Out-of-Distribution Generalization by Neural-Symbolic Joint Training

Anji Liu,[2] Hongming Xu,[3] Guy Van den Broeck,[2] Yitao Liang[1,3]

[1] Institute for Artificial Intelligence, Peking University
[2] Computer Science Department, University of California, Los Angeles
[3] Beijing Institute of General Artificial Intelligence (BIGAI)
{liuanji, guyvdb}@cs.ucla.edu, xuhongming@bigai.ai, yitaol@pku.edu.cn

Abstract

This paper develops a novel methodology to simultaneously learn a neural network and extract generalized logic rules. Different from prior neural-symbolic methods that require background knowledge and candidate logical rules to be provided, we aim to induce task semantics with minimal priors. This is achieved by a two-step learning framework that iterates between optimizing neural predictions of task labels and searching for a more accurate representation of the hidden task semantics. Notably, supervision works in both directions: (partially) induced task semantics guide the learning of the neural network and induced neural predictions admit an improved semantic representation. We demonstrate that our proposed framework is capable of achieving superior out-of-distribution generalization performance on two tasks: (i) learning multi-digit addition, where it is trained on short sequences of digits and tested on long sequences of digits; (ii) predicting the optimal action in the Tower of Hanoi, where the model is challenged to discover a policy independent of the number of disks in the puzzle.

1 Introduction

The recent rejuvenation of neural-symbolic synthesis demonstrates that we can significantly benefit from a tight integration of low-level perception and high-level reasoning in tasks featuring out-of-distribution (OOD) generalization (Manhaeve et al. 2018; Xu et al. 2018; Li and Srikumar 2019). It is especially the case on tasks whose semantics are (implicitly) encoded as symbolic rules. Take the multi-digit addition example shown in Fig. 1(left) as an example. Given two numbers represented by a sequence of MNIST images, the goal is to predict the sum of both numbers. Following the orange arrows, it is possible to train neural nets (NNs) to predict the sum directly. When the test sums are i.i.d. w.r.t. the training samples, this task poses no challenge for NNs either. However, the prediction accuracy could degrade if NNs are provided with OOD samples (e.g., unseen combination of digits or numbers with greater length). In contrast, from a neural-symbolic perspective, we can adopt NNs to extract invariant features z (in this case, the digit of every MNIST image) first, and then employ logic rules for multi-digit addition to predict the sum. This model is more robust to various types of distribution shift of the input (e.g., longer numbers) since the NN only needs to accurately predict the digit of MNIST images and the addition rules are invariant to input length.

Some prior work has success in addressing such an integration (Dai and Muggleton 2021). However, they often require abundant prior knowledge. Specifically, most of them require the ground-truth symbolic knowledge to be provided (Yang, Ishay, and Lee 2020; Manhaeve et al. 2018). In other word, such success has not been transferred to the setting where priors on task semantics are limited or not provided. We identify two challenges that hinder the development of neural-symbolic models in this more generalized setting. First, the lack of bidirectional supervision between the neural and symbolic models poses great difficulties for reliably learning meaningful symbolic features z. Specifically, an under-trained neural network cannot produce accurate grounded facts for the symbolic model to learn logic rules, and conversely, the symbolic model cannot provide effective supervision to refine the neural model. Existing approaches alleviate this problem by either providing strong priors on the symbolic rules (Dai and Muggleton 2021; Yang, Ishay, and Lee 2020) or learning symbolic models implicitly (Wang et al. 2019; Amos and Kolter 2017), which cannot deliver reliable reasoning results when facing out-of-distribution samples. Next, even if supervision signals can be properly propagated between the neural and symbolic models, it is still possible that the NN predicts spurious features, leading to bad generalization performance (an example is provided in Sec. 6).

This paper makes an initial attempt to solve these crucial yet under-explored problems. Both aforementioned challenges are mainly attributed to the gigantic set of candidate rules considered by the symbolic model. In one direction, a large number of plausible rules result in noisy update signals to the NN. In the other direction, it is hard to search for good symbolic rules from a vast candidate set. Instead of adding prior knowledge to shrink the rule space, we propose to control the symbolic model's complexity by gradually adding plausible candidate rules with a *symbolic search step*. That is, the model is first initialized with a few "general" rules. After eliminating candidates that are very unlikely to hold, we invoke the symbolic search algorithm to extend and specialize the remaining rules. This process continues until the

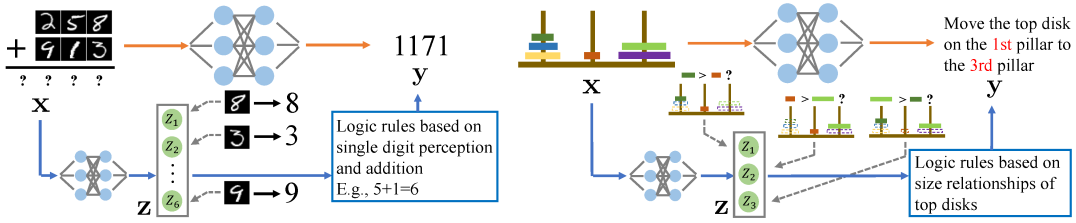

Figure 1: Illustration of the multi-digit addition task (left) and the Tower of Hanoi optimal action prediction task (right). Following the orange arrows, the output $\boldsymbol{y}$ of both tasks can be predicted end-to-end with NNs. However, the features captured by such models are unlikely to generalize well to different scenarios (e.g., increased digit length and more disks in Hanoi). Instead, following the blue arrows, NNs are used to capture invariant features $\boldsymbol{z}$ in the task, and then apply logical reasoning to obtain the answer $\boldsymbol{y}$. Accuracy will not degrade if $\boldsymbol{z}$ is invariant to the change of task scenarios.

model converges to a set of specific rules. To improve effectiveness of the candidate set expansion procedure, we borrow wisdom from the knowledge compilation community, and use Logic Circuits (LCs) (Darwiche 2011) to compactly represent candidate rules. Rule expansion is done by efficient structure learning of the LC. Further, by making use of other properties of LCs (e.g., differentiability), we propose NTOC , a neural-symbolic joint training method that learns invariant features in the NN and generalizable task semantics in the LC.

To understand when the proposed model can generalize well to unseen scenarios, we identify a special class of OOD tasks. We show that the multi-digit addition task and the Tower of Hanoi optimal action prediction task shown in Fig. 1 fit in this class of OOD. Furthermore, we demonstrate that our NTOC out-performs NNs and neural-symbolic models on both tasks by a large margin.

2 Notation and OOD Generalization

Notation We write Boolean variables as uppercase letters (e.g., X) and their assignments as lowercase letters (e.g., x). Analogously, sets of Boolean variables and their joint assignments are denoted by bold uppercase (e.g., $\mathbf{X}$) and lowercase (e.g., $\boldsymbol{x}$) letters, respectively. Define the set of all values $\boldsymbol{x}$ for variables $\mathbf{X}$ as $\mathsf{val}(\mathbf{X})$. A literal represents a variable (e.g., X) or its negation (e.g., $\neg X$). Propositional logic formulas are constructed in the usual way, from literals and logical connectives.

OOD Generalization Following Ye et al. (2021)'s formulation, we consider supervised learning tasks where the model has access to samples from a set of training domains $\mathcal{E}_{\text{train}}$, while at test time it is required to make predictions on samples drawn from a set of test domains $\mathcal{E}_{\text{test}}$. For example, in the digit addition task (Fig. 1(left)), samples from $\mathcal{E}_{\text{train}}$ and $\mathcal{E}_{\text{test}}$ may contain numbers with lengths $\{2, 3\}$ and $\{5, 6\}$, respectively. Similarly, in the Hanoi task (Fig. 1(right)), samples from $\mathcal{E}_{\text{test}}$ may contain more disks then those from $\mathcal{E}_{\text{train}}$. To achieve OOD generalization, models need to extract features that are *invariant* across both domains. However, since $\mathcal{E}_{\text{test}}$ is invisible during training, it is impossible to achieve OOD generalization without proper assumptions (Blanchard, Lee, and Scott 2011; Deshmukh et al. 2019). One reasonable assumption is that features invariant across $\mathcal{E}_{\text{train}}$ maintain their invariance in $\mathcal{E}_{\text{test}}$. Intuitively, if the domains in $\mathcal{E}_{\text{train}}$ and $\mathcal{E}_{\text{test}}$ share invariant features, it is unlikely that spurious features that fail to generalize to test domains work well across all domains.

A Special Class This paper specifically considers OOD tasks with a special type of invariant features; the features are represented as a set of symbols, whose relation with the output can be quantified by a set of logical rules. For example, Fig. 1(right) illustrates the Tower of Hanoi problem, where the task is to predict the next move given the current state represented as a single image. One possible set of invariant features $\boldsymbol{z}$ consists of three boolean variables, each representing the size relation between the top disks of two pillars. They are invariant because NNs equipped with attention mechanism can learn to "focus on" the top disks regardless of the total number of disks. The features are symbolic in the sense that there exits a set of logic rules that map them to the optimal actions. We reveal the logical rules in Sec. 6.

3 Neural-Symbolic Joint Learning

This section elaborates the high-level design principles of our proposed neural-symbolic joint learning framework. As shown in Fig. 2(a), the joint learning model consists of a NN $f_{\boldsymbol{\varphi}}$ and a logical circuit model $g_{\boldsymbol{\theta}}$, parameterized by $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$, respectively. NN $f_{\boldsymbol{\varphi}}$ maps input $\boldsymbol{x}$ to a $|\mathbf{Z}|$-dimensional vector $\mathbf{p}^{\text{nn}}$, where $\forall i \in [|\mathbf{Z}|]$, $\mathbf{p}^{\text{nn}}_i$ denotes the probability of variable $Z_i = \texttt{true}$. Logic circuit (LC) $g_{\boldsymbol{\theta}}$ compactly represents a set of weighted propositional logic rules, where the weights characterizes the importance of the corresponding rules. LC $g_{\boldsymbol{\theta}}$ takes $\mathbf{p}^{\text{nn}}$ and $\boldsymbol{y}$ as inputs, and outputs a scalar representing the probability of satisfaction. The LC as a whole encodes potential task semantics between the NN predicted features $\boldsymbol{z}$ and class labels $\boldsymbol{y}$.

The training pipeline of the proposed model is illustrated in Fig. 2(b). First, we train the NN with some primitive tasks, such as teaching it size relationships between two disks in the Hanoi problem. The pretraining step does not need to teach the NN complete semantics of the input. And it is used to provide a good initialization point to optimize the neural-symbolic model.[1] Afterwards, training proceeds by iterating over the *symbolic rule search* and the *neural model learning* steps, where the goal is to simultaneously learn to predict invariant features $\boldsymbol{z}$ and induce the logical rules. Among the three phases, the symbolic rule search step is task-independent, while the others could be task-dependent.

[1] Alternative approaches such as loss that encourages large variance in $\boldsymbol{z}$ can be used in replacement of pretraining.

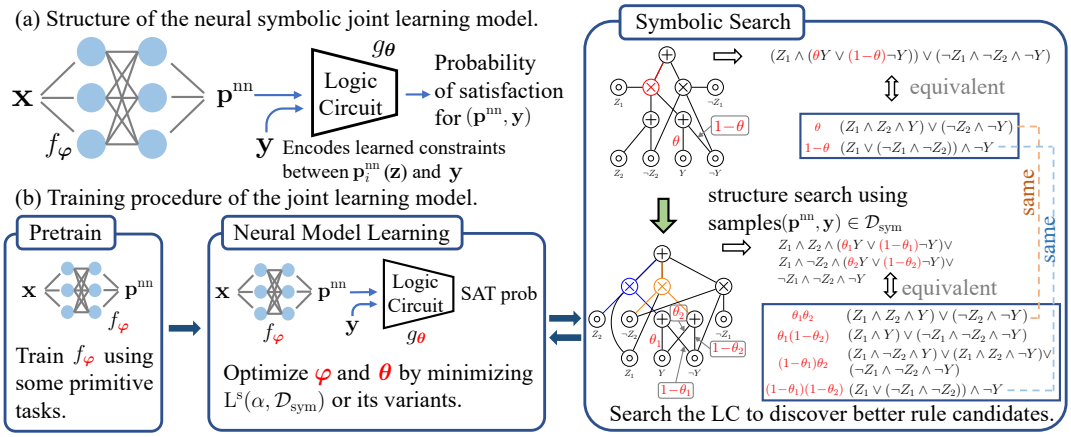

Figure 2: Illustration of the proposed neural symbolic joint learning framework. (a) A NN f_{φ} predicts the probability (i.e., $\mathbf{p}^{\text{nn}}$) of a set of logical variables (i.e., $\mathbf{Z}$). A symbolic model g_{θ} termed Logic Circuit encodes logical constraints among variables $\mathbf{Z}$ $\mathbf{Y}$, and outputs the SAT probability of ($\mathbf{p}^{\text{nn}}$ $\boldsymbol{y}$). (b) The NN of the proposed framework is first pretrained with some primitive tasks. Then, learning proceeds by iteratively updating learnable parameters $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$ in the *neural learning* step and searching a good structure of the Logic Circuit in the *symbolic search* step.

Specifically, the perception model needs to accommodate the type of input data $\boldsymbol{x}$ (e.g., CNNs for images), and the primitive pretraining task are designed to address the necessary perception need of solving the task, with minimal prior regarding the underlying invariance of the task (e.g., digit classification for learning multi-digit addition, and disk size comparison for tackling Hanoi).

3.1 Symbolic Rule Search

Given a set of probabilistic facts $\mathbf{p}^{\text{nn}}$ predicted by the NN and the corresponding labels $\boldsymbol{y}$, we denote $\mathcal{D}_{\text{sym}} := (\mathbf{p}^{\text{nn},(i)}\ \boldsymbol{y}^{(i)})_{i=1}^{N}$, where $(\boldsymbol{x}^{(i)}\ \boldsymbol{y}^{(i)})_{i=1}^{N}$ are training samples and $\mathbf{p}^{\text{nn},(i)} = f_{\boldsymbol{\varphi}}(\boldsymbol{x}^{(i)})$. The goal of the symbolic rule search step is to adjust the set of rules (i.e., propositional logic formulas) embedded in $g_{\boldsymbol{\theta}}$ such that (i) new rules satisfied by the samples in $\mathcal{D}_{\text{sym}}$ with high probability are added, and (ii) rules that are strongly against $\mathcal{D}_{\text{sym}}$ are pruned away.

Concretely, Fig. 2(b) demonstrates an example where the top logical model is transformed to the bottom one by conducting structure search using $\mathcal{D}_{\text{sym}}$. As shown on the right side, both logical models can be written as parameterized propositional logic formulas, which are equivalently expressed as the set of weighted logical formulas shown in the respective boxes. Comparing the two rule sets, we observe that this example structure search step adds two new rules.

Having demonstrated how to adjust embedded rules by conducting structure search on the logical model, the next immediate question is *how to encourage the emergence of good rules?* Since good logic rules should be satisfied (with high probability) by samples in $\mathcal{D}_{\text{sym}}$, we define the objective as maximizing their *probability of satisfaction*, defined via the Semantic Loss (Xu et al. 2018):

Definition 1 (Semantic Loss)**.** For a set of Boolean variables $\mathbf{Z}$, denote $\mathbf{p}$ as a $\mathbf{Z}$ -dimensional vector containing a probability for each variable in $\mathbf{Z}$, and α as a propositional logic sentence over $\mathbf{Z}$. The semantic loss between α and $\mathbf{p}$ is defined as

$$\mathrm{L}^{\text{s}}(\alpha\ \mathbf{p}) := -\log \sum_{\boldsymbol{z} \models \alpha} \prod_{i:\boldsymbol{z} \models Z_i} \mathbf{p}_i \prod_{j:\boldsymbol{z} \models \neg Z_j} (1 - \mathbf{p}_j) \qquad (1)$$

where $=$ denotes logical entailment. Logic sentence α entails β iff all assignments that satisfy α also satisfy β. Intuitively, $\mathrm{L}^{\text{s}}(\alpha\ \mathbf{p})$ becomes smaller as the prediction $\mathbf{p}$ is closer to satisfying α, and $\mathrm{L}^{\text{s}}(\alpha\ \mathbf{p})$ reaches its minimum 0 if $\mathbf{p}$ entails α. In the context of $g_{\boldsymbol{\theta}}$ and $\mathcal{D}_{\text{sym}}$, our goal is to add logic rules α with small semantic loss $\mathrm{L}^{\text{s}}(\alpha\ \mathcal{D}_{\text{sym}}) := \sum_{(\mathbf{p}^{\text{nn}},\boldsymbol{y})\in\mathcal{D}_{\text{sym}}} \mathrm{L}^{\text{s}}(\alpha\quad \boldsymbol{y}\ \mathbf{p}^{\text{nn}})$ to the LC through some structure transformation steps.

However, minimizing the semantic loss alone does not guarantee finding good symbolic rules since the semantic loss does not control the specificity of the logic formula. For example, if α models a tautology over $\mathbf{Z}$, $\mathrm{L}^{\text{s}}(\alpha\ \mathcal{D}_{\text{sym}})$ remains zero regardless of $\mathcal{D}_{\text{sym}}$ since any value $\boldsymbol{z}$ $\mathsf{val}(\mathbf{Z})$ entails tautology. To avoid such trivial rules, we additionally use the model count (i.e., number of satisfying assignments) to control the specificity of the learned logic formula. In summary, this step searches for logic formulas that (i) match the dataset $\mathcal{D}_{\text{sym}}$ well and (ii) are specific enough (have relatively low model count) to contain useful information. In Sec. 5.1, we will demonstrate a practical implementation of this symbolic search algorithm.

3.2 Neural Model Learning

As illustrated in Fig. 2(b), the neural learning step is responsible for jointly optimizing the parameters in $f_{\boldsymbol{\varphi}}$ and $g_{\boldsymbol{\theta}}$. That is, to adjust the NN to predict invariant features $\boldsymbol{z}$ while selecting informative rules in the LC by maximizing its corresponding weight. Different from the well-studied task of learning NNs under the supervision of groundtruth symbolic knowledge (Manhaeve et al. 2018; Xu et al. 2018; Li and Srikumar 2019; Xie et al. 2019), in the joint training framework we do not have access to groundtruth rules. Instead, the symbolic model $g_{\boldsymbol{\theta}}$ encodes a weighted set of rules. Concretely, suppose $g_{\boldsymbol{\theta}}$ encodes k logic rules α_1 α_k, whose

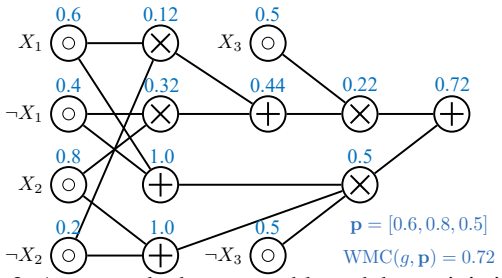

Figure 3: An example decomposable and deterministic LC g representing $X_3 \quad (X_1 \quad X_2) \quad (\quad X_1 \quad X_2)$. WMC of g w.r.t. $\mathbf{p} = [0\ 6\ \ 0\ 8\ \ 0\ 5]$ is labeled blue beside each node.

weights are $\theta_1 \qquad \theta_k$, respectively.[2] Both models are updated by minimizing the following objective:

$$\underset{\boldsymbol{\varphi},\boldsymbol{\theta}}{\text{minimize}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_{\text{train}}} \sum_{i=1}^{k} \theta_i \cdot \mathrm{L}^{\mathrm{s}}(\alpha_i \quad \boldsymbol{y} \;\; f_{\boldsymbol{\varphi}}(\boldsymbol{x})) \qquad (2)$$

where $\sum_{i=1}^{k} \theta_i = 1$. This objective is minimized when rules α_i closer to the (unknown) groundtruth have high weights, and the neural network learns to predict $\mathbf{p}$s that best satisfy these "likely" rules. In this way, we cast the logic rule selection problem into a continuous optimization problem. In Sec. 5.2, we will concretize this idea by showing how the rules are compactly represented and how to learn the weights θ together with the parameters in the neural network.

4 Representation of the Symbolic Model

In this section, we first discuss the requirements on the symbolic model posed by the symbolic rule search step and the neural model learning step (Sec. 4.1). This motivates the use of circuit representations, as they satisfy all requirements. We then conclude this section with necessary backgrounds on circuit representations (Sec. 4.2).

4.1 Key Requirements on the Symbolic Model

Firstly, a crucial requirement on implementing the symbolic rule search algorithm is to efficiently compute both search objectives: the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha\ \mathcal{D}_{\mathrm{sym}})$ and the model count w.r.t. propositional logic sentence α. Examining the definition of semantic loss, we find that computing $\mathrm{L}^{\mathrm{s}}(\alpha\ \mathbf{p})$ can be cast to a well-known automated reasoning task termed weighted model counting (WMC) (Chavira and Darwiche 2008; Sang, Beame, and Kautz 2005a,b) that generalizes model counting. Therefore, the first requirement is reduced to efficient computation of WMC. Secondly, the symbolic representation needs to be general enough to represent various logical sentences and flexible enough to support efficient search of new logical rules. Thirdly, in the neural model learning phase, we backpropagate the gradients through the symbolic model to $\mathbf{p}^{\mathrm{nn}}$ to update the neural network f. Therefore, we require the symbolic model's objective (Eq. (2)) to be differentiable w.r.t. its input $\mathbf{p}^{\mathrm{nn}}$.

Circuit representations (Darwiche 2011) satisfy all above requirements. On the one hand, they can represent all possible propositional logical rules, and support linear-time (w.r.t. model size) computation of WMC. On the other hand, circuits are computation graphs with sum and product computational units, supporting gradient computation. We proceed to formally introduce circuit representations.

[2] In LCs, the probability of a rule is implicitly defined by $\boldsymbol{\theta}$, which allows us to represent exponentially many rules w.r.t. $|\boldsymbol{\theta}|$.

4.2 Circuit Representations and the Computation of Weighted Model Counting

Circuits represent functions, including logical sentences, as (parametrized) computation graphs (Darwiche 2011; Darwiche and Marquis 2002). By imposing different structural constraints, circuits can compute various logical and probabilistic queries efficiently. The basic syntax and semantics of circuits are formalized in the following.

Definition 2 (Circuits)**.** A circuit $g(\mathbf{X})$ encodes a function over $\mathbf{X}$ via a parametrized directed acyclic graph (DAG) with a single root node n_r. Similar to neural networks, every node of the DAG defines a computational unit. Specifically, each leaf node corresponds to an *input unit* represented by a literal (e.g., X and $\quad X$), and each inner node n is either a *sum* or *product unit* that receives inputs from its children, denoted $\mathsf{in}(n)$. Let $\mathbf{p} \quad [0\ \ 1]^{|\mathbf{X}|}$ be a vector of probabilities for variables $\mathbf{X}$. The output $g_n(\mathbf{p})$ of each node n given input $\mathbf{p}$ is defined recursively as

$$g_n(\mathbf{p}) = \begin{cases} h_n(\mathbf{p}) & \text{if } n \text{ is an input unit} \\ \sum_{c\in\mathsf{in}(n)} \theta_{n,c} \cdot g_c(\mathbf{p}) & \text{if } n \text{ is a sum unit} \\ \prod_{c\in\mathsf{in}(n)} g_c(\mathbf{p}) & \text{if } n \text{ is a product unit} \end{cases} \qquad (3)$$

where h_n is any univariate function defined by n, and $\theta_{n,c}$ denotes the parameter corresponds to edge $(n\, c\,)$. In particular, Logic Circuits (LCs) are a subset of circuits where $h_n(\mathbf{p}) = \mathbf{p}_i$ if n encodes literal X_i and $h_n(\mathbf{p}) = 1 - \mathbf{p}_i$ if n encodes literal $\quad X_i$. For ease of notation, we define $g_n(\boldsymbol{x})$ as $g_n(\mathbf{p}^{\boldsymbol{x}})$, where $\mathbf{p}_i^{\boldsymbol{x}} = 1$ for all i such that $\boldsymbol{x} = X_i$ and $\mathbf{p}_i^{\boldsymbol{x}} = 0$ if $\boldsymbol{x} = \quad X_i$. The size of a circuit g, denoted $\quad g\ $, is the number of edges in its DAG.

This paper focuses on LCs that (i) represents a set of weighted logic formulas and (ii) supports linear time (w.r.t. $\ g\ $) computation of arbitrary weighted model counting queries.[3] For example, the LC shown in Fig. 3 represents the formula $X_3 \quad (X_1 \quad X_2) \quad (\quad X_1 \quad X_2)$. By slightly abusing notation, for any LC g_n, we write $\boldsymbol{x} = g_n$ if $\boldsymbol{x}$ entails the logic formula of g_n. The WMC of g_n w.r.t. weights $\mathbf{p}$, formally defined as $\sum_{\boldsymbol{x}\models g_n} \prod_{i:\boldsymbol{x}\models X_i} \mathbf{p}_i \prod_{j:\boldsymbol{x}\models\neg X_j} (1-\mathbf{p}_j)$, can be computed by a feedforward evaluation of g_n following Eq. (3) (Kimmig, Broeck, and De Raedt 2012). For instance, the WMC between the LC in Fig. 3 and $\mathbf{p} = [0\ 6\ \ 0\ 8\ \ 0\ 5]$ is $0\ 72$. Since the semantic loss between g_n and $\mathbf{p}$ is the negative logarithm of the corresponding WMC, it can also be computed efficiently. Additionally, since the feedforward computation consists of only additions and multiplications, the gradient $\quad_{\mathbf{p}} \mathrm{L}^{\mathrm{s}}(g_n \ \mathbf{p})$ can be computed by backpropagating through the DAG of g_n.

[3] For (ii) to hold, circuits must be both decomposable and deterministic. We kindly refer readers to (Darwiche 2011), who are interested in why these two properties are necessary.

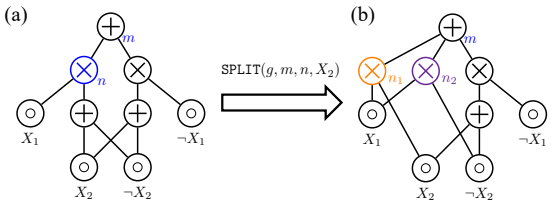

Figure 4: An example of the SPLIT operation.

Algorithm 1: Symbolic Rule Search

1: **Input:** A dataset of probabilistic facts $\mathcal{D}_{\text{sym}} = \{\mathbf{p}^{(i)}\}_{i=1}^{N}$
2: **Output:** A logic circuit g
3: **Initialize:** g as a LC representing tautology
4: **while** termination conditions not met **do**
5: $(m, n), v \leftarrow$ select an edge and a variable to SPLIT
6: $g \leftarrow \texttt{SPLIT}(g, m, n, v)$
7: Prune worlds in g that do not explain $\mathcal{D}_{\text{sym}}$
8: **end while**

5 Practical Implementation

This section introduces the implementation details of our proposed algorithm Neural to Circuit (NTOC). Details of the symbolic rule search and the neural model learning algorithms are provided in Sec. 5.1 and 5.2, respectively.

5.1 Symbolic Rule Search as Structure Learning

Recall the symbolic rule search step aims to find propositional logic rules that best explain the dataset $\mathcal{D}_{\text{sym}}$. Different from existing approaches that select a good subset of predefined rules, we directly learn a good LC $g_{\boldsymbol{\theta}}$. This avoids the need to provide task-specific background knowledge and allows us to explicitly control the balance between how well samples in $\mathcal{D}_{\text{sym}}$ satisfy g (i.e., $\mathrm{L}^{\text{s}}(g \;\; \mathcal{D}_{\text{sym}})$) and the specificity of g (i.e., its model count).

The proposed symbolic rule search algorithm (Alg. 1) takes $\mathcal{D}_{\text{sym}}$ as input, and returns a LC representing the logic rules. The algorithm starts with a LC that represents tautology (line 3). At this point, the semantic loss $\mathrm{L}^{\text{s}}(g \;\; \mathcal{D}_{\text{sym}}) = 0$ and the model count of g is at its maximum. Alg. 1 then iteratively restricts g by pruning worlds $\mathbf{w}$ that are likely untrue. This will lead to increase of the semantic loss and decrease of the model count. The goal here is to prune worlds with a good strategy to minimize the increase of $\mathrm{L}^{\text{s}}(g \;\; \mathcal{D}_{\text{sym}})$ (i.e., maintain the correctness of the rules) while maximize the decrease in model count (i.e., improve rule completeness).

Iterative pruning relies on the fact that every node n in a LC corresponds to a logic sentence α_n, thus pruning a node is equivalent to discarding the worlds that entail α_n. The main loop (lines 4-8) consists of two key steps: (i) apply transformations to the LC to expose prunable nodes (lines 5-6), and (ii) prune nodes by predefined criterions (line 7). We proceed to describe both steps in detail.[4]

Step #1: SPLIT We use the SPLIT circuit transformation (Liang, Bekker, and Van den Broeck 2017) to discover candidate nodes for pruning. Specifically, SPLIT takes as input an edge $(m\, n\,)$ (m is a parent of n; m is a sum unit and n is a product unit) and a variable $X \quad \phi(n)$, where $\phi(n)$, the scope of n, is defined as the collection of variables defined by all its descendent input nodes. Suppose n represents logic sentence α_n, SPLIT first constructs two product units n_1 and n_2, which represent $\alpha_n \quad X$ and $\alpha_n \quad\quad X$, respectively. The algorithm then replaces $(m\, n\,)$ with two edges $(m\, n\,_1)$ and $(m\, n\,_2)$. As an example, after applying SPLIT to the LC in Fig. 4(a) w.r.t. $(m\, n\,)$ and X_2, the circuit is transformed to the one shown in Fig. 4(b). Since $\alpha_n = (\alpha_n \quad X) \;\; (\alpha_n \quad\quad X)$, SPLIT does not change the LC's semantics.

SPLIT can generate prunable units in the LC. Consider again the example in Fig. 4. If the groundtruth sentence is $\beta = \quad X_1 \quad\quad X_2$, it is impossible to obtain a corresponding LC by pruning nodes from the LC in Fig. 4(a), since we cannot only eliminate the world $\quad \beta = X_1 \quad X_2$. However, after SPLIT, we can eliminate $X_1 \quad X_2$ by pruning away n_1.

Another key process in this step is to select which edge and variable to SPLIT. Specifically, we select SPLIT candidates by assigning a score to each edge-variable pair, and choose the candidate with the highest score. The score is the multiplication of two values: one measuring the number of samples in $\mathcal{D}_{\text{sym}}$ that "activate" the edge, and the other estimating the information gain of the split.

Step #2: node pruning We eliminate worlds from g via two criterions. The first is to prune away units that almost do not satisfy any sample. That is, the change in $\mathrm{L}^{\text{s}}(g \;\; \mathbf{p})$ is small for (almost) all samples. To accelerate the pruning algorithm, we use a second criterion termed prune by conjunction. Specifically, for unit n in LC g, denote g' as the LC obtained by conjoining n with a literal (e.g., X, $\quad X$). If $\mathrm{L}^{\text{s}}(g' \;\; \mathbf{p}) - \mathrm{L}^{\text{s}}(g \;\; \mathbf{p})$ is small for (almost) all samples, we eliminate worlds from g by conjoining n with the literal.

Termination conditions The trade-off between the accuracy and specificity of the learned rules is controlled by the termination condition. Specifically, define the improvement of g' over g as

$$I(g \;\; g') := \frac{\mathrm{L}^{\text{s}}(g' \;\; \mathcal{D}_{\text{sym}}) - \mathrm{L}^{\text{s}}(g \;\; \mathcal{D}_{\text{sym}})}{\texttt{MC}(g) - \texttt{MC}(g')}$$

which measures the increase in semantic loss per decrease in model count. The smaller $I(g \;\; g')$ is, the better the quality of the transformation from g to g', as it significantly decreases the model count while not increasing the semantic loss by too much. Define g^0 as the initial LC representing tautology and g^t as the circuit learned at iteration t. We terminate Alg. 1 at step t if $I(g^{t-1} \;\; g^t) > d \cdot I(g^0 \;\; g^t)$, where d is a hyperparameter chosen as 5 in all our experiments.

Computational complexity Since WMC is #P-hard in general, for tasks with complex underlying logic rules, neural-symbolic methods that explicitly use these groundtruth rules (Manhaeve et al. 2018; Xu et al. 2018) could be extremely slow for computing the semantic loss, which makes them less scalable when facing such tasks. In contrast, NTOC can gradually search for good approximations of the groundtruth rules, achieving a better balance between rule correctness and computational complexity.

5.2 NN Learning using Probabilistic Logic

Given task semantics represented by LC g, the neural model learning step aims to maximize the probability of $\mathbf{p}^{\text{nn}}$ satisfying $g \quad \boldsymbol{y}$, where $\mathbf{p}^{nn}$ is a vector of probabilities for $\boldsymbol{z}$ and is the output of neural network f. That is, the goal is to minimize the semantic loss $\mathrm{L}^{\text{s}}(g \quad \boldsymbol{y} \;\; \mathbf{p}^{\text{nn}})$, which can be

[4] Details such as hyperparameters are given in Appx. A.1.

optimized by gradient-based algorithms since $\nabla_{\mathbf{p}^{\text{nn}}} \mathrm{L}^{\text{s}}(g, \boldsymbol{y}, \mathbf{p}^{\text{nn}})$ can be computed efficiently.

When the LC is learned from data and prone to error, directly minimizing the semantic loss could lead to catastrophic failure. Take the two-digit addition task illustrated in Sec. 2 as an example. Denote $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ as the symbolic representations of both images, respectively. Suppose that in addition to the groundtruth addition rules, g contains the following rule: ($\boldsymbol{z}_1 =$ "digit 0") $\wedge$ ($\boldsymbol{z}_2 =$ "digit 0"). g is still correct in the sense that if the neural network correctly learns to classify all digits, the semantic loss can still converge to its minimum 0. However, in practice, minimizing the semantic loss will likely lead to the trivial solution of predicting all images as digit 0, which also minimizes the semantic loss.

To avoid such local minima, we make use of the fact that for each $\boldsymbol{z} \in \mathsf{val}(\mathbf{Z})$, only one $\boldsymbol{y} \in \mathsf{val}(\mathbf{Y})$ is true. Specifically, we enforce this constraint by maximizing the following objective:

$$\frac{-\mathrm{L}^{\text{s}}(g \wedge \boldsymbol{y} \wedge \mathbf{p}^{\text{nn}})}{\log \sum_{\boldsymbol{y}' \in \mathsf{val}(\mathbf{Y})} \exp(-\mathrm{L}^{\text{s}}(g \wedge \boldsymbol{y}' \wedge \mathbf{p}^{\text{nn}}))} \tag{4}$$

which pushes $\mathbf{p}^{\text{nn}}$ to satisfy $g \wedge \boldsymbol{y}$ ($\boldsymbol{y}$ is the label) but not $g \wedge \boldsymbol{y}'$ ($\boldsymbol{y}' \in \mathsf{val}(\mathbf{Y}) \wedge \boldsymbol{y}' \neq \boldsymbol{y}$).

While the above objective can avoid NN from being trapped in certain local optima, neural learning could still fail if g is not informative (e.g., it is a tautology). Fortunately, since in this case the logic rules are often too general, it is likely that the groundtruth logic sentence α^* entails g. That is, following the main idea in Sec. 3.2, if we properly select a set of sentences $\{\alpha_i\}_{i=1}^{k}$ that contains close-to-optimal ones, we can jointly refine logic rules g and the neural network f.

For LCs, learnable parameters are assigned to the edge weights $\theta_{n,c}$ defined in Def. 2. Denote the equivalent logical formula of LC unit n as α_n. Since the LCs we learn are deterministic, for every sum unit n, the logic formulas of its children, i.e., $\{\alpha_c : c \in \mathsf{in}(n)\}$, are mutually exclusive: $\forall c_1, c_2 \in \mathsf{in}(n)(c_1 \neq c_2) \rightarrow \alpha_{c_1} \wedge \alpha_{c_2} = \texttt{false}$. According to Def. 2, the edge weights of n (i.e., $\{\theta_{n,c} : c \in \mathsf{in}(n)\}$) sum up to 1. By making analogy with Eq. (2), each edge parameter $\theta_{n,c}$ can be viewed as the weights associated with logic formula α_c. Therefore, circuit representations provide a natural way to incorporate learnable parameters that weight the logical formulas. Joint learning of the LC weights and the NN parameters are simple, since parameterized LCs are still differentiable, and the LC weights can be learned via well-developed EM algorithms (Liu and Van den Broeck 2021).

A final question is which LC units should we weigh. While it is possible to weigh all edges, adding too many weights in the LC could render the learning process unstable. We choose to add learnable parameters to every sum unit n whose scope $\phi(n)$ is $\mathbf{Y}$, for the following reason. In classification tasks, for any $\boldsymbol{z} \in \mathsf{val}(\mathbf{Z})$, only one $\boldsymbol{y} \in \mathsf{val}(\mathbf{Y})$ holds. Therefore, adding weights to LC unit n with scope $\mathbf{Y}$ allows the model to select a correct $\boldsymbol{y}$ that satisfies the groundtruth constraint α^*.

In summary, given a LC g learned in the symbolic rule search step, we first add learnable weights to g. We then jointly optimize the LC weights and the neural network's parameters by maximizing Eq. (4).

6 Experiments

In this section, we evaluate the proposed NTOC model on the multi-digit addition task and the Tower of Hanoi action prediction task.[5] In both tasks, we evaluate the model's ability to (i) simultaneously training neural networks and extracting task semantics, and (ii) learn invariant features $\boldsymbol{z}$ that generalize well across various test domains.

Baselines We compare the proposed model against two NN models, LSTM (Sak, Senior, and Beaufays 2014) and DNC (Graves et al. 2016), and a neural-symbolic model DeepProbLog (Manhaeve et al. 2018). Both NN models are selected for their ability to handle sequential prediction tasks. Although not designed for joint training tasks, DeepProbLog is considered as the SoTA approach in neural-symbolic tasks such as injecting knowledge to NNs and induce symbolic rules. Note that in the experiments, DeepProbLog is additionally provided with the groundtruth logic rules. Hence, it is considered as a loose performance upper bound of other approaches. Appx. B describes the implementation details of the considered baselines. Additionally, since some neural-symbolic models such as OptNet (Amos and Kolter 2017) do not fit both tasks, we run additional tasks in Appx. C to compare against these models.

Multi-digit addition Models are only informed that the task has a recursive structure; except DeepProbLog, no considered model is provided with the multi-digit addition rules. Specifically, the recursion prior is informed by first feeding the two images corresponding to the ones place of the input numbers to predict the output number's ones place, followed by feeding the images for the tens place, hundreds place, etc. For NTOC, we use auxiliary variables, denoted $\mathbf{Z}_{\text{in}}$ and $\mathbf{Z}_{\text{out}}$, to carry memory across time steps. Specifically, $\boldsymbol{z}_{\text{in}}$ and $\boldsymbol{z}_{\text{out}}$ are the input auxiliary variables from the previous time step and fed to the next time step, respectively. For all methods, 100 MNIST images are given for pretraining.

We challenge all models by training the models on numbers of length 3 and test them with numbers of length 5-20. As shown in Tab. 1, NTOC achieves significantly better than non-symbolic models (i.e., LSTM and DNC). Moreover, NTOC is on-par with DeepProbLog, where the groundtruth multi-digit addition rules are explicitly given. To trace the source of error, we additionally evaluate all models with numbers consist of MNIST images from the training set. This bridges the generalization gap towards unseen digit images and directly challenges the model's ability to learn generalizable addition rules. Results show that NTOC still outperforms all baselines with no provided groundtruth task semantics, which shows that the NTOC is able to learn generalizable symbolic rules. Moreover, we find that the main error source of NTOC comes from the perception part, i.e., the misclassifications made by the NN.

Another benefit of NTOC is its speed. Existing neural-symbolic approaches such as DeepProbLog and ILP (Evans and Grefenstette 2018) require a compilation step to convert their symbolic models to low-level computation modules, which takes hours or days even for very simple tasks. In contrast, since LCs are represented as computation

[5] Our code can be found at https://github.com/sbx126/NToC.

Table 1: OOD performance (± std over 10 runs) on the multi-digit addition and Tower of Hanoi tasks. NTOC is compared against two NN-based models (LSTM and DNC) and a neural-symbolic approach DeepProbLog). † Note that DeepProbLog is additionally provided with groundtruth logical rules, and is considered as the performance upper-bound of other approaches. Multi-digit addition: different configurations for digit number generalization and sequence length generalization. Tower of Hanoi: a suit of different configurations for disks number generalization and movement-steps length generalization. Numbers are sequence accuracies, i.e., the fraction of correctly predicted sequences.

Model	Multi-digit addition [test seq length + train/test img]						Tower of Hanoi		
	5 w/ test	10 w/ test	20 w/ test	5 w/ train	10 w/ train	20 w/ train	Task #1	Task #2	Task #3
DeepProbLog†	89.68±1.34	80.87±2.32	timeout	95.16±0.81	90.59±1.41	timeout	89.01±1.36	96.69±0.39	88.37±1.29
LSTM	81.61±2.54	61.13±4.71	35.06±5.69	91.94±1.66	80.31±3.15	60.68±5.94	79.29±2.78	91.09±1.41	73.61±3.43
DNC	87.64±0.37	74.78±1.00	57.09±1.39	93.10±1.45	85.52±2.46	73.28±4.13	66.71±0.54	94.57±1.38	65.35±1.77
NTOC(ours)	**91.20**±0.73	**82.93**±1.95	**68.60**±3.14	**97.90**±0.29	**95.79**±0.53	**91.92**±0.99	**86.07**±0.28	**94.82**±0.19	**84.32**±0.41

graph, NTOC is free from compilation and can handle problems with many variables.

Tower of Hanoi optimal action prediction The goal of this task is to predict the optimal next move given a game state. As determined by the nature of the game, algorithms need to select particular symbols to achieve good generalization performance. We would like to reiterate that compared to the previous task, only the perception model and primitive task are different here. As illustrated by Fig. 1(right), despite the different visual appearances of Hanoi states with different number of disks, an invariant policy can be found if the NN learns to always compare the size of the top disk on each pillar.[6] This experiment challenges the models to automatically discover such an invariant policy without direct supervision. In the pretraining step, we give 500 samples to teach the model to compare between disk sizes. For every sample, each pillar contains at most one disk, so it is impossible to learn the invariant feature solely from pretraining. Furthermore, to ensure we are evaluating different method' ability to discover invariant features and policies, we only provide 1,000 training samples, while testing them on 10,000 samples. With this huge size difference between training and testing set, no considered method can resort to the shortcut of trying to remember Hanoi state-action pairs.

We use three types of OOD settings: #1 (# disk generalization): game states with 5 disks are used for training while test samples contain 7 or 9 disks; #2 (sequence length generalization): models are trained on sequences of length 3 and tested on sequences of length 5; #3 (both): combination of #1 and #2. Setting #2 is the simplest in terms of generalization, as spurious features could still help. In contrast, settings #1 and #3, which require the the model to generalize to more game states with more disks, are more appropriate testing beds to test method's ability to discover invariance. As shown in Tab. 1, our proposed method NTOC out-performs all baselines in all three settings. The lead is especially significant on settings #1 and #3.

Interpretability An additional benefit of using explicit logical models is interpretability. We examine the learned circuit on Hanoi, and find that the optimal actions are only related to the size relationship of top disks, and the last action. By summarizing the learned rules, we interpret the action policy as the following. Alternate actions between the smallest disk and a non-smallest disk. When moving the smallest disk, always move it to the left. If the smallest disk is on the first pillar, move it to the third one. When moving the non-smallest disk, take the only valid action. Details of the circuit and its relation to this policy is shown in Appx. D.

[6]We use MNIST images in place of disks; disk sizes are represented by different MNIST digits.

7 Related Work and Conclusion

The integration of perception and symbolic reasoning has been considered as a key challenge in machine learning (Mattei and Walsh 2013; De Raedt et al. 2021; Garcez et al. 2019). A well-studied task in neural-symbolic computation is to inject prior task semantics, often represented as logical rules, into neural networks. E.g., Manhaeve et al. (2018) and Xu et al. (2018) use techniques from knowledge representation to convert symbolic knowledge into differentiable representations, and use these symbolic models to refine NNs.

When such priors do not exist, neural-symbolic algorithms seek to learn the knowledge either explicitly or implicitly. Specifically, implicit neural-symbolic models use differentiable layers with logic-related inductive biases to infer symbolic rules. For instance, Wang et al. (2019) introduces a differentiable SAT solver to find the logical rules; Dong et al. (2018) designs a neural network layer that mimics the forward chaining procedure of first-order logic. However, since the reasoning process of these models are prone to noise, they tend to suffer in OOD generalization. In contrast, some other approaches use informed search strategies to find an informative set of rules that best explain the symbols predicted by the neural network (Dai and Muggleton 2021; Yang, Ishay, and Lee 2020), though they often require background knowledge written in the form of logical programs. ILP (Evans and Grefenstette 2018) assigns learnable weights to a set of given rules, and uses a differentiable forward chaining algorithm to learn the rules. However, ILP requires accurate candidate rules and sufficient negative samples, which are often hard to specify in practice. Another line of efforts show that non-symbolic models are capable of solving certain neural-symbolic tasks (Cognolato and Testolin 2022).

Different from all these existing approaches, our proposed algorithm NTOC learns explicit symbolic rules without assuming the existence of background knowledge or candidate rules. We show that the proposed model exhibits good out-of-distribution generalization on two neural-symbolic tasks.

Acknowledgements

This work was funded in part by a grant from National Key R&D Program of China (2021ZD0150200), the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005, NSF grants #IIS-1943641, #IIS1956441, #CCF-1837129, Samsung, CISCO, a Sloan Fellowship, and a UCLA Samueli Fellowship. We thank Honghua Zhang for providing helpful feedback on an initial version of the paper.

References

Amos, B.; and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 136–145. PMLR.

Blanchard, G.; Lee, G.; and Scott, C. 2011. Generalizing from several related classification tasks to a new unlabeled sample. *NeurIPS*, 24.

Chavira, M.; and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7): 772–799.

Choi, Y.; Dang, M.; and Van den Broeck, G. 2021. Group Fairness by Probabilistic Modeling with Latent Fair Decisions. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

Cognolato, S.; and Testolin, A. 2022. Transformers discover an elementary calculation system exploiting local attention and grid-like problem representation. *arXiv preprint arXiv:2207.02536*.

Dai, W.-Z.; and Muggleton, S. H. 2021. Abductive knowledge induction from raw data. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17: 229–264.

De Raedt, L.; Dumancic, S.; Manhaeve, R.; and Marra, G. 2021. From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In *Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI), Yokohama, Japan, Januray 7-15, 2021.*, 4943–4950. ijcai. org.

Deshmukh, A. A.; Lei, Y.; Sharma, S.; Dogan, U.; Cutler, J. W.; and Scott, C. 2019. A generalization error bound for multi-class domain generalization. *arXiv preprint arXiv:1905.10392*.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2018. Neural Logic Machines. In *International Conference on Learning Representations*.

Evans, R.; and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61: 1–64.

Garcez, A.; Gori, M.; Lamb, L.; Serafini, L.; Spranger, M.; and Tran, S. 2019. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4): 611–632.

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476.

Kimmig, A.; Broeck, G. V. d.; and De Raedt, L. 2012. Algebraic model counting. *arXiv preprint arXiv:1211.4475*.

Li, T.; and Srikumar, V. 2019. Augmenting Neural Networks with First-order Logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 292–302.

Liang, Y.; Bekker, J.; and Van den Broeck, G. 2017. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*.

Liu, A.; and Van den Broeck, G. 2021. Tractable Regularization of Probabilistic Circuits. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In *32nd Conference on Neural Information Processing Systems (NIPS 2018), Montreal, Canada, December 2-8, 2018*, 3753–3760. Neural Information Processing Systems Foundation Inc.

Mattei, N.; and Walsh, T. 2013. Preflib: A library for preferences http://www. preflib. org. In *International Conference on Algorithmic Decision Theory*, 259–270. Springer.

Sak, H.; Senior, A. W.; and Beaufays, F. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Neural Computation*.

Sang, T.; Beame, P.; and Kautz, H. 2005a. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, 475–482. AAAI Press.

Sang, T.; Beame, P.; and Kautz, H. A. 2005b. Performing Bayesian inference by weighted model counting. In *AAAI*, volume 5, 475–481.

Wang, P.-W.; Donti, P.; Wilder, B.; and Kolter, Z. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, 6545–6554. PMLR.

Xie, Y.; Xu, Z.; Kankanhalli, M. S.; Meel, K. S.; and Soh, H. 2019. Embedding symbolic knowledge into deep networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 4233–4243.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Broeck, G. 2018. A semantic loss function for deep learning with symbolic knowledge. *International conference on machine learning*.

Yang, Z.; Ishay, A.; and Lee, J. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. In *29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*.

Ye, H.; Xie, C.; Cai, T.; Li, R.; Li, Z.; and Wang, L. 2021. Towards a theoretical framework of out-of-distribution generalization. *Advances in Neural Information Processing Systems*, 34.

Supplementary Material

A Details of the Proposed Neural-Symbolic Joint Learning Algorithm

This section provides additional details on the proposed symbolic rule search algorithm (Appx. A.1) and the neural model learning algorithm (Appx. A.2).

A.1 The Symbolic Rule Search Algorithm

This section provides additional details on (i) how to select split candidates (Alg. 1, line 5) and (ii) the criterions for pruning worlds in the LC (Alg. 1, line 7). Before that, we introduce the node flow of circuits, which is an important concept repeatedly used in the rule search algorithm.

Definition 3 (Circuit flows)**.** The node flow $\mathrm{F}_n(\mathbf{p})$ of every unit n in a circuit g is defined in a top-down manner: for the base case, flow of the root node n_r is defined as $g_{n_r}(\mathbf{p})$. For every other node n, its flow is the sum of flows cumulated from all its parents, denoted $\mathsf{pa}(n)$:

$$\mathrm{F}_n(\mathbf{p}) = \sum_{m \in \mathsf{pa}(n)} \mathrm{F}_{m,n}(\mathbf{p}) \qquad \text{where } \mathrm{F}_{m,n}(\mathbf{p}) = \begin{cases} \mathrm{F}_m(\mathbf{p}) \cdot \frac{g_n(\mathbf{p}) \cdot \theta_{m,n}}{g_m(\mathbf{p})} & \text{if } m \text{ is a sum unit} \\ \mathrm{F}_m(\mathbf{p}) & \text{if } m \text{ is a product unit} \end{cases}$$

Intuitively, $\mathrm{F}_n(\mathbf{p})$ denotes the contribution of n to the final value of g (i.e., $g_{n_r}(\mathbf{p})$). That is, how much will $g_{n_r}(\mathbf{p})$ decrease if n is removed from g.

Selecting `SPLIT` candidates We assign each candidate a score and select the `SPLIT` candidate with the maximum score. Given dataset $\mathcal{D} = \{\mathbf{p}^{(i)}\}_{i=1}^{N}$, the score of each candidate $(m\, n\, X\ )$ ($(m\, n\ )$ is an edge and X is a variable) is defined as

$$\mathrm{score}(m\, n\, X\ ) := -\left(\sum_{i=1}^{N} \mathrm{F}_{m,n}(\mathbf{p}^{(i)})\right) \cdot \Big(p_X \log p_X + (1-p_X)\log(1-p_X)\Big)$$

$$\text{where } p_X = \frac{\sum_{i=1}^{N} \mathrm{F}_{m,n}(\mathbf{p}^{(i)}) \cdot \mathbf{p}_X^{(j)}}{\sum_{j=1}^{N} \mathrm{F}_{m,n}(\mathbf{p}^{(j)})}$$

The first term measures the "importance" of $(m\, n\ )$: if the edge is removed, how much will the value $\sum_{i=1}^{N} g(\mathbf{p}^{(i)})$ decrease. The second term estimates the information gain of splitting over $(m\, n\, X\ )$. Intuitively, if the value of X for every sample $\mathbf{p}^{(i)}$ that "activates" $(m\, n\ )$ (with high edge flow) is close to 0 or 1, splitting $(m\, n\, X\ )$ does not effectively separate the samples to the two generated edges.

Circuit node pruning We adopt the two following criterions to prune worlds from LC g.

- For a node n in g, if the flow of most ($\geq 1-\epsilon$, where ϵ is a hyperparameter) samples are b-times smaller than the average sample likelihood, we prune away this node. Here, ϵ and b are hyperparameters, and we fix $\epsilon = 0\ 002$ and $b = 2$ for all experiments.
- For each node n, node $\mathcal{D}_n$ as the collection of samples $\mathbf{p}$ such that $\mathrm{F}_n(\mathbf{p})$ is greater than $1\ \ b$ times the average sample likelihood. For any variable X, if most ($\geq 1-\epsilon$) samples in $\mathcal{D}_n$ have $\mathbf{p}_X > 1-\eta$ (resp. $\mathbf{p}_X < \eta$), we prune away worlds by replacing n with the conjunction of n and X (resp. $\ \ X$).

Computational complexity As mentioned in the last paragraph of Sec. 5.1, although each step in Alg. 1 takes linear time, like most other learning tasks (e.g., finding the optimal parameters of a NN), finding the optimal LC that encodes the correct logic formula is still NP-hard in general. In our experiments, we have shown that NTOC is able to discover the logic rules for the multi-digit addition task and the Hanoi task. Although it might be hard to figure out very complex rules, NTOC is able to effectively incorporate prior knowledge about the logic rules by using task-specific LC structures to improve search efficiency. We will leave this to future work.

A.2 The Neural Model Learning Algorithm

This section provides additional details to the neural model learning algorithm. Specifically, we use expected flows (Choi, Dang, and Van den Broeck 2021) to compute the target for EM updates, and use a step size of $0\ 02$.

B Implementation Details

This section describes implementation details of the baseline models. Note that for all method, structure of the perception network (i.e., the model that takes $\boldsymbol{x}$ as input and predicts $\boldsymbol{z}$) is the same as used by NTOC , and only the symbolic model (i.e., the model that takes $\boldsymbol{z}$ as input and predicts $\boldsymbol{y}$) is changed.

LSTM We use the standard LSTM layer in PyTorch with hidden dimension 40. The output of the LSTM is then connected to a 2-layer MLP classifier to predict $\boldsymbol{y}$. For the Tower of Hanoi task, we initialize the hidden state of the LSTM using embeddings (i.e., `nn.Embedding`) obtained with the action that leads to the initial game state.

Table 2: ***Single-digit addition results.*** Test set accuracy on balanced/imbalanced training datasets with various sample sizes (2000-20000). MLP refers to a 3-layer multi-layer perceptron; DeepPL stands for DeepProbLog (Manhaeve et al. 2018); OPT denotes OPTNET (Amos and Kolter 2017); NToC stands for our proposed method NTOC. Different from the three other models, DeepProbLog additionally takes the groundtruth symbolic knowledge as input. Therefore, its accuracies are considered as the upper-bound for other approaches, including NTOC.

# samples	Balanced data				Imbalanced data			
	MLP	OPT	DeepPL	NToC	MLP	OPT	DeepPL	NToC
2000	91.00±0.59	80.10±5.25	93.21±0.38	92.26±0.51	90.10±0.82	73.49±5.69	91.89±0.73	92.13±0.74
5000	94.59±0.40	90.20±1.25	95.30±0.29	94.98±0.27	94.01±0.37	77.58±5.53	94.33±0.28	94.73±0.39
10000	95.61±0.14	92.92±0.29	97.01±0.14	96.17±0.37	95.43±0.12	86.29±1.26	95.70±0.23	96.08±0.21
20000	96.31±0.13	95.09±0.29	97.00±0.01	96.75±0.34	96.11±0.18	92.65±2.51	96.79±0.02	96.63±0.14

DNC We use the code repository from https://github.com/RobertCsordas/dnc. Specifically, we use the LSTM controller whose hyperparameters are taken from the LSTM model described above. We found most other hyperparameters (e.g., number of read/write heads) insensitive to the overall performance. Therefore, other hyperparameters are set to the default value as suggested in the repository. Additionally, we also initialize the memory of the DNC using embeddings of the last action for the Tower of Hanoi task.

Deepproblog We use the author-provided code from https://github.com/ML-KULeuven/deepproblog.

OPTNET We perform all experiments with the following official code https://github.com/locuslab/optnet. We set `nHidden=40` and all other hyperparameters are kept as default.

NTOC For all experiments, NToC uses the Adam optimizer with learning rate 1e-3. For the multi-digit addition task, we used a simple CNN with two Conv-ReLU-MaxPool blocks (hidden size 32 and 64, respectively), together with an FC layer to map the input to the 10-dimensional Z. For the Hanoi task, all models (including baselines) use a CNN+attention model: first a three-layer CNN is used to process visual features (hidden size 32, 64, 64), then a single multi-headed Attention layer is used to enable to network to focus on specific parts of the input.

Hardware specifications All experiments are done on a workstation with 32 CPUs, 64 GB memory, and one 3090 GPU.

C Additional Experiments

Due to the sequential nature of the tasks in Sec. 6, some existing neural-symbolic approaches cannot be applied directly. Therefore, we create a single-digit version of the MNIST addition task described in the paper. Specifically, the input $\boldsymbol{x}$ consists of two MNIST images, and the label is their sum. We challenge the algorithms from two aspects — their sample complexity and robustness to unseen digit combinations. Specifically, to examine the models' sample complexity, we created four training sets, with 2,000, 5,000, 10,000, and 20,000 samples, respectively. To examine the models' ability to generalize to unseen digit combinations, we additionally create four imbalanced datasets, where for all samples, the digit of the first MNIST image is larger than or equal to that of the second image. As shown in Tab. 2, in all settings, NTOC out-performs MLP and OPT, and performs on-par compared to DeepProbLog. This suggests that NTOC is capable of performing neural-symbolic joint learning and generalize to rarely-observed digit combinations.

To further challenge the ability of NTOC to learn generalizable logic rules with minimum pretraining signal, we design the following variant of the Hanoi task. Specifically, during pretraining, we only provide the model with two MNIST digits arranged side-by-side on the input image, and train the neural network to predict whether the left digit is larger than the right digit. Since we need at least six symbols to accomplish the Hanoi task, we replace the final classification layer of the NN with a randomly initialized fully-connected layer with 6 outputs. In this setting, the pretraining task only teaches the NN to compare two digits, and no information about which digits should be compared is implied in the pretraining step. Similar to the experiments in the main paper, after pretraining, we train all models using game states with 2 disks and test them with game states containing 3 disks. Additionally, models are trained on sequences of length 2 and tested on sequences of length 5. As shown in the following table, NTOC maintains its superior performance even under this more challenging setting.

Model	NTOC(ours)	LSTM	DNC
Test accuracy	**77.05**±2.60	62.88±2.14	61.29±1.02

D Learned Circuit Representations of Task Semantics

In this section, we visualize and briefly describe task semantics learned by our circuit representation. We use the Tower of Hanoi problem as an example since the learned circuit has moderate size.

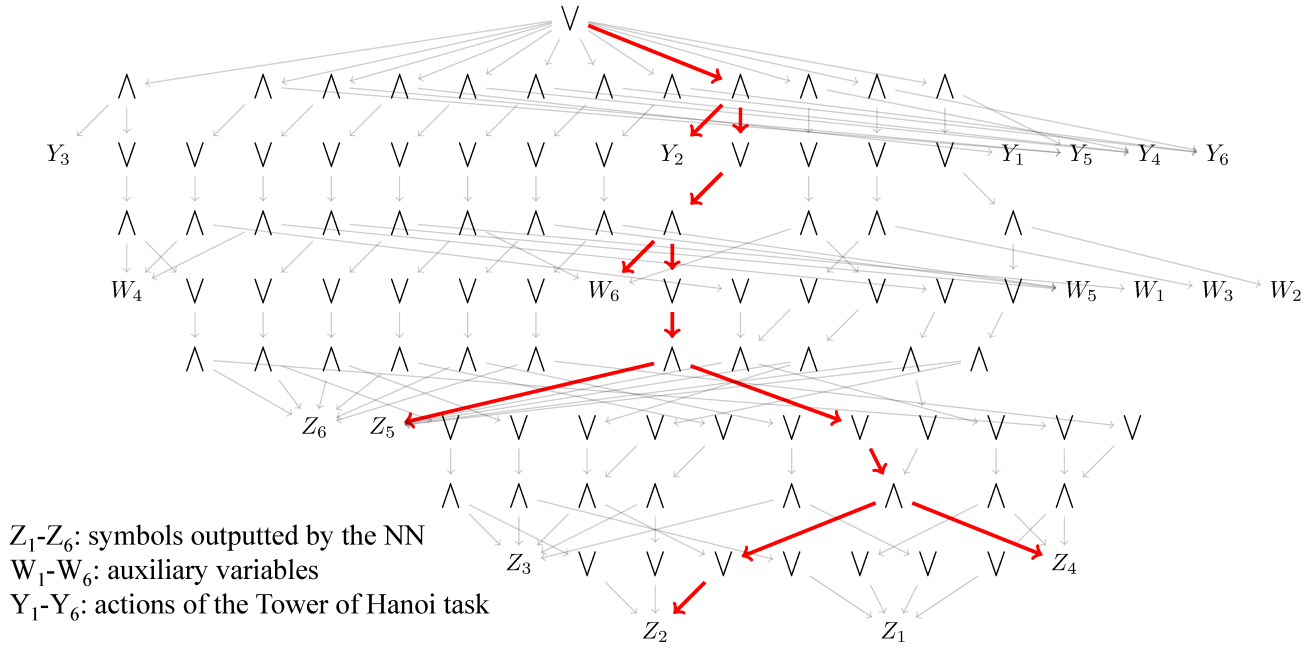

Figure 5: Learned LC representation of the task semantics of the Tower of Hanoi problem.

The learned circuit is shown in Fig. 5, defines product nodes, denotes or nodes, and variables represents input nodes corresponding to the labeled variables. Specifically, variables Z_1 to Z_6 are the symbols outputted by the neural network f_φ; W_1 to W_6 denote the auxiliary variables; Y_1 to Y_6 represents the 6 possible actions.

Suppose the NN f_φ predicts $\Pr(A) = 1\,(A \quad Z_2\ Z_4\ Z_5\ W_6\ )$ and $\Pr(A) = 0$ for all other variables A, we can *query* the most likely output variable, which is $Y_2 = \texttt{true}$ (i.e., move the top disk from the second pillar to the third one) in this case.