Learning Symmetric Rules with SATNet

Anonymous Author(s)
Affiliation
Address
email

Abstract

SATNet is a differentiable constraint solver with a custom backpropagation algorithm, which can be used as a layer in a deep-learning system. It is a promising proposal for bridging deep learning and logical reasoning. In fact, SATNet has been successfully applied to learn, among others, the rules of a complex logical puzzle, such as Sudoku, just from input and output pairs where inputs are given as images. In this paper, we show how to improve the learning of SATNet by exploiting symmetries in the target rules of a given but unknown logical puzzle or more generally a logical formula. We present SymSATNet, a variant of SATNet that translates the given symmetries of the target rules to a condition on the parameters of SATNet and requires that the parameters should have a particular parametric form that guarantees the condition. The requirement dramatically reduces the number of parameters to learn for the rules with enough symmetries, and makes the parameter learning of SymSATNet much easier than that of SATNet. We also describe a technique for automatically discovering symmetries of the target rules from examples. Our experiments with Sudoku and Rubik's cube show the substantial improvement of SymSATNet over the baseline SATNet.

1 Introduction

Bringing the ability of reasoning to the deep-learning systems has been the aim of a large amount of recent research efforts [Yang et al., 2017, Evans and Grefenstette, 2018, Cingillioglu and Russo, 2019, Wang et al., 2019, Topan et al., 2021]. One notable outcome of these endeavours is SATNet [Wang et al., 2019], a differentiable constraint solver with an efficient custom backpropagation algorithm. SATNet can be used as a component of a deep-learning system and make the system capable of learning and reasoning about sophisticated logical rules. Its potential has been demonstrated successfully with the tasks of learning the rules of complex logical puzzles, such as Sudoku, just from input-output examples where the inputs are given as images.

We show how to improve the rule (or constraint) learning of SATNet, when the target rules have symmetries. By symmetries, we mean transformations of candidate solutions of those rules, and by rules having symmetries, we mean that the solution set of those rules is closed under the transformations of those symmetries. For example, in Sudoku, if a completed 9×9 Sudoku board is a solution, permuting the numbers 1 to 9 in the board, the first three rows, or the last three columns always gives rise to another solution. Thus, these transformations are symmetries of Sudoku.

Our improvement is a variant of SATNet, called SymSATNet, which abbreviates symmetry-aware SATNet. SymSATNet assumes that some symmetries of the target rules are given a priori although the rules themselves are unknown. It then translates these symmetries into a condition on the parameter matrix $C \in \mathbb{R}^{n \times n}$ of SATNet (or our minor generalisation), and requires that the parameters have a particular parametric form that guarantees the condition. Concretely, the translated condition says that the matrix C regarded as a linear map should be equivariant with respect to the group G determined by the given symmetries, and the requirement is that C should be a linear combination of elements in

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

a basis for the space of G-equivariant symmetric matrices. The coefficients of this linear combination are the parameters of our SymSATNet, and their number is often substantially smaller than that of the parameters of SATNet.[1] For Sudoku, the former is 18, while the latter is 729^2 or $k \cdot 729$ for some $k \in \mathbb{N}$ at best. The reduced number of parameters implies that SymSATNet has to tackle an easier learning problem than SATNet, and has a potential to learn faster and generalise better than SATNet.

Who provides symmetries for SymSATNet? The default answer is domain experts, but we show that a better alternative is possible for some applications. We present an algorithm that automatically discovers symmetries. Our algorithm is based on our empirical observation that for Sudoku and a problem related to Rubik's cube, symmetries emerge in the parameter matrix C of SATNet in the early phase of training, as clusters of similar entries in C. Our algorithm takes a snapshot of the matrix C at some training epoch of SATNet, and searches for a group G such that (i) whenever two entries of C are required to have similar values by the G-equivariance condition, they indeed have such values, and (ii) the number of SymSATNet parameters under G is minimised.

We empirically evaluate SymSATNet and our symmetry-discovering algorithm with Sudoku and a problem related to Rubik's cube. For both problems, our algorithm discovered nontrivial symmetries, and SymSATNet with manually specified or automatically found symmetries outperformed the baseline SATNet in learning the rules, in terms of both efficiency and generalisation.

Related work There have been multiple studies on discovering symmetries present in conjunctive normal form (CNF) formulas in order to reduce the search space of satisfiability (SAT) solvers. Crawford [1992] proved that the symmetry-detection problem is equivalent to the graph isomorphism problem, and showed how to reduce the complexity of pigeonhole problems using symmetries. Crawford et al. [1996] proposed symmetry-breaking predicates (SBPs), and Aloul et al. [2003, 2006] developed SBPs with more efficient constructions. For automatic symmetry detection, Darga et al. [2004] presented a method that improves the partition refinement procedure introduced by McKay [1976, 1981], and Darga et al. [2008] proposed an algorithm that achieve efficiency by exploiting the sparsity of symmetries. In contrast to these global and static methods, Benhamou et al. [2010], Devriendt et al. [2017] handled local symmetries that dynamically appear during search. In the context of deep learning, Basu et al. [2021], Dehmamy et al. [2021] described algorithms that find and exploit symmetries via group decompositions and Lie algebra convolutions. Our work is also related to the studies on learning logical rules from examples using gradients. Yang et al. [2017] proposed neural logic programming, an end-to-end differentiable system which learns first-order logical rules, Evans and Grefenstette [2018] proposed a differentiable inductive logic programming system which is robust to noise of training data, and Cingillioglu and Russo [2019] introduced an RNN-based model to learn logical reasoning tasks end-to-end. Wang et al. [2018, 2019] presented SATNet using the mixing method, and Topan et al. [2021] further improved SATNet by solving the symbol grounding problem, a key challenge of SATNet. Our work extends these lines of work by proposing how to discover and exploit symmetries when learning logical rules with SATNet.

2 Background

We review SATNet, the formalisation of symmetries using groups, and equivariant maps. For a natural number n, let $[n] = \{1, \ldots, n\}$, and for a matrix M, let $M_{i,j}$ be the (i, j)-th entry of M.

2.1 SATNet

A good starting point for learning about SATNet is to look at its origin, the mixing method [Wang et al., 2018], which is an efficient algorithm for solving semidefinite programming problems with diagonal constraints. Let $n, k \in \mathbb{N}$ and C be a real-valued symmetric matrix in $\mathbb{R}^{n \times n}$. The mixing method aims at solving the following optimisation problem:

$$\operatorname*{argmin}_{V \in \mathbb{R}^{k \times n}} \langle C, V^T V \rangle \quad \text{subject to } \|v_i\| = 1 \text{ for } i \in [n] \tag{1}$$

where v_i is the i-th column of the matrix V, and $\|v_i\|$ is L_2 norm of v_i. The mixing method solves (1) by coordinate descent, where each column v_i of V is repeatedly updated as follows:

[1] SATNet assumes that C is of the form $S^T S$ for some $S \in \mathbb{R}^{m \times n}$ for $m < n$, so that the number of parameters is $mn < n^2$. But often it is still substantially larger than the number of parameters of SymSATNet.

$g_i \leftarrow \sum_{j\in[n], j\neq i} C_{i,j} v_j$ and $v_i \leftarrow -\frac{g_i}{\|g_i\|}$. This always finds a fixed point of the equations. In fact, it is shown that almost surely this fixed point attains a global optimum of the optimisation problem.

An example of the above optimisation problem most relevant to us is a continuous relaxation of MAXSAT. MAXSAT is a problem of finding truth assignments to n boolean variables $b_1, \ldots, b_n$. It assumes m clauses of those variables, $F_1, \ldots, F_m$, where F_ℓ is the disjunction of some variables with or without negation: $F_\ell = b_{i_1} \vee \ldots \vee b_{i_p} \vee \neg b_{i_{p+1}} \vee \ldots \neg b_{i_{p+q}}$. Then, MAXSAT asks for a truth assignment on the variables that maximises the number of true clauses F_i under the assignment. The rules of many problems, including Sudoku, can be expressed as an instance of MAXSAT.

To apply the mixing method to MAXSAT, we introduce relaxed vectors $v_1, \ldots, v_n \in \mathbb{R}^k$ that encode the boolean variables, and construct the matrix $S \in \mathbb{R}^{m\times n}$ that encodes the m clauses of MAXSAT: the (ℓ, j)-th entry of S has 1 if F_ℓ contains b_j; and -1 if F_ℓ includes $\neg b_j$; and 0 if neither of these cases holds. Then, the problem in (1) is formed with $C = S^T S$, and solved by the mixing method. Our presentation is simplified, and the original version can be found in [Wang et al., 2018, 2019].

SATNet is a variant of the mixing method where some of the columns of V are fixed and the optimisation is over the rest of the columns.[2] Concretely, it assumes that the column indices in $[n]$ are split into two disjoint sets, $\mathcal{I}$ and $\mathcal{O}$ (i.e., $\mathcal{I} \cup \mathcal{O} = [n]$ and $\mathcal{I} \cap \mathcal{O} = \emptyset$). The inputs of SATNet are the columns v_i of V with $i \in \mathcal{I}$, and the outputs are the rest of the columns (i.e., the v_o's with $o \in \mathcal{O}$). The symmetric matrix C is the parameter of SATNet. Given the input vectors, SATNet repeatedly executes the coordinate descent updates on each output column, until it converges.

One important feature of SATNet is that it has a custom algorithm for backpropagation. Let $V_\mathcal{I}$ be the matrix of the input columns to SATNet, and $V_\mathcal{O}$ be that of the output columns computed by SATNet on the input $V_\mathcal{I}$ under the parameter C. Assume that l is a loss of the output $V_\mathcal{O}$. In this context, SATNet provides formulas and algorithms for computing the derivatives $\partial l/\partial V_\mathcal{I}$ and $\partial l/\partial C$.

We recall the formulas for the derivatives. Let $o_1 < o_2 < \ldots < o_{|\mathcal{O}|}$ be the sorting of the indices in $\mathcal{O}$. Assume that SATNet was run until convergence, so that the output columns in $V_\mathcal{O}$ are the fixed point of the coordinate descent updates: for all $o \in \mathcal{O}$, $g_o = \sum_{j\in[n], j\neq o} C_{o,j} v_j$ and $v_o = -\frac{g_o}{\|g_o\|}$. The formulas for $\partial l/\partial V_\mathcal{I}$ and $\partial l/\partial C$ at $(V_\mathcal{I}, V_\mathcal{O}, C)$ are defined in terms of the next quantities:

$$C', D' \in \mathbb{R}^{|\mathcal{O}|\times|\mathcal{O}|}, \qquad P \in \mathbb{R}^{|\mathcal{O}|k\times|\mathcal{O}|k}, \qquad U \in \mathbb{R}^{|\mathcal{O}|k\times 1}, \qquad W \in \mathbb{R}^{|\mathcal{O}|k\times n^2}.$$

They have the following definitions: for $i, j \in [|\mathcal{O}|]$, $p, q \in [k]$, and $r, s \in [n]$,

$$(C')_{i,j} = \begin{cases} 0 & \text{if } i = j \\ C_{o_i,o_j} & \text{if } i \neq j, \end{cases} \qquad U = (P((D' + C') \otimes I_k))^\dagger \left(\frac{\partial l}{\partial \operatorname{vec}(V_\mathcal{O})}\right)^T,$$

$$(D')_{i,j} = \begin{cases} \|g_{o_i}\| & \text{if } i = j \\ 0 & \text{if } i \neq j, \end{cases}$$

$$P_{\substack{(i-1)k+p,\\(j-1)k+q}} = \begin{cases} (I_k - v_{o_i} v_{o_i}^T)_{p,q} & \text{if } i = j \\ 0 & \text{if } i \neq j, \end{cases} \qquad W_{\substack{(i-1)k+p,\\(r-1)n+s}} = \begin{cases} 0 & \text{if } r = o_i \text{ and } s = o_i \\ V_{p,s} & \text{if } r = o_i \text{ and } s \neq o_i \\ V_{p,s} & \text{if } r \neq o_i \text{ and } s = o_i \\ 0 & \text{if } r \neq o_i \text{ and } s \neq o_i. \end{cases}$$

Here $\otimes$ is the Kronecker product, $_^\dagger$ is Moore-Penrose inverse (also known as pseudo inverse), and $\operatorname{vec}(V_\mathcal{O})$ is the vector obtained by stacking the columns of $V_\mathcal{O}$. Let $C_{\mathcal{O},\mathcal{I}} \in \mathbb{R}^{|\mathcal{O}|\times|\mathcal{I}|}$ be obtained by restricting C to the indices (o, i) with $o \in \mathcal{O}$ and $i \in \mathcal{I}$. Then,

$$\partial l/\partial \operatorname{vec}(C) = -U^T W, \qquad \partial l/\partial \operatorname{vec}(V_\mathcal{I}) = -U^T (C_{\mathcal{O},\mathcal{I}} \otimes I_k). \tag{2}$$

SATNet computes the above derivative formulas efficiently by iterative algorithms.

2.2 Symmetries and equivariant maps

By symmetries on a set $\mathcal{X}$, we mean a group G that acts on $\mathcal{X}$. The acting here refers to a function $_\cdot_$ from $G \times \mathcal{X}$ to $\mathcal{X}$, called *group action*, such that (i) $e \cdot x = x$ for the unit $e \in G$ and any $x \in \mathcal{X}$, and (ii) $(g \circ g') \cdot x = g \cdot (g' \cdot x)$ for all $g, g' \in G$ and $x \in \mathcal{X}$, where $_\circ_$ is the group operator of G.

[2] The original SATNet assumes that C has the form $S^T S$ for some $m \times n$ matrix S. We drop this assumption and adjust the forward and backward computations of SATNet accordingly. The main steps of derivations of the formulas for the forward and backward computations are from the work on SATNet [Wang et al., 2019].

We use symmetries generated from a collection of permutations on a finite set. The set $\mathcal{X}$ in our case is $\mathbb{R}^{k\times n}$, the space of the matrix V in (1), and G is a subgroup of the group $\mathcal{S}_n$ of all permutations on $[n]$. The group action $g \cdot V$ is then defined by permuting the columns of V by g: for all $i, j \in [n]$, $(g\cdot V)_{i,j} = V_{i,g^{-1}(j)}$. This group action can be expressed compactly with the $n\times n$ permutation matrix $P_{g^{-1}}$ where $(P_{g^{-1}})_{i,j} = \mathbb{1}_{\{i=g^{-1}(j)\}}$ and $g \cdot V = VP_{g^{-1}}$ for the indicator function $\mathbb{1}$. Throughout the paper, we often equate each element g of G with its permutation matrix P_g, and view G itself as the group of permutation matrices P_g for $g \in G$ with the standard matrix multiplication.

One important reason for considering symmetries is to study maps that preserve these symmetries, called equivariant maps. Let G be a group that acts on sets $\mathcal{X}$ and $\mathcal{Y}$.

Definition 2.1. A function $f : \mathcal{X} \to \mathcal{Y}$ is *G-equivariant* or *equivariant* if $f(g \cdot x) = g \cdot (f(x))$ for all $g \in G$ and $x \in \mathcal{X}$. It is *G-invariant* or *invariant* if $f(g \cdot x) = f(x)$ for all $g \in G$ and $x \in \mathcal{X}$.

The forms of equivariant maps have been studied extensively in the work on equivariant neural networks and group representation theory [Cohen and Welling, 2016, Maron et al., 2019, Wang et al., 2020]. In particular, when f is linear, various representation theorems for different G's describe the matrix form of f. We use permutation groups defined inductively by the following three operations.

Definition 2.2. Let G and H be permutation groups on $[p]$ and $[q]$, with each group element viewed as a $p \times p$ or $q \times q$ permutation matrix. The *direct sum* $G \oplus H$, the *direct product* $G \otimes H$, and the *wreath product* $H \wr G$ are the following groups of $(p + q) \times (p + q)$ or $pq \times pq$ permutation matrices with matrix multiplication as their composition: $G \oplus H = \{g \oplus h : g \in G,\, h \in H\}$; $G \otimes H = \{g \otimes h : g \in G,\, h \in H\}$; $H \wr G = \{\mathrm{wr}(\vec{h}, g) : g \in G, \vec{h} \in H^p\}$. Here $g \oplus h$ is the block diagonal matrix with $p\times p$ matrix g as its upper-left corner and $q\times q$ matrix h as the lower-right corner, $g \otimes h$ is the Kronecker product of two matrices g and h, and $\mathrm{wr}(\vec{h}, g)$ is the $pq \times pq$ permutation matrix defined by $\mathrm{wr}(\vec{h}, g)_{(i-1)q+j,(i'-1)q+j'} = \mathbb{1}_{\{g_{i,i'}=(h_i)_{j,j'}=1\}}$ for all $i, i' \in [p]$ and $j, j' \in [q]$.

The next theorem specifies the representation of G-equivariant linear maps for an inductively-constructed G, by describing a basis of the vector space of those linear maps. The theorem can be proved by slightly modifying the argument in [Wang et al., 2020]. For a permutation group G on $[m]$, let $\mathcal{E}(G) = \{M \in \mathbb{R}^{m\times m} : gM = Mg,\, \forall g \in G\}$, the vector space of G-equivariant linear maps, where each $g \in G$ is regarded as a permutation matrix and gM is the multiplication of matrices g and M (*i.e.*, the composition of two linear maps g and M). See Appendix B for the proof of Theorem 2.3.

Theorem 2.3. *Let G, H be permutation groups on $[p]$ and $[q]$, and $\mathcal{B}(G), \mathcal{B}(H)$ be some bases of $\mathcal{E}(G)$ and $\mathcal{E}(H)$, respectively. Then, the following sets form bases for $G \oplus H$, $G \otimes H$, and $H \wr G$:*

$$\mathcal{B}(G \oplus H) = \{A \oplus \mathbf{0}_q, \mathbf{0}_p \oplus B : A \in \mathcal{B}(G), B \in \mathcal{B}(H)\} \cup \{\mathbf{1}_{O\times(p+O')}, \mathbf{1}_{(p+O')\times O} : O \in \mathcal{O}(G), O' \in \mathcal{O}(H)\};$$

$$\mathcal{B}(G \otimes H) = \{A \otimes B : A \in \mathcal{B}(G), B \in \mathcal{B}(H)\};\ \mathcal{B}(H \wr G) = \{A \otimes \mathbf{1}_q, I_p \otimes B : A \in \mathcal{B}(G), B \in \mathcal{B}(H)\}.$$

Here I_m, $\mathbf{0}_m$, and $\mathbf{1}_m$ are identity, everywhere-zero, and everywhere-one matrices in $\mathbb{R}^{m\times m}$. $\mathbf{1}_{R\times S}$ for $R, S \subseteq [p + q]$ is the matrix in $\mathbb{R}^{(p+q)\times(p+q)}$ defined by $(\mathbf{1}_{R\times S})_{i,j} = \mathbb{1}_{\{i\in R,\, j\in S\}}$. Also, $\mathcal{O}(G) = \{\{g(i) : g \in G\} : i \in [p]\}$ (i.e., the set of G-orbits), and $p + O = \{p + i : i \in O\}$.

3 Symmetry-aware SATNet

In this section, we present SymSATNet, which abbreviates symmetry-aware SATNet. This variant is designed to operate when symmetries of a learning task are known a priori (via an algorithm or a domain expert). The proofs of the theorem and the lemma in the section are in Appendix C.

SymSATNet solves the optimisation problem of SATNet, but under the following assumptions:

1. The optimisation objective as a map on $\mathbb{R}^{k\times n}$ is invariant with respect to certain symmetries specified by a group G acting on $\mathbb{R}^{k\times n}$, the space of matrices V.
2. The group G and its action are of the type in Section 2.2. That is, each of $g \in G$ acts as a permutation on the columns of V. Continuing our convention, we denote P_g by g.

One immediate consequence of these assumptions is that for all $g \in G$ and $V \in \mathbb{R}^{k\times n}$,

$$\langle C, V^TV\rangle = \langle C, (g \cdot V)^T(g \cdot V)\rangle = \langle C, (Vg^{-1})^T(Vg^{-1})\rangle.$$

The next theorem re-phrases this property of the optimisation objective as equivariance of C:

Theorem 3.1. *Let C be a symmetric $n \times n$ matrix. Then,*

$$\langle C, V^T V \rangle = \langle C, (Vg^{-1})^T (Vg^{-1}) \rangle \tag{3}$$

for all $V \in \mathbb{R}^{k \times n}$ and $g \in G$ if C as a linear map on $\mathbb{R}^n$ is G-equivariant, that is, $Cg = gC$ for all $g \in G$. Furthermore, if $k = n$, the converse also holds.

What we discussed so far lets us incorporate the symmetries into the objective of SATNet, as we will show shortly. This leaves the diagonal constraints of SATNet. The next lemma says that those constraints require no special treatment since they are preserved by the action of any $g \in G$.

Lemma 3.2. *Let $V \in \mathbb{R}^{k \times n}$ and $g \in G$. Every column of V has the L_2-norm 1 if and only if every column of Vg^{-1} has the L_2-norm 1.*

Let $\mathcal{E}(G) = \{M \in \mathbb{R}^{n \times n} : Mg = gM, \forall g \in G\}$ be the vector space of G-equivariant matrices and $\mathcal{E}(G)_s$ be the subset of $\mathcal{E}(G)$ containing symmetric matrices. When G is a permutation group constructed inductively by direct sum, direct product, wreath product, we can generate a basis $\mathcal{B}(G)$ of $\mathcal{E}(G)$ automatically using Theorem 2.3. Then, we can convert $\mathcal{E}(G)$ to an orthogonal basis of $\mathcal{E}(G)_s$ by applying the Gram-Schmidt orthogonalisation to $\{B + B^T : B \in \mathcal{B}(G)\}$. Let $\mathcal{B}(G)_s = \{B_1, \ldots, B_d\}$ be such an automatically-generated orthogonal basis of $\mathcal{E}(G)_s$.

SymSATNet is SATNet where the matrix C in the optimisation objective has the form:

$$C = \sum_{\alpha=1}^{d} \theta_\alpha B_\alpha \tag{4}$$

for some scalar $\theta_1, \ldots, \theta_d \in \mathbb{R}$. Note that by this condition on the form of C, SymSATNet has only d parameters $\theta_1, \ldots, \theta_d$, instead of $n \times m$ for some m in the original formulation of SATNet. While learning C with symmetries, d is usually far smaller than n^2 or even n, and this reduction brings speed-up and improved generalisation. Note that (4) amounts to requiring that the optimisation problem of SATNet remain invariant with respect to G; by Theorem 3.1, it ensures that the optimisation objective is G-invariant, and by Lemma 3.2, the diagonal constraints do not change by the action of G.

The forward computation of SymSATNet is precisely that of SATNet, the repeated coordinate-wise updates until convergence, and the backward computation is the one of SATNet extended (by the chain rule) with a step backpropagating the derivatives $\partial l / \partial C$ to each $\partial l / \partial \theta_\alpha$ for $\alpha \in [d]$.[3]

We summarise SymSATNet below using the usual notation of SATNet ($\mathcal{I}$, $\mathcal{O}$, $V_\mathcal{I}$, $V_\mathcal{O}$, and V):

- The input is $V_\mathcal{I}$, the matrix of the input columns of V.
- The parameters are $(\theta_1, \ldots, \theta_d) \in \mathbb{R}^d$. They define the matrix C by (4).
- The forward computation solves the following optimisation problem using coordinate descent, and returns $V_\mathcal{O}$, the matrix of the output columns of V:

$$\operatorname*{argmin}_{V_\mathcal{O} \in \mathbb{R}^{k \times |\mathcal{O}|}} \langle C, V^T V \rangle \quad \text{subject to } \|v_o\| = 1 \text{ for } o \in \mathcal{O}.$$

- The backward computation computes $\partial l / \partial V_\mathcal{I}$ and $\partial l / \partial \theta_\alpha$ by (2) and the chain rule:

$$\partial l / \partial \theta_\alpha = (\partial l / \partial \operatorname{vec}(C))^T \operatorname{vec}(B_\alpha) = -W^T U \operatorname{vec}(B_\alpha).$$

4 Discovery of symmetries

One obstacle for using SymSATNet is that the user has to specify symmetries. We now discuss how to alleviate this issue by presenting an algorithm for discovering candidate symmetries automatically.

The goal of our algorithm, denoted by SymFind, is to find a permutation group G that captures the symmetries of an unknown learning target and is expressible by the grammar $G ::= \mathcal{I}_m \mid \mathbb{Z}_m \mid \mathcal{S}_m \mid G \oplus G \mid G \otimes G \mid G \wr G$, where $m \in \mathbb{N}$. The $\mathcal{I}_m$ denotes the trivial group containing only the identity permutation on $[m]$, and $\mathbb{Z}_m$ denotes the group of cyclic permutations on $[m]$, each of which maps $i \in [m]$ to $(i + n) \bmod m$ for some n. The $\mathcal{S}_m$ is the group of all the permutations on $[m]$. The last three cases are direct sum, direct product, and wreath product (see Definition 2.2). They describe

[3] SymSATNet is implemented based on the SATNet code [Wang et al., 2019] available under an MIT License.

three ways of decomposing a group G into smaller parts. Having such a decomposition of G brings the benefit that we can recursively and efficiently compute a basis of G-equivariant linear maps.

The design of SymFind is based on our empirical observation that a softened version of symmetries often emerges in the parameter matrix C of the original SATNet during training. Even in the early part of training, many entries of C share similar values, so that there is a group G_0 of approximate symmetries of C such that $Cg \approx gC$ for any $g \in G_0$. Furthermore, we observed that such a G_0 often consists of symmetries of the learning target. This observation suggests an algorithm that takes C as input and finds such a G_0 expressible in our grammar or its slight extension.

The input of SymFind is a matrix $M \in \mathbb{R}^{m \times m}$. As previously explained, when SymFind is called at the top level, it receives as input the parameter C of SATNet learnt by a fixed number of training steps. However, subsequent recursive calls to SymFind may have input M different from C. Then, SymFind returns a group G in our grammar and a permutation σ on $[m]$, together defining a permutation group on $[m]$:

$$\textsc{SymFind}(M) = (G, \sigma);$$
$$\mathrm{grp}(G, \sigma) = \{\sigma \circ g \circ \sigma^{-1} : g \in G\},$$

where $\circ$ is the composition of permutations. When G is decomposed into, say $G_1 \oplus G_2$, the σ specifies which indices in $[m]$ get permuted by G_1 and G_2. Once top-level SymFind returns (G, σ), we construct $\mathcal{B}(\mathrm{grp}(G, \sigma))_s$, as explained in Section 3 with a minor adjustment with σ.[4]

Algorithm 1 SymFind with a threshold $\lambda > 0$

Input: $M \in \mathbb{R}^{m \times m}$ **Output:** (G, σ)

1 **if** $\| \mathrm{prj}(\mathrm{grp}(\mathcal{S}_m, \mathrm{id}_m), M) - M \|_F \leq \lambda$ **then**
2   **return** $(\mathcal{S}_m, \mathrm{id}_m)$
3 **end if**
4 $\mathcal{A} \leftarrow \{(\mathcal{I}_m, \mathrm{id}_m)\}$
5 **if** $\| \mathrm{prj}(\mathrm{grp}(\mathbb{Z}_m, \mathrm{id}_m), M) - M \|_F \leq \lambda$ **then**
6   $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\mathbb{Z}_m, \mathrm{id}_m)\}$
7 **end if**
8 $(G', \sigma') \leftarrow \textsc{SumFind}(M)$;
9 $\mathcal{A} \leftarrow \mathcal{A} \cup \{(G', \sigma')\}$
10 **for** every divisor p of m **do**
11   $(G'', \sigma'') \leftarrow \textsc{ProdFind}(M, p)$;
12   $\mathcal{A} \leftarrow \mathcal{A} \cup \{(G'', \sigma'')\}$
13 **end for**
14 $(G, \sigma) \leftarrow \mathrm{argmin}_{(G,\sigma) \in \mathcal{A}} \dim(\mathcal{E}(\mathrm{grp}(G, \sigma)))$
return (G, σ)

Algorithm 1 describes SymFind, where id_m is the identity permutation on $[m]$, $\| \cdot \|_F$ is the Frobenius norm, $\dim(\mathcal{V})$ is the dimension of a vector space $\mathcal{V}$, and the Reynolds operator prj projects a matrix $M \in \mathbb{R}^{m \times m}$ orthogonally to the subspace of G-equivariant $m \times m$ matrices (i.e., it maps M to the closest G-equivariant matrix in L^2 distance):

$$\mathrm{prj}(G, M) = \frac{1}{|G|} \sum_{g \in G} g M g^T,$$

so that $\| \mathrm{prj}(\mathrm{grp}(G, M)) - M \|_F$ computes the distance between the marix M and the space $\mathcal{E}(G)$.

In the lines 1-3, the algorithm first checks whether $\mathcal{S}_m$ models symmetries of the input M accurately. If so, the algorithm returns $(\mathcal{S}_m, \mathrm{id}_m)$. Otherwise, it assumes that an appropriate group for M's symmetries is one of the remaining cases in the grammar, and constructs a list $\mathcal{A}$ of candidates intially containing the trivial group $(\mathcal{I}_m, \mathrm{id}_m)$. In the lines 5-7, the algorithm adds $(\mathbb{Z}_m, \mathrm{id}_m)$ to $\mathcal{A}$ if they approximate M's symmetries well. In the lines 8-9, it handles the case that M's symmetries are modelled by a direct sum $\bigoplus_i G'_i$. The algorithm calls the subroutine SumFind which returns a group G' of the form $\bigoplus_i G'_i$ and a permutation σ' if it can find such $\mathrm{grp}(G', \sigma')$ that approximates M's symmetries well. The returned group-permutation pair is added to $\mathcal{A}$. In the lines 10-13, the algorithm calls the other subroutine ProdFind for every divisor p of m. For each p, ProdFind finds a group G'' of the form $G''_1 \otimes G''_2$ or $G''_2 \wr G''_1$ and an appropriate permutation σ'' where G''_1 is a permutation group on $[p]$ and G''_2 on $[m/p]$, and such $\mathrm{grp}(G'', \sigma'')$ approximates M's symmetries well. The result of each such call is added to the list $\mathcal{A}$. Finally, in the line 14, our SymFind picks a pair (G, σ) from $\mathcal{A}$ with the strongest level of symmetries in the sense that the basis of $\mathrm{grp}(G, \sigma)$-equivariant matrices for the selected (G, σ) has the fewest elements.

The subroutine SumFind clusters entries of M as blocks since block-shaped clusters commonly arise in matrices equivariant with respect to a direct sum of groups. The other subroutine ProdFind uses the technique of Van Loan and Pitsianis [1993]; it exploits a typical pattern of Kronecker product of matrices, and detects the presence of the pattern in M by applying SVD to a reshaped version of M. Each subroutine may call SymFind recursively. See Appendix D for the details.

[4] We construct $\mathcal{B}(\mathrm{grp}(G, \sigma)) = \{\sigma B \sigma^T : B \in \mathcal{B}(G)\}$, which is an orthogonal basis for $\mathcal{E}(\mathrm{grp}(G, \sigma))$.

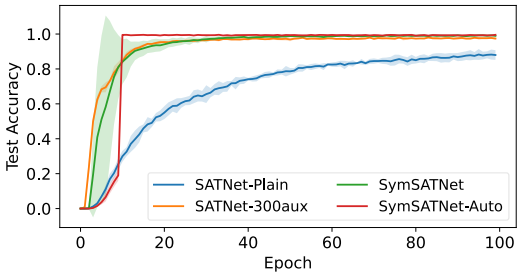
5 Experimental results

We experimentally evaluated SymSATNet and the SymFind algorithm on the tasks of learning rules of two problems, namely, Sudoku and what we call completion problem of Rubik's cube. Our evaluation used the original SATNet as a baseline, and was under both manually-specified and automatically-discovered symmetries. For SymFind, we randomly generated matrices M equivariant with respect to known symmetries and tested whether SymFind could recover those symmetries from M. We observed significant improvement of SymSATNet over SATNet in the learning tasks, and also the promising results and limitation of SymFind.

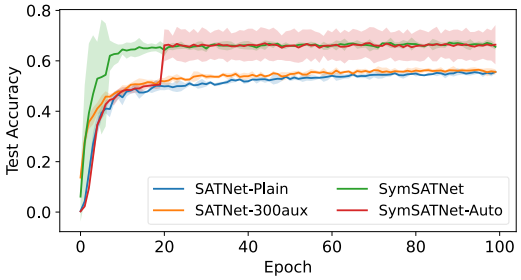
Sudoku problem In Sudoku, we are asked to fill in the empty cells of a 9×9 board such that every row, every column, and each of nine 3×3 blocks have all of 1 up to 9. Let $A \in \{0,1\}^{9\times9\times9}$ be the encoding of a full number assignment for the board where the (i,j,k)-th entry of A is 1 if the (i,j)-th cell of the board contains k. In SATNet, we flatten A to the assignment on the $n = 9^3 = 729$ boolean variables, and relax each of these variables so that it contains a vector in $\mathbb{R}^k$, resulting $V \in \mathbb{R}^{k\times n}$ in the optimisation objective of SATNet.

The rules of Sudoku have symmetries formalised by the permutation group $G = (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes \mathcal{S}_9$. Each of two $\mathcal{S}_3 \wr \mathcal{S}_3$ refers to solution-preserving permutations for rows and columns in Sudoku. The last $\mathcal{S}_9$ refers to permutations of the assigned numbers $1-9$ in each cell. See Appendix E for more information about the symmetry group for Sudoku.

To learn the rules of Sudoku using SymSATNet, we constructed a basis $\mathcal{B}(G)_s$ as explained in Section 3. It has 18 elements, which mean that SymSATNet has 18 parameters to learn.



(a) Sudoku



(b) Rubik's cube

Figure 1: Test accuracies for Sudoku and Rubik's cube throughout training epochs.

We used 9K training and 1K test examples generated by the Sudoku generator of Park [2018]. Each example is a pair $(V_\mathcal{I}, V_\mathcal{O})$ where the input $V_\mathcal{I}$ assigns the values of 39-50 cells (out of 81 cells) and the output $V_\mathcal{O}$ specifies the remaining cells. SymSATNet was compared with two types of SATNet: SATNet-Plain without any auxiliary variables, and SATNet-300aux with 300 auxiliary variables. We trained these 3 instances under the binary cross entropy loss and the Adam optimizer [Kingma and Ba, 2015], with the learning rate $\eta = 2 \times 10^{-3}$ for SATNet-Plain and SATNet-300aux as in the original work on SATNet, and $\eta = 4 \times 10^{-2}$ for SymSATNet. We measured test accuracy, which is the average number of the correctly-solved Sudoku instances in the 1K test set. Each of our experiments was run four times, and we report the average with 95% confidence interval in each run.

The results are in Figure 1 and Table 1. (SymSATNet-Auto is SymSATNet with our automatic symmetry-discovery algorithm, and will be described in the last paragraph of this section.) Our SymSATNet outperformed SATNet-Plain and SATNet-300aux. On average, its training run for 100 iterations finished 2-4× faster in terms of wall clock than those of these two alternatives. This is due to the reduced number of parameters and avoidance of certain matrix operations. See Appendix F for more information about the operations in SymSATNet. Despite

Table 1: Best test accuracies during 100 epochs and average train times (sec/epoch) for Sudoku and Rubik's cube.

Model	Sudoku		Cube	
	Acc.	Time	Acc.	Time
SATNet-Plain	88.3% ±3.1%	48.4 ±0.3	55.7% ±1.4%	1.8 ±0.01
SATNet-300aux	98.0% ±0.4%	91.0 ±1.1	56.5% ±0.6%	14.3 ±0.19
SymSATNet	99.3% ±0.5%	25.7 ±0.2	**67.2% ±0.6%**	**1.1 ±0.00**
SymSATNet-Auto	**99.6% ±0.2%**	**23.1 ±0.6**	67.1% ±6.3%	2.6 ±1.37

the speed up, the best test accuracy of SymSATNet (99.3%) on average was significantly better than SATNet-Plain (88.3%) and slightly better than SATNet-300aux (98.0%).

Completion problem of Rubik's cube The standard Rubik's cube is composed of 6 faces, each of which has 9 facelets. We considered a constraint satisfaction problem related to Rubik's cube where we are given a partial colour assignment of facelets, and then asked to complete the remaining facelets such that the resulting colour assignment is solvable; by moving the cube, we can make all facelets in each face have the same colour, and no same colours appear in two faces.

Let $A \in \{0,1\}^{6\times 9\times 6}$ be a colour assignment of Rubik's cube where the (i,j,k)-th entry has 1 if and only if the j-th facelet of the i-th face has colour k. We formulate the optimisation objective of SATNet for Rubik's cube using the relaxation of such an assignment A to $V \in \mathbb{R}^{k\times n}$ for $n = 6\times 9\times 6$.

This problem has symmetries formalised by a group $G = \mathcal{R}_{54} \otimes \mathcal{R}_6$ on $[n]$. Here $\mathcal{R}_{54}$ and $\mathcal{R}_6$ are permutation groups on $[54]$ and $[6]$, each of which captures the allowed rotations of facelets, and the rotations of the whole cube. If a colour assignment A is solvable, so is the transformation of A by any permutations in G. See Appendix E for more information about the symmetries of this problem.

We generated a basis $\mathcal{B}(G)_s$ in three steps. We first created $\mathcal{B}(\mathcal{R}_{54})$ and $\mathcal{B}(\mathcal{R}_6)$ using the technique of Finzi et al. [2021]. Next, we combined them using Theorem 2.3, and got $\mathcal{B}(G)$, which was converted to a symmetric orthogonal basis $\mathcal{B}(G)_s$ via Gram-Schmidt. The final result has 48 basis elements.

We used a dataset of 9K training and 1K test examples generated by randomly applying moves to the solution of the cube. Each example is a pair $(V_{\mathcal{I}}, V_{\mathcal{O}})$ where $V_{\mathcal{I}}$ assigns colours to facelets except for two corner facelets two edge facelets, and one center facelet, and $V_{\mathcal{O}}$ specifies the colours of those five missing facelets. In the test examples, only $V_{\mathcal{I}}$ is used. We trained SymSATNet, SATNet-Plain, and SATNet-300aux for 100 epochs, under the same configuration as in the Sudoku case.

The results appear in Figure 1 and Table 1. (SymSATNet-Auto will be described below.) On average, the 100-epoch training of SymSATNet completed faster in the wall-clock time than those of SATNet-Plain and SATNet-300aux. Also, it achieved better test accuracies (67.2%) than these alternatives (55.7% and 56.5%). Note that unlike Sudoku, the test accuracy of SATNet-300aux was only marginally better than that of SATNet-Plain, which indicates that both suffered from the overfitting issue. Note also the sharp increase in the training time of SATNet-300aux. These two indicate that adding auxiliary variables is not so effective for the completion problem for Rubik's cube, while exploiting symmetries is still useful as indicated by the performance of SymSATNet.

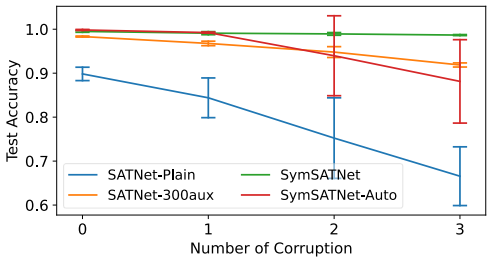
Automatic discovery of symmetries To test the effectiveness of SymFind, we tested whether SymFind could find proper symmetries in Sudoku and Rubik's cube. We trained SATNet-Plain for 10-epochs in Sudoku and 20-epochs in Rubik's cube, and applied SymFind to the parameter matrices C. For Sudoku, SymFind always recovered the full symmetries with $G = (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes (\mathcal{S}_3 \wr \mathcal{S}_3) \otimes \mathcal{S}_9$ in our 10 trials. For Rubik's cube, the group of full symmetries is $\mathrm{grp}(G,\sigma)$ for $G = ((\mathcal{S}_2 \wr \mathcal{S}_3) \oplus (\mathcal{S}_3 \wr \mathcal{S}_8) \oplus (\mathcal{S}_2 \wr \mathcal{S}_{12})) \otimes (\mathcal{S}_2 \wr \mathcal{S}_3)$. SymFind recovered all the parts except $\mathcal{S}_2 \wr \mathcal{S}_{12}$. Instead of this, the algorithm found $\mathcal{S}_{12} \otimes \mathcal{S}_2$ or $\mathcal{S}_3 \otimes \mathcal{S}_8$, or $\mathcal{S}_4 \otimes \mathcal{S}_6$ in our 10 trials. We manually observed that the entries of C in the portion of $\mathcal{S}_2 \wr \mathcal{S}_{12}$ part were difficult to be clustered, violating the assumption of SymFind. This illustrates the fundamental limitation of SymFind.

To account for the limitation of SymFind, we refined the group G of detected symmetries to a subgroup in an additional validation step, before training SymSATNet with those symmetries. In the validation step, we check the usefulness of each part G_i of the expression of G in our grammar. Concretely, we rewrite G only with the part G_i in concern, where all the other parts of G are masked by the trivial groups $\mathcal{I}_k$. We denote each resulting group of this rewriting by G'_i. Then, we project C onto the space of G'_i-equivariant matrices using the corresponding Reynolds operator, and measure the accuracy of SATNet with C before and after the projection over validation examples. Finally, we keep the parts G_i that lead to the improvement of validation accuracy larger than a threshold, and assemble such parts to gain G', which is finally used to train SymSATNet. For example, if $G = (\mathbb{Z}_3 \otimes \mathcal{S}_4) \oplus \mathcal{S}_5$ is discovered by SymFind, we consider the parts $G_1 = \mathbb{Z}_3$, $G_2 = \mathcal{S}_4$, and $G_3 = \mathcal{S}_5$. Then we construct three groups $G'_1 = (\mathbb{Z}_3 \otimes \mathcal{I}_4) \oplus \mathcal{I}_5$, $G'_2 = (\mathcal{I}_3 \otimes \mathcal{S}_4) \oplus \mathcal{I}_5$, and $G'_3 = (\mathcal{I}_3 \otimes \mathcal{I}_4) \oplus \mathcal{S}_5$, and measure the accuracy of SATNet with C projected by each G'_i over validation examples. If G'_1 and G'_3 show accuracy improvements greater than a threshold, we combine G_1 and G_3 to form $G' = (\mathbb{Z}_3 \otimes \mathcal{I}_4) \oplus \mathcal{S}_5$, which is then used to train SymSATNet.

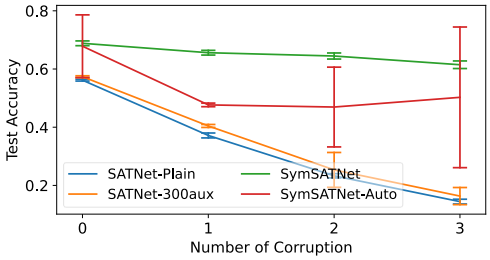
We used 8K training, 1K validation, and 1K test examples to train SymSATNet with symmetries found by SymFind and the validation step. We denote these runs by SymSATNet-Auto. For Sudoku, we took a group G discovered from the SATNet-Plain parameter C at the 10-th training epoch, constructed its subgroup G' via the validation step, and computed an orthogonal basis $\mathcal{B}(G')_s$. SymSATNet was then trained after being initialised by the projection of C with $\mathcal{B}(G')_s$. The same process was applied to Rubik's cube, but with C taken at the 20-th training epoch. The other configurations are the same as in our experiments with manually-specified symmetries.

As shown in Figure 1 and Table 1, SymSATNet-Auto performed the best for Sudoku (99.6%), better than even SymSATNet. For Rubik's cube, SymSATNet-Auto (67.1%) also outperformed SATNet-Plain and SATNet-300aux. During the four trials with Sudoku, SymSATNet-Auto was always given the full symmetries in Sudoku. In two of the four trials with Rubik's cube, SymSATNet-Auto was given correct subgroups $((\mathcal{S}_2 \wr \mathcal{S}_3) \oplus (\mathcal{S}_3 \wr \mathcal{S}_8) \oplus (\mathcal{I}_4 \otimes \mathcal{I}_6)) \otimes (\mathcal{S}_2 \wr \mathcal{S}_3)$ and $((\mathcal{S}_2 \wr \mathcal{S}_3) \oplus (\mathcal{S}_3 \wr (\mathcal{S}_2 \wr \mathcal{S}_4)) \oplus (\mathcal{I}_8 \otimes \mathcal{I}_3)) \otimes (\mathcal{S}_2 \wr \mathcal{S}_3)$, and performed even better than SymSATNet. In the other trials, SymSATNet-Auto exploited slightly incorrect symmetries, but it outperformed SATNet-Plain and SATNet-300aux. These results show that exploiting the partial symmetries of subgroups derived by the validation step are still useful for learing, even when the symmetries are slightly inaccurate.

Robustness to noise We tested robustness of SymSATNet and SymSATNet-Auto to noise by training them with noise-corrupted datasets. We generated noisy Sudoku and Rubik's cube datasets where each training example has a randomly-chosen cell or facelet corrupted with noise: the noise alters the value of the cell or the colour of the facelet to a random value other than the original. We used noisy datasets with 0-3 corrupted instances to measure the test accuracy, and repeated three runs for each dataset to report the average and 95% confidence interval. All the other setups are the same as before. Figure 2 shows the results. In both problems, SymSATNet was the most robust to noise, showing remarkably consistent accuracies. SymSATNet-Auto showed comparable robustness to SATNet-300aux in noisy Sudoku, but outperformed the two baselines in noisy Rubik's cube.



(a) Sudoku



(b) Rubik's cube

Figure 2: Best test accuracies for noisy Sudoku and Rubik's cube datasets.

Next, to show the robustness of SymFind, we applied the algorithm to the problem of restoring a permutation group G from a noise-corrupted G-equivariant symmetric matrix M. We picked four group-permutation pairs (G_i, σ_i) for $i \in [4]$ where $\sigma_1, \sigma_2, \sigma_3$ are random permutations on $[15]$, $[30]$, $[12]$, and σ_4 is the identity permutation on $[8]$, and

$$G_1 = \mathbb{Z}_3 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}_3, \quad G_2 = \mathcal{S}_3 \wr \mathcal{S}_{10}, \quad G_3 = (\mathcal{S}_3 \wr \mathcal{S}_3) \oplus \mathbb{Z}_3, \quad G_4 = \mathcal{S}_2 \otimes \mathcal{S}_2 \otimes \mathcal{S}_2.$$

Then, we generated $\mathrm{grp}(G_i, \sigma_i)$-equivariant symmetric matrices M_i by projecting random matrices with standard normal entries into the space $\mathcal{E}(\mathrm{grp}(G_i, \sigma_i))_s$. Then, Gaussian noises from $\mathcal{N}(0, \omega^2)$ for $\omega = 5 \times 10^{-3}$ are added to M_i's entries, and the resulting matrix M_i' is given to SymFind.

For each (G_i, σ_i), we repeatedly generated M_i' and ran SymFind on M_i' for 1K times, and measured the portion of cases out of $1K$ where SymFind recovered (G_i, σ_i) exactly (full accuracy), and also the portion of cases where SymFind returned a subgroup of (G_i, σ_i) which is not the trivial group $\mathcal{I}_m$ (partial accuracy). As Table 2 shows, the measured full accuracies were in the range of $59.8 - 88.7\%$, and the partial accuracies were in the range of $76.7 - 89.5\%$. These results show the ability of SymFind to recover meaningful and sometimes full symmetries.

Table 2: Full accuracies and partial accuracies of SymFind for each group symmetries.

Group	Full Acc.	Partial Acc.
$\bigoplus_{i=1}^{5} \mathbb{Z}_3$	78.1%	80.1%
$\mathcal{S}_3 \wr \mathcal{S}_{10}$	59.8%	76.7%
$(\mathcal{S}_3 \wr \mathcal{S}_3) \oplus \mathbb{Z}_3$	80.4%	87.9%
$\mathcal{S}_2 \otimes \mathcal{S}_2 \otimes \mathcal{S}_2$	88.7%	89.5%

References

Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1117–1137, 2003.

Fadi A Aloul, Karem A Sakallah, and Igor L Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006.

Sourya Basu, Akshayaa Magesh, Harshit Yadav, and Lav R Varshney. Group equivariant neural architecture search via group decomposition and reinforcement learning. *arXiv e-prints*, pages arXiv–2104, 2021.

Belaıd Benhamou, Tarek Nabhani, Richard Ostrowski, and Mohamed Réda Saıdi. Dynamic symmetry breaking in the satisfiability problem. In *Proceedings of the 16th international conference on Logic for Programming, Artificial intelligence, and Reasoning. LPAR-16, Dakar, Senegal (April 25-may 1, 2010)*, 2010.

Nuri Cingillioglu and Alessandra Russo. Deeplogic: Towards end-to-end differentiable logical reasoning. In *Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019)*, 2019.

Taco S. Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of ICML'16*, pages 2990–2999, 2016.

James Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *AAAI Workshop on Tractable Reasoning*, pages 17–22. Citeseer, 1992.

James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann Publishers Inc., 1996.

Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for cnf. In *Proceedings of the 41st Annual Design Automation Conference*, pages 530–534. Association for Computing Machinery, 2004.

Paul T Darga, Karem A Sakallah, and Igor L Markov. Faster symmetry discovery using sparsity of symmetries. In *2008 45th ACM/IEEE Design Automation Conference*, pages 149–154. IEEE, 2008.

Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. *Advances in Neural Information Processing Systems*, 34, 2021.

Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 83–100. Springer, 2017.

David S Dummit and Richard M Foote. *Abstract algebra*, volume 1999. Prentice Hall Englewood Cliffs, NJ, 1991.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

Marc Finzi, Max Welling, and Andrew Gordon Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *Proceedings of ICML'21*, pages 3318–3328, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Subrata Majumdar, Kalyan Kumar Dey, and Mohd Altab Hossain. Direct product and wreath product of transformation semigroups. *GANIT: Journal of Bangladesh Mathematical Society*, 31:1–7, Apr. 2012.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *Proceedings of ICLR'19*, 2019.

Brendan D. McKay. *Backtrack programming and the graph isomorphism problem*. PhD thesis, University of Melbourne, 1976.

Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

Kyubyong Park. Can convolutional neural networks crack sudoku puzzles?, 2018.

Sever Topan, David Rolnick, and Xujie Si. Techniques for symbol grounding with satnet. In *Advances in Neural Information Processing Systems*, volume 34, pages 20733–20744. Curran Associates, Inc., 2021.

Charles F Van Loan and Nikos Pitsianis. Approximation with kronecker products. In *Linear algebra for large scale and real-time applications*, pages 293–314. Springer, 1993.

Po-Wei Wang, Wei-Cheng Chang, and J. Zico Kolter. The mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. *CoRR*, abs/1706.00476, 2018.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of ICML'19*, pages 6545–6554, 2019. URL https://github.com/locuslab/SATNet.git.

Renhao Wang, Marjan Albooyeh, and Siamak Ravanbakhsh. Equivariant networks for hierarchical structures. In *Proceedings of NeurIPS'20*, pages 13806–13817, 2020.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Proceedings of NeurIPS'17*, pages 2319–2328, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Section 3.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]