We have introduced our proposed deep-logic module (DLM), which is a general FOL-based formula learner capable of learning symbolic relationships between symbols. In this section, we will introduce how DLM together with deep neural networks (DNNs) is able to handle the neural-symbolic tasks through absorbing semantic inputs and reasoning over their symbolic relationships. We also present details on the proposed deep&logic optimization (DLO) algorithm to optimize DLM and DNNs jointly.

5.1 Optimization

Our goal is to maximize the log-likelihood of the true label y^* given $\mathbf{x}$ with Eq. (1):

$$\begin{aligned} \mathcal{J}(\mathbf{x}, y^*) &= \log p(y^*|\mathbf{x}) \\ &= \log \sum_{\mathbf{z}\in\mathcal{Z}} \Big(p_\theta(\mathbf{z}|\mathbf{x}) \cdot p_\phi(y|\{z_i\})\Big). \end{aligned} \tag{13}$$

The optimal θ and ϕ are obtained through the following equation:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{(\mathbf{x},y^*)\in\mathcal{D}} \mathcal{J}(\mathbf{x}, y^*), \tag{14}$$

where $\mathcal{D}$ is the task dataset. However, directly optimizing θ and ϕ are intractable due to the non-differentiability in essence.

5.1.1 *Optimize p_θ with p_ϕ fixed*

Applying the likelihood ratio trick to Eq. 13, the gradient of θ becomes:

$$\begin{aligned} \nabla_\theta \mathcal{J}(\mathbf{x}, y^*) &= \nabla_\theta \log p(y^*|\mathbf{x}) \\ &= \frac{1}{p(y^*|\mathbf{x})} \nabla_\theta \left(\sum_{\mathbf{z}\in Z^N} p_\theta(\mathbf{z}|\mathbf{x}) \cdot p_\phi(y^*|\mathbf{z})\right) \\ &= \sum_{\mathbf{z}} \frac{p_\theta(\mathbf{z}|\mathbf{x}) p_\phi(y^*|\mathbf{z})}{\sum_{\mathbf{z}'} p_\theta(\mathbf{z}'|x) p_\phi(y^*|\mathbf{z}')} \nabla_\theta \log p_\theta(\mathbf{z}|\mathbf{x}) \\ &= \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x},y^*)}\left[\nabla_\theta \log p_\theta(\mathbf{z}|\mathbf{x})\right]. \end{aligned} \tag{15}$$

Moreover, if we assume that $\{z_i\}$ could be disentangled easily (which is usually the case as different attributes are irrelevant), then Eq. (15) could be further written as:

$$\begin{aligned} \nabla_\theta \mathcal{J}(\mathbf{x}, y^*) &= \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x},y^*)}\left[\nabla_\theta \log p_{\theta,i}(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x},y^*)}\left[\nabla_\theta \log \prod_i p_{\theta,i}(z_i|\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p(\cdot|\mathbf{x},y^*)}\left[\sum_i \nabla_{\theta,i} \log p_{\theta,i}(z_i|\mathbf{x})\right]. \end{aligned} \tag{16}$$

This means the gradient can be attribute-wisely decomposed, given that label y^*, the samples $\{z_i\}$, as well as their gradients can be separately computed. Now, the task turns into efficiently sampling $\mathbf{z}$ from the posterior $p(\mathbf{z}|\mathbf{x}, y^*)$. By considering Eq. (1) and Eq. (3), $p(\mathbf{z}|x, y^*)$ can be rewritten as:

$$\begin{aligned} p(\mathbf{z}|\mathbf{x}, y^*) &= \frac{p_\theta(\mathbf{z}|\mathbf{x}) p_\phi(y|\mathbf{z})}{\sum_{\mathbf{z}'} p_\theta(\mathbf{z}'|\mathbf{x}) p_\phi(y|\mathbf{z}')} \\ &= \begin{cases} \frac{p_\theta(\mathbf{z}|\mathbf{x})}{\sum_{\mathbf{z}'} p_\theta(\mathbf{z}'|\mathbf{x}) \cdot \mathbf{1}(\Delta_\phi[\mathbf{z}'] \models y^*)}, & \Delta_\phi[\mathbf{z}] \models y \\ 0, & otherwise \end{cases} \end{aligned} \tag{17}$$

where $\mathbf{1}(\cdot)$ is the indicator function. Eq. (16) and (17) show how logical reasoning supervises neural perception: **i.e., the logic module filters out $\mathbf{z}$ unsatisfying the "logic entailment" requirement.**

Qing *et al.* [28] propose an efficiently Monte Carlo sampling method to estimate the gradient by sampling

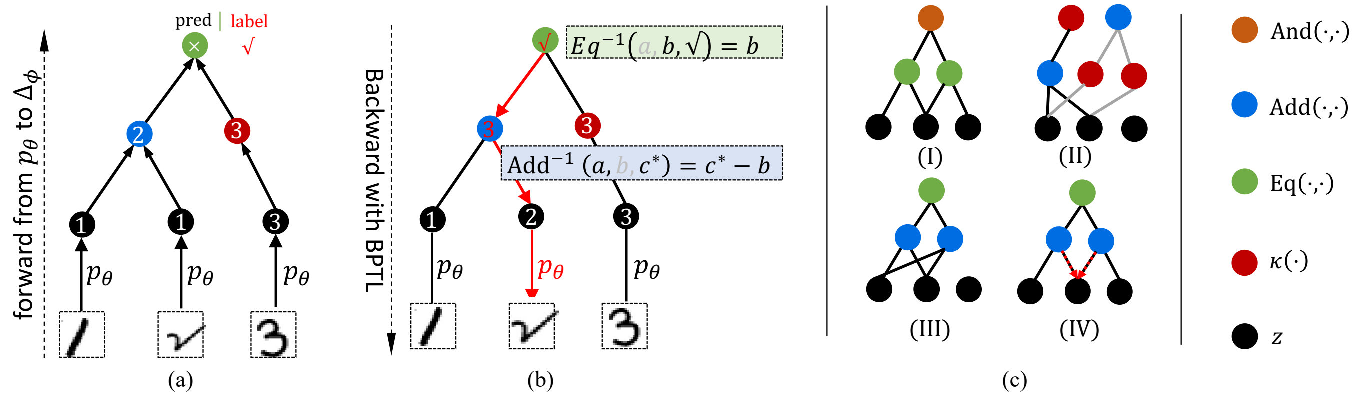
pred
label
forward from p_θ to Δ_ϕ
p_θ
(a)
Backward with BPTL
$Eq^{-1}(a, b, \surd) = b$
$\text{Add}^{-1}(a, b, c^*) = c^* - b$
p_θ
(b)
(I)
(II)
(III)
(IV)
(c)
$\text{And}(\cdot,\cdot)$
$\text{Add}(\cdot,\cdot)$
$\text{Eq}(\cdot,\cdot)$
$\kappa(\cdot)$
z

Fig. 3: (a) The forward pass of DeepLogic from p_θ to Δ_ϕ. (b) The backward pass of DeepLogic with the BPTL algorithm (Alg. 1). (c) Illustration of several cases of `deeplogic formulas`: (I) the formula for "$And(Eq(Z_1, Z_2), Eq(Z_2, Z_3))$"; (II) A case that two layers define the same term "$Add(Z_1, Z_2)$" (black line and gray line); (III) and (IV) ill /self-conflict case of formulas. In (III) the equation is always "`True`", and in (IV) the BPTL algorithm will encounter self-confliction in the middle node.

$\mathbf{z}$ from the posterior distribution, where the logic module $p(y|\mathbf{z})$ is a pre-defined tree-structure Grammar module (for arithmetic computation). Our proposed DeepLogic framework differs from the previous work in enabling $p_\phi(y|\mathbf{z})$ to be learned from data instead of pre-defined. However, the learned formula may not always satisfy the true rules because it is possible to include ill or even self-conflict formulas. For example, case(III) in Fig. 3(c) means that $Eq(Add(Z_1, Z_2), Add(Z_1, Z_2))$ is always `True`, thus resulting in this formula being ill and making no sense. As for case(IV)in Fig. 3(c), the formula $Eq(Add(Z_1, Z_2), Add(Z_2, Z_3))$ with $Z = [❶, ❷, ❸]$, and y =`True` would obtain different correction from ❶ and ❸ that try to correct ❷, which would always result in conflict supervision for symbol ❷ during the backward propagation. We thus propose the *Back Propagate Through Logic* (BPTL) algorithm taking these self-confliction into account to tackle the challenge, as shown in Alg. 1.

The basic idea of Alg. 1 is to: 1) back-propagate label $y*$ through the deep-logic module by recursively rectifying $\psi^{-1}(\{s_i^{(i)}\}; s^{(o)})$ for the input $\{s_i^{(i)}\}$ of operator ψ [2] such that the output is $s^{(o)}$; 2) at the same time, track symbol changes to eliminate those conflicting recursions. We continue this two steps until we could correct $\mathbf{z}$ to $\mathbf{z}^*$ such that $\Delta_\phi[\mathbf{z}^\star] \models y^\star$.

Moreover, we require the obtained $\mathbf{z}^*$ to follow the distribution defined in Eq. 17. By denoting the above process of finding valid $\mathbf{z}^*$ from $\mathbf{z}$ as $Q(\mathbf{z}^*|\mathbf{z})$ (See also the `BPTL` function in Alg. 1), we add an acceptance probability as:

$$\alpha(\mathbf{z}, \mathbf{z}^*) = \frac{\pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*)}{\pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*) + \pi(\mathbf{z})Q(\mathbf{z}^*|\mathbf{z})}. \tag{18}$$

It is easy to see that the transmission probability from any two states $\mathbf{z}$ and $\mathbf{z}^*$ are the same:

$$\begin{aligned}\pi(\mathbf{z})K(\mathbf{z}, \mathbf{z}^*) &= \pi(\mathbf{z})Q(\mathbf{z}^*|\mathbf{z})\frac{\pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*)}{\pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*) + \pi(\mathbf{z})Q(\mathbf{z}^*|\mathbf{z})}\\ &= \pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*)\frac{\pi(\mathbf{z})Q(\mathbf{z}^*|\mathbf{z})}{\pi(\mathbf{z}^*)Q(\mathbf{z}|\mathbf{z}^*) + \pi(\mathbf{z})Q(\mathbf{z}^*|\mathbf{z})}\\ &= \pi(\mathbf{z}^*)K(\mathbf{z}^*, \mathbf{z}),\end{aligned}$$

where $K(\mathbf{z}, \mathbf{z}^*) = Q(\mathbf{z}^*|\mathbf{z})\alpha(\mathbf{z}, \mathbf{z}^*)$ is the transmission probability from $\mathbf{z}$ to $\mathbf{z}^*$. By the detailed balanced condition [35], the whole process would efficiently sample z^* from $\pi(\mathbf{z}^*) = p(\mathbf{z}^*|\mathbf{x}, y^*)$.

In real implementation, as that $Q(\mathbf{z}|\mathbf{z}^*)$ could be 0 if $\Delta_\phi[\mathbf{z}] \not\models y^*$, we compute the value of $\alpha(\mathbf{z}, \mathbf{z}^*)$ by adding a small δ to $Q(\mathbf{z}'|\mathbf{z})$, then we could obtain the following three conditions:

1) $\pi(\mathbf{z}) = 0$ & $\pi(\mathbf{z}^*) > 0 \Rightarrow \alpha(\mathbf{z}, \mathbf{z}^*) = 1$ (always accept).
2) $\pi(\mathbf{z}) = 0$ & $\pi(\mathbf{z}^*) = 0 \Rightarrow \alpha(\mathbf{z}, \mathbf{z}^*) = 0$ (always reject).
3) $\pi(\mathbf{z}) > 0$ & $\pi(\mathbf{z}^*) > 0 \Rightarrow \alpha(\mathbf{z}, \mathbf{z}^*) \in (0, 1)$ (conditional).

This sampling process, from another view, generates pseudo labels for $nn_\theta(x)$. Every time $p_\theta(\cdot|x)$ gives a prediction z, Alg. 1 finds one of its neighbors that fit the logic constrains by inversely traveling through the logic tree Δ_ϕ to find $z^* = \Delta_\phi^{-1}[y; y^*]$ as a pseudo label for optimization.

5.1.2 Optimize p_ϕ with p_θ fixed

Because $p_\phi(\cdot|\{z_i\}) = \Delta_\phi[\{z_i\}]$ is non-differentiable with respect to ϕ , ϕ cannot be simply optimized with gradient ascend as θ does. Instead, we try to find ϕ^* such that $\mathcal{J}(\mathbf{x}, y^*)$ is maximized:

$$\begin{aligned}\phi^* &= \arg\max_\phi \mathcal{J}(\mathbf{x}, y^*) = \arg\max_\phi p(y^*|\mathbf{x})\\ &= \arg\max_\phi \sum_{\mathbf{z}\in Z^N} p_\theta(\mathbf{z}|\mathbf{x}) \cdot p_\phi(y^*|\mathbf{z})\\ &= \arg\max_\phi \mathop{\mathbb{E}}_{\mathbf{z}\sim p_\theta(\cdot|\mathbf{x})} \left[p_\phi(y^*|\mathbf{z})\right].\\ &= \arg\max_\phi \mathop{\mathbb{E}}_{\mathbf{z}\sim p_\theta(\cdot|\mathbf{x})} \left[\mathop{\mathbb{E}}_{\Lambda\sim p_\phi(\cdot)} \left[\mathbf{1}(\Delta_\Lambda[\mathbf{z}] \models y^*\right]\right],\end{aligned} \tag{19}$$

2. Note that some operator ψ would have indeterminate inverse operation, *e.g.*, for the inverse operation $a^* \leftarrow Or^{-1}(a$,`True`;`True`), the value of a^* could be either `True` or `False`. When such situation happens, we randomly select one.

```
Algorithm 1: Backpropagate Through Logic (BPTL)
Input: $p_\theta(\cdot|\mathbf{x})$; $\Delta_\phi$; $y^\star$        // prior; formula; target
Output: $\mathbf{z}^\star$        // $\mathbf{z}^\star \sim p(|\mathbf{x}, y^\star)$, see Eq.17
Function BPC($s$):
    /* $s$ and its children remain unchanged */
    if $s$ is leaf node then
        return $\{s \leftarrow s\}$
    else
        Retrive $\psi_s$, child $\{\dot{s}_1, \dot{s}_2, ...\dot{s}_{r_s}\}$        // Eq.9
        $\blacktriangle\mathbf{z} \leftarrow \varnothing$
        for $\dot{s} \in \{\dot{s}_i\}$ do
            $\blacktriangle\mathbf{z} \leftarrow \blacktriangle\mathbf{z} \cup$ BPC ($\dot{s}$)
        return $\blacktriangle\mathbf{z}$
Function BPTL($s$, $\hat{s}$):
    /* $\mathbf{s}$ is supposed to be $\hat{\mathbf{s}}$, find $\Delta\mathbf{z}$ */
    if $s$ is leaf node then
        return $\{s \leftarrow \hat{s}\}$        // reach leaf node;
    else
        Retrive $\psi_s$, child $\{\dot{s}_1, \dot{s}_2, ...\dot{s}_{r_s}\}$        // Eq.9
        for $\dot{s} \sim \{\dot{s}_i\}$ do
            /* note:s could be the same with $\hat{s}$ */
            $\triangle\mathbf{z}$ = BPTL ($\dot{s}$, $\phi_s^{-1}(\dot{s}_1, ...\dot{s}, ..., \dot{s}_{r_s}; \hat{s})$)
            $\blacktriangle\mathbf{z}$ = BPC ($\dot{s}_1, ...\dot{s}, ..., \dot{s}_{r_s}; \hat{s}$)
            if $\triangle\mathbf{z} \bowtie \blacktriangle\mathbf{z}$ then
                return $\triangle\mathbf{z} \cup \blacktriangle\mathbf{z}$ // $\triangle\mathbf{z}/\blacktriangle\mathbf{z}$ no conflict
            else
                return RAND($\varnothing$)        // $\triangle\mathbf{z}/\blacktriangle\mathbf{z}$ conflict
repeat
    $\mathbf{z}' \sim p(\cdot|\mathbf{x})$    // randomly sample a start point
    repeat
        $\Delta\mathbf{z}$ =BPTL ($\Delta_\phi[\mathbf{z}']$, $y^*$)
        $\mathbf{z}' = \Delta\mathbf{z}(\mathbf{z})$        // apply $\Delta\mathbf{z}'$ to $\mathbf{z}$
    until rand(0, 1)> $\alpha(\mathbf{z}, \mathbf{z}')$ OR Reach max-try
    if $\mathbf{z}$ is valid then
        return $\mathbf{z}'$        // find a valid $\mathbf{z}^\star$
until Reach max-try
return $\varnothing$    // $\Delta_\phi$ could be self-conflict(Fig.3)
```

where $p(y^*|\mathbf{x})$ is the distribution determined by Eq. (13). After changing the order of the two expectations, Eq. (19) could be rewritten as:

$$\phi^\star = \arg\max_{\phi} \mathop{\mathbb{E}}_{\Lambda \sim p_\phi(\cdot)} \left[R_\theta(\Lambda)\right], \tag{20}$$

where

$$R_\theta(\Lambda) = \mathop{\mathbb{E}}_{\mathbf{z} \sim p_\theta(\cdot|\mathbf{x})} \left[\mathbf{1}(\Delta_\Lambda[\mathbf{z}] \models y^*\right]. \tag{21}$$

.

Using the log gradient trick [36], the gradient to ϕ could be:

$$\nabla_\phi \mathcal{J}'(x, y^*) = \mathop{\mathbb{E}}_{\Lambda \sim p_\phi(\cdot)} \left[R_\theta(\Lambda) \cdot \nabla_\phi \log q_\phi(\Lambda)\right], \tag{22}$$

where $\mathcal{J}'(\mathbf{x}, y^*) = p(y^*|\mathbf{x})$ has the same optimal value with $\mathcal{J}(\mathbf{x}, y^*)$. This can also be regarded as the REINFORCE [37] algorithm with a reward function $R_\theta(\Lambda)$, which means: **the perception model acts as a reward function, assigning higher rewards to formulas Λs more consistent with data $(\mathbf{x}, y^*)$.**

```
Algorithm 2: The Deep&Logic Optimization
Input: $\mathcal{D} = \{(\mathbf{x}, y^*)\}$        // $y^*$ is the 1-bit label.
Output: $p_\theta(\cdot|\mathbf{x})$,$\Delta_\phi[\cdot]$        // perception/logic model
$p_\theta \leftarrow$ MIN_PRETRAIN($\mathcal{D}'$)        // See Sec.5.2
repeat
    /* Update $\phi$ while keeping $\theta$ fixed */
    for $(\mathbf{x}, y^* \in \mathcal{D})$ do
        $\Lambda \sim p_\phi(\cdot)$, $z \sim p(\cdot|\mathbf{x})$        // sample data
        Compute $R_\theta(\Lambda)$        // see Eq.21
        $\phi \leftarrow \phi + \nabla_\phi \mathcal{J}'(x, y^*)$        // see Eq.22
    /* Update $\theta$ while keeping $\phi$ fixed */
    for $(\mathbf{x}, y^* \in \mathcal{D})$ do
        $z \sim p_\theta(\cdot|x)$, $z^* \sim Q(z^*|z)$        // See Alg.1
        $\theta \leftarrow \theta + \nabla_\theta \log p_\theta(z^*|x)$        // see Eq.16
until performance unchanged
```

5.2 Pretraining p_θ and Convergence Analysis

Based on Eq. (16) and (22), p_θ and p_ϕ can be learned by alternatively optimizing θ and ϕ. However, alternative optimization from random initialization makes both p_θ and p_ϕ suffer from noisy gradients, and hard to converge. In literature, [7] uses a predefined Prolog [27] program, while [28] uses a python calculator as the logic model p_ϕ. In this paper, as our goal is to learn both p_θ and p_ϕ jointly, we face bigger challenge than [7], [28] do.

To solve the above problem, we propose that p_θ could be pretrained before the alternating optimization with a very small pretraining cost. This pretraining has a very small cost but could stabilize the alternating optimization process and ensure convergence. 6.1.2. The whole learning algorithm is presented in Alg. 2. Moreover, we show why a minimal pretraining on p_θ would ensure the convergence.

Given the target Λ^*, another random structure [3] Λ^- and an optimal perception model p_{θ^*}, we have:

$$\begin{aligned} R_{\theta^*}(\Lambda^*) &= \mathop{\mathbb{E}}_{\mathbf{z} \sim p_{\theta^*}(\cdot|\mathbf{x})} \left[\mathbf{1}(\Delta_{\Lambda^*}[\mathbf{z}] \models y^*\right] \\ &= \mathbf{1}\left(\Delta_{\Lambda^*}[\mathbf{z}] \models y^*\right) = 1 > R_{\theta^*}(\Lambda^-). \end{aligned} \tag{23}$$

With an untrained perception model p_{θ^-}, there would be no evidence which formula would perform better, *i.e.*:

$$R_{\theta^-}(\Lambda^*) \approxeq R_{\theta^-}(\Lambda^-). \tag{24}$$

Based on the above two inequalities, for a pre-trained p_θ, *s.t.* $p_\theta = \epsilon p_{\theta^*} + (1-\epsilon) p_{\theta^-}$ with some $\epsilon > 0$, we have:

$$\begin{aligned} R_\theta(\Lambda^*) &= \mathop{\mathbb{E}}_{\mathbf{z} \sim p_\theta(\cdot|vx)} \left[\mathbf{1}(\Delta_{\Lambda^*}[\mathbf{z}] \models y^*\right] \\ &= \epsilon R_{\theta*}(\Lambda^*) + (1-\epsilon) * R_{\theta^-}(\Lambda^*) \\ &> \epsilon R_{\theta*}(\Lambda^-) + (1-\epsilon) * R_{\theta^-}(\Lambda^-) \\ &= \mathop{\mathbb{E}}_{\mathbf{z} \sim p_\theta(\cdot|\mathbf{x})} \left[\mathbf{1}(\Delta_{\Lambda^-}[\mathbf{z}] \models y^*\right] = R_\theta(\Lambda^-). \end{aligned} \tag{25}$$

This also implies that even a weak pre-trained model is able to distinguish the target optimal formula from random formulas.

3. Please note if two formulas both have reward 1, then they are logical equivalent under the distribution of $p(x, y^*)$.

TABLE 1: Accuracy on the MNIST-ADD dataset, where EXTRA SUP indicates whether the model is trained with extra perception supervision or the only one-bit logic supervision, EXTRA TOOL indicats whether the model uses any extra tools or not. PERCEPTION is the accuracy of $x \xrightarrow{p_\theta} z$ indicating the performance of p_θ while LOGIC is the accuracy of $x \xrightarrow{p_\theta} z \xrightarrow{p_\phi} y$ indicating the performance of the cascaded p_θ and p_ϕ. Baselines without extra perception supervision could not report a PERCEPTION accuracy (mark with '/'). Besides, DeepLogic$^-$ does not further train p_θ. Both DeepLogic and DeepLogic$^-$ are pretrained for 6 batches.

MODEL	EXTRA SUP	EXTRA TOOL	MNIST-ADD-α				MNIST-ADD-β			
			PERCEPTION		LOGIC		PERCEPTION		LOGIC	
			TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST
MLP	√	×	99.30%	98.88%	98.12%	75.20%	99.42%	98.71%	98.17%	24.01%
LSTM	√	×	99.41%	98.52%	99.25%	78.91%	99.03%	98.98%	99.90%	41.04%
RN	√	×	99.10%	99.00%	92.41%	83.10%	99.44%	98.45%	92.33%	49.62%
MLP	×	×	/	/	98.44%	66.42%	/	/	98.18%	22.32%
LSTM	×	×	/	/	99.11%	71.91%	/	/	99.60%	38.23%
RN	×	×	/	/	87.23%	79.12%	/	/	79.43%	56.20%
ABL [7]	×	√	/	/	41.11%	31.00%	/	/	37.45%	30.91%
ABL [7]+PRETRAIN	×	√	99.42%	98.89%	99.35%	99.12%	99.10%	98.90%	99.41%	99.12%
DEEPLOGIC$^-$	×	×	64.19%	63.81%	62.10%	62.22%	62.56%	61.44%	60.61%	60.12%
DEEPLOGIC	×	×	99.70%	99.34%	99.92%	99.54%	99.30%	98.95%	99.46%	99.51%

6 EXPERIMENTS

In this section, we evaluate the performance, convergence, stability, and generalization ability of the proposed DeepLogic framework on three logic-reasoning datasets. The first and second datasets are manually constructed from MNIST with multiple attributes and different rules, while the third one is a widely used reasoning dataset designed to evaluate machine's reasoning ability. The detailed information of datasets is summarized as follows:

• **MNIST-ADD**. MNIST-ADD is a simple single-digit addition dataset. The task is to learn "*single-digit addition*" formula given three MNIST images and a 1-bit "`True/False`" label. The dataset includes 20,000 instances for training and 20,000 for testing. We further split the dataset into α and β split with different splitting strategies. In the β split, the testing set has different addition instances from instances in the training set. This setting is also known as "training/testing distribution shift" which is difficult to solve for neural networks.

• **C-MNIST-RULE**. C-MNIST-RULE is an extension to MNIST-ADD where an extra attribute "`color`" and two extra formulas "`progression`" and "`mutual exclusion`" are included. Note that we use the same **DeepLogic** model for MNIST-ADD and C-MNIST-RULE with the only difference lying in the number of output formula Δ_ϕ, which is 1 in and 3 in C-MNIST-RULE. DeepLogic is capable of learning multiple formulas and perceptions simultaneously.

• **RAVEN**. RAVEN [15] dataset is developed with *Raven's Progressive Matrices* [18] to measure visual reasoning ability. Though relaxing all the logic operations to continuous forms using fuzzy logic [14], DLM can be combined with state-of-the-art model CoPINet [38] to obtain significant performance improvement.

6.1 Evaluation on MNIST-ADD

Each instance in this dataset is sampled from single-digit integer addition instances such as ("3+4=7", `True`) and ("4+1=6", `False`). Then, three corresponding MNIST images are sampled as the inputs. There are two splits of this dataset: i) MNIST-ADD-α, where the training and test set both contain all the possible integer addition instances; ii) MNIST-ADD-β, where the training set and test set each includes different (disjoint) sets of instances, *e.g.*, "3+4=7" appears in the training set and does not appear in the test set.

6.1.1 Setup

Configuration. We first define the basic operations and predicates for MNIST-ADD:

1) The operations set includes $Add(\cdot)$ and $Keep(\cdot)$ where the subscription implies the arity;
2) The predicates set includes $Eq(\cdot)$;
3) The logic connectives include $And(\cdot)$, $Or(\cdot)$, $Not(\cdot)$ and $Keep(\cdot)$.

The system is basically implemented with two term layers and one formula layer, and we also provide results with more layers to test the robustness and stability of our proposed DeepLogic framework.

Baselines. We mainly focus on evaluating the model's superiority compared with DNN-based methods, and show that our algorithm is able to learn both the neural perception and logical reasoning jointly. Basically, all the methods share the same **CNN**-based perception model, while the logical reasoning are incorporated into **MLP** [39], **LSTM** [40], two popular models used in traditional DNN methods, and **Relation Network (RN)** [41] which is widely used to model relationships. Under the weak-supervision setting of this paper, these baselines perform worse and only have a final logic accuracy. We thus add extra perception supervisions for them. Besides, we compare our methods with [7] who has an extra predefined out-of-box Prolog [27] program to conduct the logic reasoning.

Evaluation. We use the official MNIST-CNN net [4] to conduct perception across all the approaches. To alleviate overfitting, we preliminarily select the hyperparameters on an extra

4. https://github.com/pytorch/examples/blob/master/mnist/main.py

TABLE 2: Accuracy on C-MNIST-RULE, where f indicates the model is trained with extra symbol annotations and w indicates no extra symbol annotations are involved. DeepLogic$^-$ does not further train p_θ. Both DeepLogic and DeepLogic$^-$ are pretrained for 10 batches. LOGIC is the accuracy of final prediction y while PERCEPTION is the accuracy for predicting the hidden symbol z.

MODEL	f/w	PERCEPTION		LOGIC	
		TRAIN	TEST	TRAIN	TEST
MLP	f	99.82%	99.71%	75.4%	67.57%
LSTM	f	99.73%	99.58%	78.56%	68.37%
RN	f	99.51%	98.95%	50.11%	50.24%
MLP	w	/	/	55.23%	50.12%
LSTM	w	/	/	56.12%	50.49%
RN	w	/	/	50.23%	49.90%
DEEPLOGIC$^-$	w	74.45%	73.28%	70.55%	72.12%
DEEPLOGIC	w	99.88%	99.64%	99.44%	99.54%

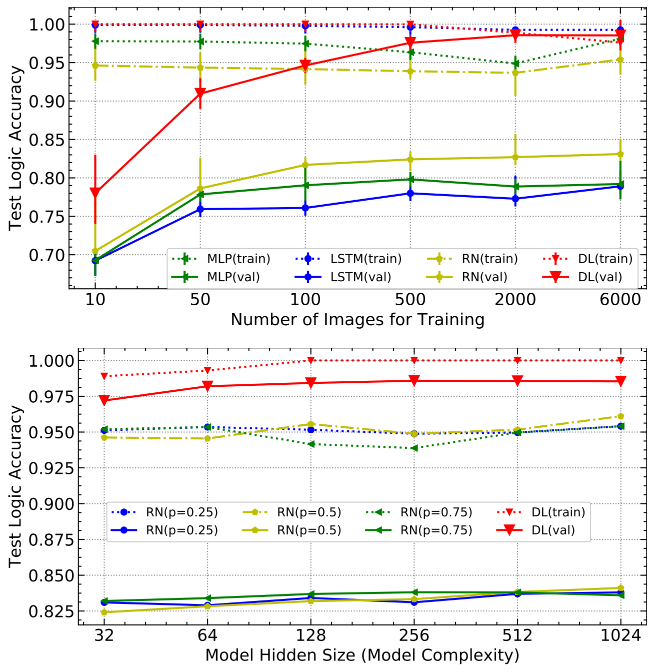

Fig. 4: Top: test accuracy with different scales of training images on MNIST-ADD-α, where DL is short for DeepLogic. Bottom: test accuracy with different model hidden sizes and different dropout probabilities of **RN** and **DL**.

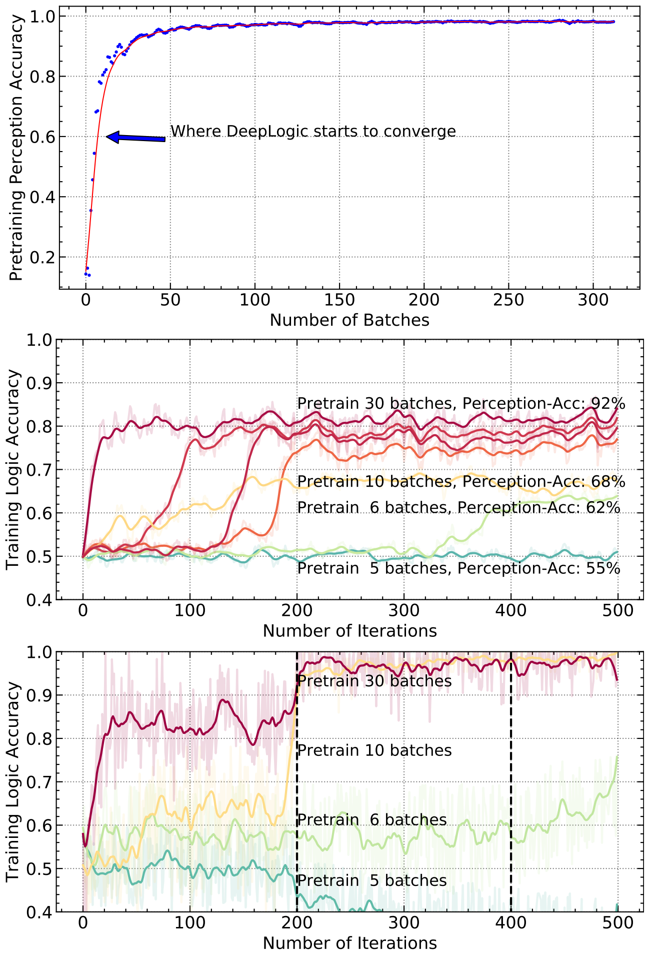

Fig. 5: Top: PERCEPTION accuracy when pretraining p_θ on MNIST-ADD-α dataset; Middle: Training LOGIC accuracy of DeepLogic- with different batches of pretraining data on MNIST-ADD-α datasets; Bottom: Training LOGIC accuracy of DeepLogic with different batches of pretraining data on MNIST-ADD-α dataset. The major findings are: 1) more pretraining batches ensures better accuracy; 2) only very few pretraining is required for DeepLogic to finally converge.

validation set. Finally, we train these models with 20 epochs, and report the performance of the final trained model. Our proposed **DeepLogic** is trained with Alg. 2 using the weak-supervision signals, and DNN-baselines are either trained with the same weak-supervision signals or full-supervision signals with extra digit labels for each MNIST image.

6.1.2 Results and Analysis

Overall Results. The overall results are summarized in Tab. 1. On both MNSIT-ADd-α and MNIST-ADD-β datasets, DNN models overfit to the training set. We further utilize the widely-used methods (changing model size, using dropout, *etc.*) to overcome the overfitting issue, though they are shown to have little help as illustrted in Fig 4 (Bottom). Especially when it comes to the unbalanced β split, those DNN-based models perform extreme worse. They could distinguish those digits and thus have an acceptable perception accuracy, but they fail to model the underlying logic and perform worse with respect to the logic accuracy. Compared with the ABL [7] model, our model could reach a higher accuracy without the out-of-box Prolog [7] program, which is more flexible. Finally, the comparison between DeepLogic$^-$ and DeepLogic shows that back-propagation through the logic (Alg. 1) could indeed help to provide supervision for the perception model.

Data Efficiency. As shown in Fig. 4(top), DeepLogic outperforms its DNN-counterparts across all settings. Moreover, DeepLogic converges to an accuracy of more than 95% only with roughly 100 training images, which is because the neural-symbolic learning actually disentangles the process of neural perception and logical reasoning. Therefore, it is sufficient to train the perception model with quite a few images.

The Necessity of Pretraining. As the theoretical analysis in Sec. 5.2 demonstrates, the whole system would not converge without pretraining. Here, we experimentally validate

TABLE 3: Typical formulas learned under different settings in MNIST-ADD dataset. M denotes the number of term layers and N denotes the number of formula layers. The last column is the percentage of successful convergence in 5 random trials.

M + N	EXAMPLE LEARNED FORMULAS	SUCCESS
$2+1$	$\mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3))$	100%
$2+2$	$\mathrm{Keep}\left(\mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3))\right)$ $\mathrm{And}\left(\mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3)), \mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Add}(z_1,z_2))\right)$	100%
$2+3$	$\mathrm{Or}\left(\mathrm{And}\left(\mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3)), \mathrm{Eq}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3))\right), \mathrm{Not}\left(\mathrm{Eq}(\mathrm{Keep}(z_1),\mathrm{Keep}(z_1))\right)\right)$	100%
$3+1$	$\mathrm{Eq}\left(\mathrm{Add}(\mathrm{Keep}(z_1),\mathrm{Keep}(z_2)), \mathrm{Keep}(\mathrm{Keep}(z_3))\right)$	80%
$3+2$	$\mathrm{And}\left(\mathrm{Eq}\left(\mathrm{Add}(\mathrm{Add}(z_1,z_2),\mathrm{Add}(z_1,z_2)), \mathrm{Add}(\mathrm{Add}(z_1,z_2),\mathrm{Add}(z_1,z_2))\right), \mathrm{Eq}\left(\mathrm{Add}(\mathrm{Add}(z_1,z_2),\mathrm{Keep}(z_3)), \mathrm{Add}(\mathrm{Add}(z_1,z_2),\mathrm{Add}(z_1,z_2))\right)\right)$	80%

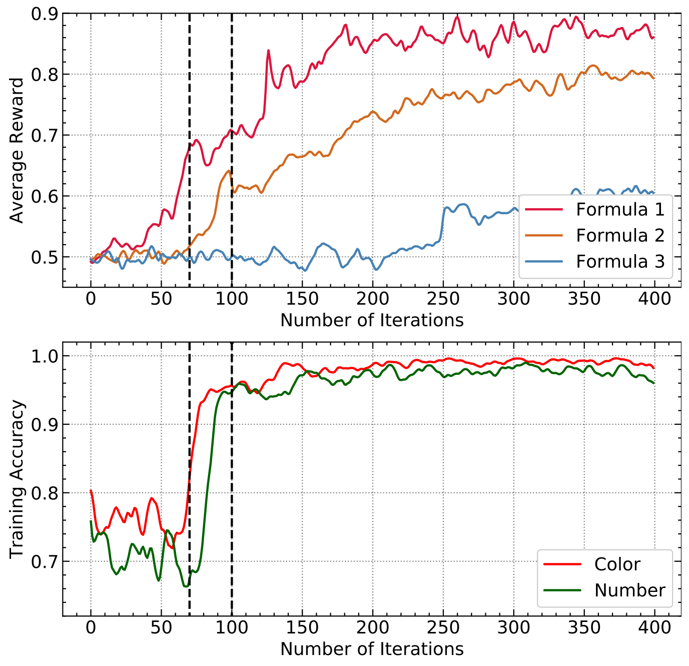

Fig. 6: Average reward of three formulas and accuracy of two attributes of DeepLogic on C-MNIST-RULE. Best viewed in color.

this claim and further show that **the cost of pretraining actually can be very small to guarantee the convergence.** In Fig. 5, we show that DeepLogic requires only 6 batches of pretraining to reach convergence, indicating that we can take only a few efforts in pretraining to finally obtain a well-performed and logically reasonable model. Moreover, Fig. 5 also shows that higher pretraining accuracy will boost the convergence speed of logic learning, which plays an important role in multi-rule and multi-attribute scenarios such as the C-MNIST-RULE dataset (Sec. 6.2).

Model Stability. The most proper setting for this task is *two term layers with one formula layer* to learn "Eq(Add(Z_1, Z_2), Keep(Z_3))". However, the proper setting may not be available in real applications. In Tab. 3, we show the different logic results learned by the system with different settings. Each of these experiments is conducted with 5 different random seeds, and the best learned models are also traced. We observe that the model converges easily even with much redundant information, and the system is also robust with respect to different initializations.

6.2 Evaluation on C-MNIST-RULE.

C-MNIST-RULE contains multiple rules and attributes, where we color MNIST images to add the color property and implement three rules according to Raven's Progressive Matrix (RPM) [18]. Similar to MNIST-ADD, the C-MNIST-RULE dataset includes 20,000 training instances and 20,000 testing instances.

6.2.1 Setup

C-MNIST-RULE is similar with MNIST-ADD except that it contains more attributes and rules.

Attributes. C-MNIST-RULE contains 2 different attributes:

1) **NUMBER**. The digit number of each image.
2) **COLOR**. Each image has a color attribute which is sampled from the color wheel [5] with 12 different colors.

Logic Rules. C-MNIST-RULE implements 3 rules according to [18], which are as follows,

1) **ADD**. The first two attributes can sum up to get the third one. The ground-truth logic formula is $Eq(Add(Z_1, Z_2), Keep(Z_3))$
2) **Progression**. The three attributes are increased or decreased in an equidistant manner, with step size of 1. For example, they could be $3, 4, 5$ or, $5, 4, 3$ respectively. The attributes could be $Eq(Add(Z_1, Z_3), Add(Z_2, Z_2))$.
3) **Mutual Exclusion**. The attributes of the three input images are different with each other, i.e., mutually excluded. These attributes can be logically formalized as $And(Not(Eq(Z_1, Z_2)), Not(Eq(Z_1, Z_3)))$.

In Tab. 2, we show the accuracy of different models on C-MNIST-RULE. We can observe that:

- Pure DNN-based methods get worse performance compared to the results without symbol annotations on C-MNIST-RULE.
- Pure DNN-based methods converge with the help of extra symbol annotations, which is also consistent with [15] where pure DNN or even ResNet fails to perform better than random guess without extra annotations.

In Fig. 6, we show the learning curves of our model on C-MNIST-RULE dataset, and we We discover that:

- *formula 1* converges fast.

5. https://en.wikipedia.org/wiki/Color_wheel

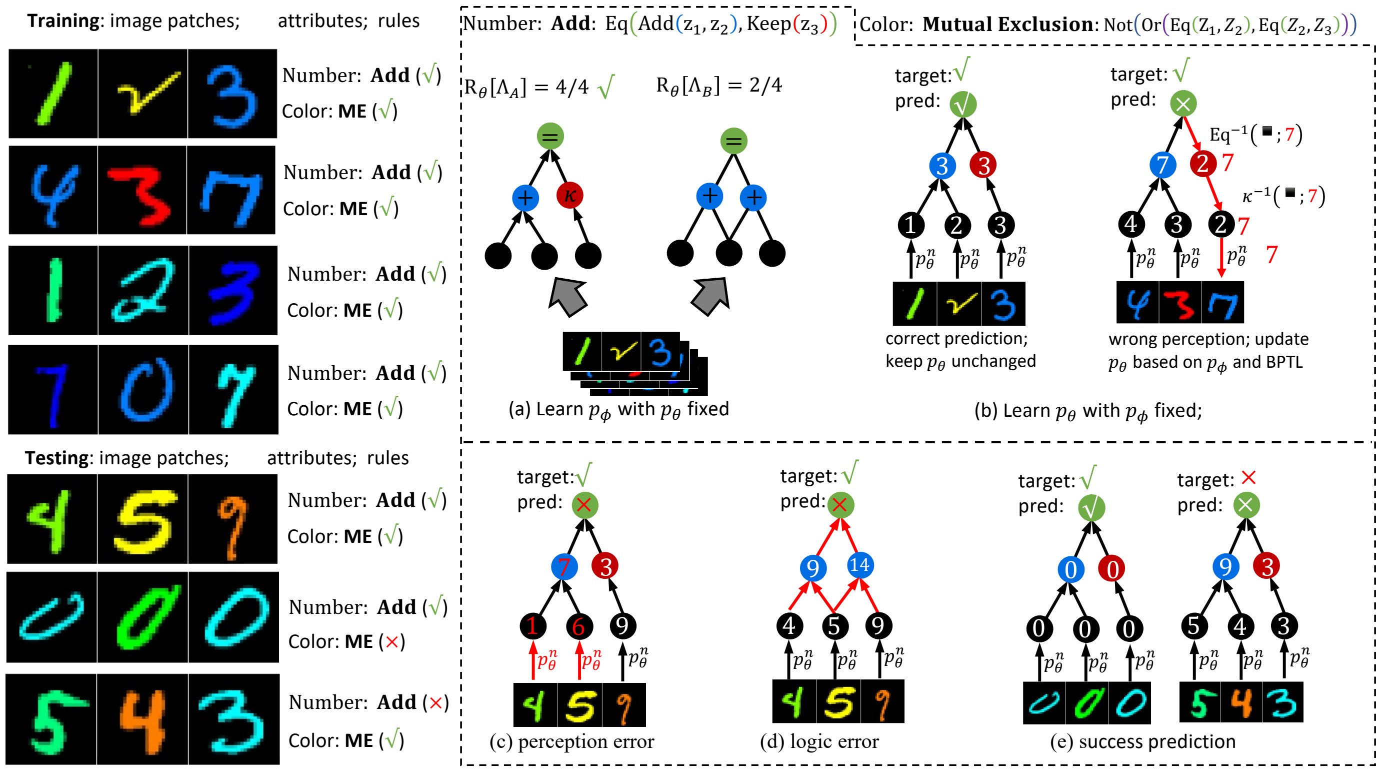
Training: image patches; attributes; rules
Number: Add (√)
Color: ME (√)
Testing: image patches; attributes; rules
Color: ME (×)
Number: Add (×)
Number: Add: Eq(Add(z1, z2), Keep(z3))
Color: Mutual Exclusion: Not(Or(Eq(Z1, Z2), Eq(Z2, Z3)))
Rθ[ΛA] = 4/4 √
Rθ[ΛB] = 2/4
target: √
pred:
target: ×
Eq−1(■; 7)
κ−1(■; 7)
correct prediction; keep pθ unchanged
wrong perception; update pθ based on pϕ and BPTL
(a) Learn pϕ with pθ fixed
(b) Learn pθ with pϕ fixed;
(c) perception error
(d) logic error
(e) success prediction

Fig. 7: Left: data illustrations from C-MNIST-RULE dataset where the **number** attribute follows the **ADD** rule while the **color** attribute follows the **Mutual Exclusion** rule. Right: training/testing illustration for the **number** attribute and **Add** rule. (a) and (b) illustrates the learning process of p_ϕ and p_θ resptively using Eq. 22 and Eq. 16; (c) and (d) are unsuccessful cases due to perception error (non-optimal p_θ) and logic structure error(non-optimal p_ϕ), respetively; (e) successful cases.

- The converged *formula 1* supervises the *Color* property to converge.
- The converged *Color* property further boosts the learning of *formula 2* and others.

In Fig. 7, we illustrate several instances of the C-MNIST-RULE dataset along with the logic back propagation and show the final learned formula.

6.3 Evaluation on RAVEN.

RAVEN [15] is proposed to measure the abstraction and reasoning ability of neural models. Each instance contains an 3×3 image blocks, where the first two rows / columns illustrate one rule sampled from *Arithmetic, Progression, Mutual Exclusion, Constant* on image attributes. The third row / column missed its last image (the 9-th image in the instance), and the task is to find that missing image from several candidates. An exemplar illustration of RAVEN is shown in Fig. 8. More details can be found in [15] and [38].

We make a relaxation in our deep-logic module (DLM) by replacing all the logic operators with continuous representations through fuzzy logic [14] to construct Soft-DLM module. Basically, the new connective operators are defined as:

$$a \vee b \triangleq a + b - ab; \; a \wedge b \triangleq ab; \; \neg a \triangleq 1 - a, \qquad (26)$$

where $a, b \in [0, 1]$ are the relaxation of `True/False`.

By replacing the original fusion method in the state-of-the-art model CoPINet [38] with Soft-DLM module, we can obtain a significant performance improvement, as shown in Tab. 4. Particularly, in those complex scenarios like "2×2" and "3×3", Soft-DLM brings much more performance im-

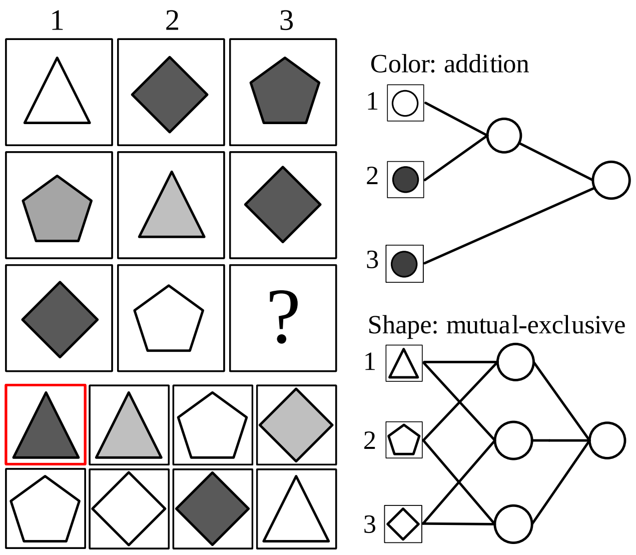

Fig. 8: Illustration of the DLM module in RPM task. Images are treated as inputs and then fed into logic layers, where the logic operation is selected from all the possible candidate combinations.

TABLE 4: Testing Accuracy on RAVEN dataset. ACC is the final accuracy, while other columns represent different task configurations.

METHOD	ACC	CENTER	2X2GRID	3X3GRID	L-R	U-D	O-IC	O-IG
RESNET* [15]	53.43%	52.82%	41.86%	44.29%	58.77%	60.16%	63.19%	53.12%
COPINET [38]	91.42%	95.05%	77.45%	78.85%	99.19%	99.65%	98.50%	91.35%
COPINET+SOFTDLM	**95.60%**	**98.20%**	**85.30%**	**89.65%**	**99.85%**	**99.85%**	**99.55%**	**96.80%**
HUMAN [15]	84.41%	95.45%	81.82%	79.55%	86.36%	81.81%	86.36%	81.81%

provement, which validates the generalizability of DeepLogic and its potential in the continuous domain.

7 CONCLUSION

In this paper, we propose DeepLogic, a novel framework that targets at jointly learning neural perception and logical reasoning for the neural-symbolic learning task. Our proposed DeepLogic framework is able to learn logic formulas and deep perception model with weak 1-bit supervisions. In particular, we design the deep-logic module (DLM) which is capable of representing any first-order logic formula, and we further propose the deep&logic optimization (DLO) algorithm to unify logical reasoning with the neural perception through mutual supervised signals between them. A theoretical proof for the model convergence is also provided to demonstrate the soundness of the proposed DeepLogic framework. Experimental results show that our proposed DeepLogic framework outperforms DNN-baselines by a significant margin. Further analysis validates the convergence, robustness, and generalization ability of DeepLogic, demonstrating its potential to be applied in more complex scenarios.

ACKNOWLEDGMENTS

This research is supported by the National Key Research and Development Program of China (No.2020AAA0106300, No.2020AAA0107800), National Natural Science Foundation of China No.62050110 and Tsinghua GuoQiang Research Center Grant 2020GQG1014.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[3] B. Lake and M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2873–2882.

[4] A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer, and P. Blunsom, "Neural arithmetic logic units," *arXiv preprint arXiv:1808.00508*, 2018.

[5] P. Flach, A. C. Kakas, and A. M. Hadjiantonis, *Abduction and Induction: Essays on Their Relation and Integration*. Springer Science & Business Media, 2000, vol. 18.

[6] J. W. Lloyd, *Foundations of logic programming*. Springer Science & Business Media, 2012.

[7] W.-Z. Dai, Q. Xu, Y. Yu, and Z.-H. Zhou, "Bridging machine learning and logical reasoning by abductive learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 2811–2822.

[8] Y. Bengio, "From system 1 deep learning to system 2 deep learning," in *Thirty-third Conference on Neural Information Processing Systems*, 2019.

[9] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. B. Tenenbaum, "Neural-symbolic vqa: Disentangling reasoning from vision and language understanding," *arXiv preprint arXiv:1810.02338*, 2018.

[10] Y. Yang and L. Song, "Learn to explain efficiently via neural logic inductive learning," *arXiv preprint arXiv:1910.02481*, 2019.

[11] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt, "Deepproblog: Neural probabilistic logic programming," in *Advances in Neural Information Processing Systems*, 2018, pp. 3749–3759.

[12] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," *Journal of Artificial Intelligence Research*, vol. 61, pp. 1–64, 2018.

[13] M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel, "Programming with a differentiable forth interpreter," in *International conference on machine learning*. PMLR, 2017, pp. 547–556.

[14] P. Hájek, *Metamathematics of fuzzy logic*. Springer Science & Business Media, 2013, vol. 4.

[15] C. Zhang, F. Gao, B. Jia, Y. Zhu, and S.-C. Zhu, "Raven: A dataset for relational and analogical visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5317–5327.

[16] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[17] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou, "Neural logic machines," *arXiv preprint arXiv:1904.11694*, 2019.

[18] P. A. Carpenter, M. A. Just, and P. Shell, "What one intelligence test measures: a theoretical account of the processing in the raven progressive matrices test." *Psychological review*, vol. 97, no. 3, p. 404, 1990.

[19] A. Newell, "Physical symbol systems," *Cognitive science*, vol. 4, no. 2, pp. 135–183, 1980.

[20] A. Newell and H. Simon, "The logic theory machine–a complex information processing system," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 61–79, 1956.

[21] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou, "Neural logic machines," *arXiv preprint arXiv:1904.11694*, 2019.

[22] D. A. Hudson and C. D. Manning, "Compositional attention networks for machine reasoning," *arXiv preprint arXiv:1803.03067*, 2018.

[23] M. Zimmer, X. Feng, C. Glanois, Z. Jiang, J. Zhang, P. Weng, L. Dong, H. Jianye, and L. Wulong, "Differentiable logic machines," *arXiv preprint arXiv:2102.11529*, 2021.

[24] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 39–48.

[25] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 804–813.

[26] A. Saha, S. Joty, and S. C. Hoi, "Weakly supervised neuro-symbolic module networks for numerical reasoning," *arXiv preprint arXiv:2101.11802*, 2021.

[27] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[28] Q. Li, S. Huang, Y. Hong, Y. Chen, Y. N. Wu, and S.-C. Zhu, "Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5884–5894.

[29] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, "The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision," *arXiv preprint arXiv:1904.12584*, 2019.

[30] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2901–2910.

[31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[32] Y. Yu, H. Qian, and Y.-Q. Hu, "Derivative-free optimization via classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[33] Y. I. Manin, *A course in mathematical logic for mathematicians*. Springer Science & Business Media, 2009, vol. 53.

[34] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[35] L. Boltzmann, *Lectures on gas theory*. Courier Corporation, 2012.

[36] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[37] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[38] C. Zhang, B. Jia, F. Gao, Y. Zhu, H. Lu, and S.-C. Zhu, "Learning perceptual inference by contrasting," in *Advances in Neural Information Processing Systems*, 2019, pp. 1073–1085.

[39] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in neural information processing systems*, 2017, pp. 4967–4976.

Xuguang Duan is a graduate student at the Department of Computer Science and Technology, Tsinghua University. He got his B.E degree at the Department of Electronic Engineering, Tsinghua University. His research interests include machine learning, neural-symbolic systems, video understanding. He has published some research paper in top conference include NeurIPS, ACM Multimedia etc.

Xin Wang is currently an Assistant Professor at the Department of Computer Science and Technology, Tsinghua University. He got both of his Ph.D. and B.E degrees in Computer Science and Technology from Zhejiang University, China. He also holds a Ph.D. degree in Computing Science from Simon Fraser University, Canada. His research interests include relational media big data analysis, multimedia intelligence and recommendation in social media. He has published several high-quality research papers in top conferences including ICML, KDD, WWW, SIGIR ACM Multimedia etc. He is the recipient of 2017 China Postdoctoral innovative talents supporting program. He receives the ACM China Rising Star Award in 2020.

Peilin Zhao is currently a Principal Researcher at Tencent AI Lab, China. Previously, he worked at Rutgers University, Institute for Infocomm Research (I2R), and Ant Group. His research interests include: Online Learning, Recommendation System, Automatic Machine Learning, Deep Graph Learning, and Reinforcement Learning etc. He has published over 100 papers in top venues, including JMLR, ICML, KDD, etc. He has been invited as a reviewer, area chair or editor for many international conferences and journals, such as ICML, JMLR, etc. He received his bachelor's degree from Zhejiang University, and his PhD degree from Nanyang Technological University.

Guangyao Shen is currently a Ph.D. candidate in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include affective computing and video generation.

Wenwu Zhu is currently a Professor in the Department of Computer Science and Technology at Tsinghua University. He also serves as the Vice Dean of National Research Center for Information Science and Technology, and the Vice Director of Tsinghua Center for Big Data. Prior to his current post, he was a Senior Researcher and Research Manager at Microsoft Research Asia. He was the Chief Scientist and Director at Intel Research China from 2004 to 2008. He worked at Bell Labs, New Jersey as Member of Technical Staff during 1996-1999. He received his Ph.D. degree from New York University in 1996. His research interests are in the area of data-driven multimedia networking and Cross-media big data computing. He has published over 350 referred papers and is the inventor or co-inventor of over 50 patents. He received eight Best Paper Awards, including ACM Multimedia 2012 and IEEE Transactions on Circuits and Systems for Video Technology in 2001 and 2019.

He served as EiC for IEEE Transactions on Multimedia from 2017-2019. He served in the steering committee for IEEE Transactions on Multimedia (2015-2016) and IEEE Transactions on Mobile Computing (2007-2010), respectively.He is an AAAS Fellow, IEEE Fellow, SPIE Fellow, and a member of The Academy of Europe (Academia Europaea).