SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver

Po-Wei Wang [1] Priya L. Donti [1 2] Bryan Wilder [3] Zico Kolter [1 4]

Abstract

Integrating logical reasoning within deep learning architectures has been a major goal of modern AI systems. In this paper, we propose a new direction toward this goal by introducing a differentiable (smoothed) maximum satisfiability (MAXSAT) solver that can be integrated into the loop of larger deep learning systems. Our (approximate) solver is based upon a fast coordinate descent approach to solving the semidefinite program (SDP) associated with the MAXSAT problem. We show how to analytically differentiate through the solution to this SDP and efficiently solve the associated backward pass. We demonstrate that by integrating this solver into end-to-end learning systems, we can learn the logical structure of challenging problems in a minimally supervised fashion. In particular, we show that we can learn the parity function using single-bit supervision (a traditionally hard task for deep networks) and learn how to play 9×9 Sudoku solely from examples. We also solve a "visual Sudoku" problem that maps images of Sudoku puzzles to their associated logical solutions by combining our MAXSAT solver with a traditional convolutional architecture. Our approach thus shows promise in integrating logical structures within deep learning.

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA [2]Department of Engineering & Public Policy, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA [3]Department of Computer Science, University of Southern California, Los Angeles, California, USA [4]Bosch Center for Artificial Intelligence, Pittsburgh, Pennsylvania, USA. Correspondence to: Po-Wei Wang <poweiw@cs.cmu.edu>, Priya Donti <pdonti@cmu.edu>, Bryan Wilder <bwilder@usc.edu>, Zico Kolter <zkolter@cs.cmu.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

1. Introduction

Although modern deep learning has produced groundbreaking improvements in a variety of domains, state-of-the-art methods still struggle to capture "hard" and "global" constraints arising from discrete logical relationships. Motivated by this deficiency, there has been a great deal of recent interest in integrating logical or symbolic reasoning into neural network architectures (Palm et al., 2017; Yang et al., 2017; Cingillioglu & Russo, 2018; Evans & Grefenstette, 2018). However, with few exceptions, previous work primarily focuses on integrating *preexisting* relationships into a larger differentiable system via tunable continuous parameters, not on *discovering* the discrete relationships that produce a set of observations in a truly end-to-end fashion. As an illustrative example, consider the popular logic-based puzzle game Sudoku, in which a player must fill in a 9×9 partially-filled grid of numbers to satisfy specific constraints. If the rules of Sudoku (i.e. the relationships between problem variables) are not given, then it may be desirable to jointly learn the rules of the game *and* learn how to solve Sudoku puzzles in an end-to-end manner.

We consider the problem of learning logical structure specifically as expressed by satisfiability problems – concretely, problems that are well-modeled as instances of SAT or MAXSAT (the optimization analogue of SAT). This is a rich class of domains encompassing much of symbolic AI, which has traditionally been difficult to incorporate into neural network architectures since neural networks rely on continuous and differentiable parameterizations. Our key contribution is to develop and derive a differentiable smoothed MAXSAT solver that can be embedded within more complex deep architectures, and show that this solver enables effective end-to-end learning of logical relationships from examples (without hard-coding of these relationships). More specifically, we build upon recent work in fast block coordinate descent methods for solving SDPs (Wang et al., 2017) to build a differentiable solver for the smoothed SDP relaxation of MAXSAT. We provide an efficient mechanism to differentiate through the optimal solution of this SDP by using a similar block coordinate descent solver as used in the forward pass. Our module is amenable to GPU acceleration, greatly improving training scalability.

Using this framework, we are able to solve several problems that, despite their simplicity, prove essentially impossible for traditional deep learning methods and existing logical learning methods to reliably learn without any prior knowledge. In particular, we show that we can learn the parity function, known to be challenging for deep classifiers (Shalev-Shwartz et al., 2017), with only single bit supervision. We also show that we can learn to play 9×9 Sudoku, a problem that is challenging for modern neural network architectures (Palm et al., 2017). We demonstrate that our module quickly recovers the constraints that describe a feasible Sudoku solution, learning to correctly solve 98.3% of puzzles at test time *without any hand-coded knowledge of the problem structure*. Finally, we show that we can embed this differentiable solver into larger architectures, solving a "visual Sudoku" problem where the input is an image of a Sudoku puzzle rather than a binary representation. We show that, in a fully end-to-end setting, our method is able to integrate classical convolutional networks (for digit recognition) with the differentiable MAXSAT solver (to learn the logical portion). Taken together, this presents a substantial advance toward a major goal of modern AI: integrating logical reasoning into deep learning architectures.

2. Related work

Recently, the deep learning community has given increasing attention to the concept of embedding complex, "non-traditional" layers within deep networks in order to train systems end-to-end. Major examples have included logical reasoning modules and optimization layers. Our work combines research in these two areas by exploiting optimization-based relaxations of logical reasoning structures, namely an SDP relaxation of MAXSAT. We explore each of these relevant areas of research in more detail below.

Logical reasoning in deep networks. Our work is closely related to recent interest in integrating logical reasoning into deep learning architectures (Garcez et al., 2015). Most previous systems have focused on creating differentiable modules from an existing set of *known relationships*, so that a deep network can learn the parameters of these relationships (Dai et al., 2018; Manhaeve et al., 2018; Sourek et al., 2018; Xu et al., 2018; Hu et al., 2016; Yang et al., 2017; Selsam et al., 2018). For example, Palm et al. (2017) introduce a network that carries out relational reasoning using hand-coded information about which variables are allowed to interact, and test this network on 9×9 Sudoku. Similarly, Evans & Grefenstette (2018) integrate inductive logic programming into neural networks by constructing differentiable SAT-based representations for specific "rule templates." While these networks are seeded with prior information about the relationships between variables, our approach learns these relationships *and* their associated parameters end-to-end. While other recent work has also tried to jointly learn rules and parameters, the problem classes captured by these architectures have been limited. For instance, Cingillioglu & Russo (2018) train a neural network to apply a specific class of logic programs, namely the binary classification problem of whether a given set of propositions entails a specific conclusion. While this approach does not rely on prior hand-coded structure, our method applies to a broader class of domains, encompassing any problem reducible to MAXSAT.

Differentiable optimization layers. Our work also fits within a line of research leveraging optimization as a layer in neural networks. For instance, previous work has introduced differentiable modules for quadratic programs (Amos & Kolter, 2017; Donti et al., 2017), submodular optimization problems (Djolonga & Krause, 2017; Tschiatschek et al., 2018; Wilder et al., 2018), and equilibrium computation in zero-sum games (Ling et al., 2018). To our knowledge, ours is the first work to use differentiable SDP relaxations to capture relationships between discrete variables.

MAXSAT SDP relaxations. We build on a long line of research exploring SDP relaxations as a tool for solving MAXSAT and related problems. Classical work shows that such relaxations produce strong approximation guarantees for MAXCUT and MAX-2SAT (Goemans & Williamson, 1995), and are empirically tighter than standard linear programming relaxations (Gomes et al., 2006). More recent work, e.g. Wang et al. (2017); Wang & Kolter (2019), has developed low-rank SDP solvers for general MAXSAT problems. We extend the work of Wang et al. (2017) to create a differentiable optimization-based MAXSAT solver that can be employed in the loop of deep learning.

3. A differentiable satisfiability solver

The MAXSAT problem is the optimization analogue of the well-known satisfiability (SAT) problem, in which the goal is to *maximize* the number of clauses satisfied. We present a differentiable, smoothed approximate MAXSAT solver that can be integrated into modern deep network architectures. This solver uses a fast coordinate descent approach to solving an SDP relaxation of MAXSAT. We describe our MAXSAT SDP relaxation as well as the forward pass of our MAXSAT deep network layer (which employs this relaxation). We then show how to analytically differentiate through the MAXSAT SDP and efficiently solve the associated backward pass.

3.1. Solving an SDP formulation of satisfiability

Consider a MAXSAT instance with n variables and m clauses. Let $\tilde{v} \in \{-1, 1\}^n$ denote binary assignments of the problem variables, where $\tilde{v}_i$ is the truth value of

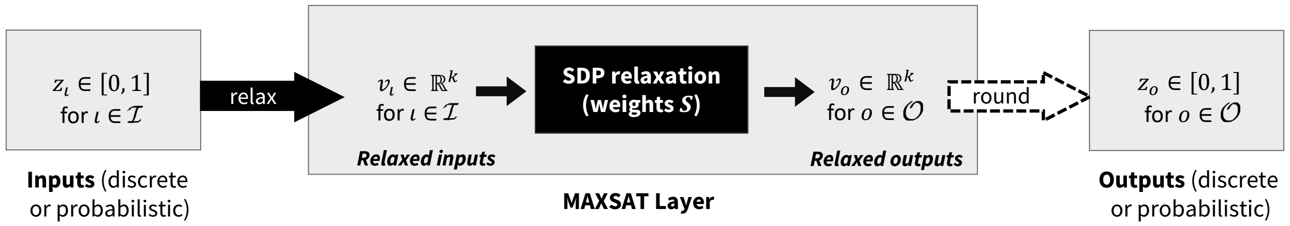


Figure 1. The forward pass of our MAXSAT layer. The layer takes as input the discrete or probabilistic assignments of known MAXSAT variables, and outputs guesses for the assignments of unknown variables via a MAXSAT SDP relaxation with weights S.

variable $i \in \{1,\dots,n\}$, and define $\tilde{s}_i \in \{-1,0,1\}^m$ for $i \in \{1,\dots,n\}$, where $\tilde{s}_{ij}$ denotes the sign of $\tilde{v}_i$ in clause $j \in \{1,\dots,m\}$. We then write the MAXSAT problem as

$$\underset{\tilde{v}\in\{-1,1\}^n}{\text{maximize}} \sum_{j=1}^{m} \bigvee_{i=1}^{n} \mathbf{1}\{\tilde{s}_{ij}\tilde{v}_i > 0\}. \tag{1}$$

As derived in Goemans & Williamson (1995); Wang & Kolter (2019), to form a semidefinite relaxation of (1), we first relax the discrete variables $\tilde{v}_i$ into associated continuous variables $v_i \in \mathbb{R}^k$, $\|v_i\| = 1$ with respect to some "truth direction" $v_\top \in \mathbb{R}^k$, $\|v_\top\| = 1$. Specifically, we relate the continuous v_i to the discrete $\tilde{v}_i$ probabilistically via $P(\tilde{v}_i = 1) = \cos^{-1}(-v_i^T v_\top)/\pi$ based on randomized rounding (Goemans & Williamson (1995); see Section 3.2.4). We additionally define a coefficient vector $\tilde{s}_\top = \{-1\}^m$ associated with $v_\top$. Our SDP relaxation of MAXSAT is then

$$\begin{aligned} &\underset{V\in\mathbb{R}^{k\times(n+1)}}{\text{minimize}} \quad \langle S^T S, V^T V\rangle, \\ &\text{subject to} \quad \|v_i\| = 1, \quad i = \top, 1, \dots, n \end{aligned} \tag{2}$$

where $V \equiv \begin{bmatrix} v_\top & v_1 & \dots & v_n \end{bmatrix} \in \mathbb{R}^{k\times(n+1)}$, and $S \equiv \begin{bmatrix} \tilde{s}_\top & \tilde{s}_1 & \dots & \tilde{s}_n \end{bmatrix} \operatorname{diag}(1/\sqrt{4|\tilde{s}_j|}) \in \mathbb{R}^{m\times(n+1)}$. We note that this problem is a low-rank (but non-convex) formulation of MIN-UNSAT, which is equivalent to MAXSAT. This formulation can be rewritten as an SDP, and has been shown to recover the optimal SDP solution given $k > \sqrt{2n}$ (Barvinok, 1995; Pataki, 1998).

Despite its non-convexity, problem (2) can then be solved optimally via coordinate descent for all $i = \top, 1, \dots, n$. In particular, the objective terms that depend on v_i are given by $v_i^T \sum_{j=0}^{n} s_i^T s_j v_j$, where s_i is the ith column vector of S. Minimizing this quantity over v_i subject to the constraint that $\|v_i\| = 1$ yields the coordinate descent update

$$v_i = -g_i/\|g_i\|, \quad \text{where } g_i = VS^T s_i - \|s_i\|^2 v_i. \tag{3}$$

These updates provably converge to the globally optimal fixed point of the SDP (2) (Wang et al., 2017). A more detailed derivation of this update can be found in Appendix A.

Algorithm 1 SATNet Layer

```
1:  procedure INIT()
2:      // rank, num aux vars, initial weights, rand vectors
3:      init $m, n_{\text{aux}}, S$
4:      init random unit vectors $v_\top, v_i^{\text{rand}}\ \forall i \in \{1,\dots,n\}$
5:      // smallest k for which (2) recovers SDP solution
6:      set $k = \sqrt{2n} + 1$
7:
8:  procedure FORWARD($Z_\mathcal{I}$)
9:      compute $V_\mathcal{I}$ from $Z_\mathcal{I}$ via (5)
10:     compute $V_\mathcal{O}$ from $V_\mathcal{I}$ via coord. descent (Alg 2)
11:     compute $Z_\mathcal{O}$ from $V_\mathcal{O}$ via (7)
12:     return $Z_\mathcal{O}$
13:
14: procedure BACKWARD($\partial\ell/\partial Z_\mathcal{O}$)
15:     compute $\partial\ell/\partial V_\mathcal{O}$ via (8)
16:     compute $U$ from $\partial\ell/\partial V_\mathcal{O}$ via coord. descent (Alg 3)
17:     compute $\partial\ell/\partial Z_\mathcal{I}$, $\partial\ell/\partial S$ from $U$ via (12), (11)
18:     return $\partial\ell/\partial Z_\mathcal{I}$
```

3.2. SATNet: Satisfiability solving as a layer

Using our MAXSAT SDP relaxation and associated coordinate descent updates, we create a deep network layer for satisfiability solving (SATNet). Define $\mathcal{I} \subset \{1,\dots,n\}$ to be the indices of MAXSAT variables with known assignments, and let $\mathcal{O} \equiv \{1,\dots,n\} \setminus \mathcal{I}$ correspond to the indices of variables with unknown assignments. Our layer admits probabilistic or binary inputs $z_\iota \in [0,1]$, $\iota \in \mathcal{I}$, and then outputs the assignments of unknown variables $z_o \in [0,1]$, $o \in \mathcal{O}$ which are similarly probabilistic or (optionally, at test time) binary. We let $Z_\mathcal{I} \in [0,1]^{|\mathcal{I}|}$ and $Z_\mathcal{O} \in [0,1]^{|\mathcal{O}|}$ refer to all input and output assignments, respectively.

The outputs $Z_\mathcal{O}$ are generated from inputs $Z_\mathcal{I}$ via the SDP (2), and the weights of our layer correspond to the SDP's low-rank coefficient matrix S. This forward pass procedure is pictured in Figure 1. We describe the steps of layer initialization and the forward pass in Algorithm 1, and in more detail below.

3.2.1. Layer initialization

When initializing SATNet, the user must specify a maximum number of clauses m that this layer can represent. It is often desirable to set m to be low; in particular, *low-rank structure* can prevent overfitting and thus improve generalization.

Given this low-rank structure, a user may wish to somewhat increase the layer's representational ability via auxiliary variables. The high-level intuition here follows from the conjunctive normal form (CNF) representation of boolean satisfaction problems; adding additional variables to a problem can dramatically reduce the number of CNF clauses needed to describe that problem, as these variables play a role akin to register memory that is useful for inference.

Finally, we set $k = \sqrt{2n} + 1$, where here n captures the number of actual problem variables in addition to auxiliary variables. This is the minimum value of k required for our MAXSAT relaxation (2) to recover the optimal solution of its associated SDP (Barvinok, 1995; Pataki, 1998).

3.2.2. Step 1: Relaxing layer inputs

Our layer first relaxes its inputs $Z_{\mathcal{I}}$ into continuous vectors for use in the SDP formulation (2). That is, we relax each layer input $z_\iota, \iota \in \mathcal{I}$ to an associated random unit vector $v_\iota \in \mathbb{R}^k$ so that

$$v_\iota^T v_\top = -\cos(\pi z_\iota). \tag{4}$$

(This equation is derived from the probabilistic relationship described in Section 3.1 between discrete variables and their continuous relaxations.) Constraint (4) can be satisfied by

$$v_\iota = -\cos(\pi z_\iota) v_\top + \sin(\pi z_\iota)(I_k - v_\top v_\top^T) v_\iota^{\text{rand}}, \tag{5}$$

where v_ι^{rand} is a random unit vector. For simplicity, we use the notation $V_{\mathcal{I}} \in \mathbb{R}^{k \times |\mathcal{I}|}$ (i.e. the $\mathcal{I}$-indexed column subset of V) to collectively refer to all relaxed layer inputs derived via Equation (5).

3.2.3. Step 2: Generating continuous relaxations of outputs via SDP

Given the continuous input relaxations $V_{\mathcal{I}}$, our layer employs the coordinate descent updates (3) to compute values for continuous output relaxations v_o, $o \in \mathcal{O}$ (which we collectively refer to as $V_{\mathcal{O}} \in \mathbb{R}^{k \times |\mathcal{O}|}$). Notably, coordinate descent updates are *only computed for output variables*, i.e. are not computed for variables whose assignments are given as input to the layer.

Our coordinate descent algorithm for the forward pass is detailed in Algorithm 2. This algorithm maintains the term $\Omega = VS^T$ needed to compute g_o, and then modifies it via a rank-one update during each inner iteration. Accordingly, the per-iteration runtime is $O(nmk)$ (and in practice, only a small number of iterations is required for convergence).

Algorithm 2 Forward pass coordinate descent

```
1: input $V_{\mathcal{I}}$          // inputs for known variables
2: init $v_o$ with random vector $v_o^{\text{rand}}$, $\forall o \in \mathcal{O}$.
3: compute $\Omega = VS^T$
4: while not converged do
5:     for $o \in \mathcal{O}$ do     // for all output variables
6:         compute $g_o = \Omega s_o - \|s_o\|^2 v_o$ as in (3)
7:         compute $v_o = -g_o / \|g_o\|$ as in (3)
8:         update $\Omega = \Omega + (v_o - v_o^{\text{prev}}) s_o^T$
9: output $V_{\mathcal{O}}$          // final guess for output cols of V
```

3.2.4. Step 3: Generating discrete or probabilistic outputs

Given the relaxed outputs $V_{\mathcal{O}}$ from coordinate descent, our layer converts these outputs to discrete or probabilistic variable assignments $Z_{\mathcal{O}}$ via either thresholding or randomized rounding (which we describe here).

The main idea of randomized rounding is that for every $v_o, o \in \mathcal{O}$, we can take a random hyperplane r from the unit sphere and assign

$$\tilde{v}_o = \begin{cases} 1 & \text{if } \operatorname{sign}(v_o^T r) = \operatorname{sign}(v_\top^T r) \\ -1 & \text{otherwise} \end{cases}, \quad o \in \mathcal{O}, \tag{6}$$

where $\tilde{v}_o$ is the boolean output for v_o. Intuitively, this scheme sets $\tilde{v}_o$ to "true" if and only if v_o and the truth vector $v_\top$ are on the same side of the random hyperplane r. Given the correct weights S, this randomized rounding procedure assures an optimal expected approximation ratio for certain NP-hard problems (Goemans & Williamson, 1995).

During training, we do not explicitly perform randomized rounding. We instead note that the probability that v_o and $v_\top$ are on the same side of any given r is

$$P(\tilde{v}_o) = \cos^{-1}(-v_o^T v_\top)/\pi, \tag{7}$$

and thus set $z_o = P(\tilde{v}_o)$ to equal this probability.

During testing, we can either output probabilistic outputs in the same fashion, or output discrete assignments via thresholding or randomized rounding. If using randomized rounding, we round multiple times, and then set z_o to be the boolean solution maximizing the MAXSAT objective in Equation (1). Prior work has observed that such repeated rounding improves approximation ratios in practice, especially for MAXSAT problems (Wang & Kolter, 2019).

3.3. Computing the backward pass

We now derive backpropagation updates through our SATNet layer to enable its integration into a neural network. That is, given the gradients $\partial \ell / \partial Z_{\mathcal{O}}$ of the network loss ℓ

with respect to the layer outputs, we must compute the gradients $\partial\ell/\partial Z_{\mathcal{I}}$ with respect to layer inputs and $\partial\ell/\partial S$ with respect to layer weights. As it would be inefficient in terms of time and memory to explicitly unroll the forward-pass computations and store intermediate Jacobians, we instead derive analytical expressions to *compute the desired gradients directly*, employing an efficient coordinate descent algorithm. The procedure for computing these gradients is summarized in Algorithm 1 and derived below.

3.3.1. From probabilistic outputs to their continuous relaxations

Given $\partial\ell/\partial Z_{\mathcal{O}}$ (with respect to the layer outputs), we first derive an expression for $\partial\ell/\partial V_{\mathcal{O}}$ (with respect to the output relaxations) by pushing gradients through the probability assignment mechanism described in Section 3.2.4. That is, for each $o \in \mathcal{O}$,

$$\frac{\partial\ell}{\partial v_o} = \left(\frac{\partial\ell}{\partial z_o}\right)\left(\frac{\partial z_o}{\partial v_o}\right) = \left(\frac{\partial\ell}{\partial z_o}\right)\frac{1}{\pi\sin(\pi z_o)}v_\top, \quad (8)$$

where we obtain $\partial z_o/\partial v_o$ by differentiating through Equation (7) (or, more readily, by implicitly differentiating through its rearrangement $\cos(\pi z_o) = -v_\top^T v_o$).

3.3.2. Backpropagation through the SDP

Given the analytical form for $\partial\ell/\partial V_{\mathcal{O}}$ (with respect to the output relaxations), we next seek to derive $\partial\ell/\partial V_{\mathcal{I}}$ (with respect to the input relaxations) and $\partial\ell/\partial S$ (with respect to the layer weights) by pushing gradients through our SDP solution procedure (Section 3.2.3). We describe the analytical form for the resultant gradients in Theorem 1.

Theorem 1. *Define $P_o \equiv I_k - v_o v_o^T$ for each $o \in \mathcal{O}$. Then, define $U \in \mathbb{R}^{k\times n}$, where the columns $U_{\mathcal{I}} = 0$ and the columns $U_{\mathcal{O}}$ are given by*

$$\mathrm{vec}(U_{\mathcal{O}}) = (P((C+D)\otimes I_k)P)^\dagger\, \mathrm{vec}\left(\frac{\partial\ell}{\partial V_{\mathcal{O}}}\right), \quad (9)$$

where $P \equiv \mathrm{diag}(P_o)$, where $C \equiv S_{\mathcal{O}}^T S_{\mathcal{O}} - \mathrm{diag}(\|s_o\|^2)$, and where $D \equiv \mathrm{diag}(\|g_o\|)$. Then, the gradient of the network loss ℓ with respect to the relaxed layer inputs is

$$\frac{\partial\ell}{\partial V_{\mathcal{I}}} = -\Big(\sum_{o\in\mathcal{O}} u_o s_o^T\Big) S_{\mathcal{I}}, \quad (10)$$

where $S_{\mathcal{I}}$ is the $\mathcal{I}$-indexed column subset of S, and the gradient with respect to the layer weights is

$$\frac{\partial\ell}{\partial S} = -\Big(\sum_{o\in\mathcal{O}} u_o s_o^T\Big)^T V - (SV^T)U. \quad (11)$$

We defer the derivation of Theorem 1 to Appendix B. Although this derivation is somewhat involved, the concept at a high level is quite simple: we differentiate the solution of the SDP problem (Section 3.1) with respect to the problem's parameters and input, which requires computing the (relatively large) matrix-vector solve given in Equation (9).

Algorithm 3 Backward pass coordinate descent

```
1: input {$\partial\ell/\partial v_o \mid o \in \mathcal{O}$}   // grads w.r.t. relaxed outputs
2: // Compute $U_{\mathcal{O}}$ from Equation (9)
3: init $U_{\mathcal{O}} = 0$ and $\Psi = (U_{\mathcal{O}})S_{\mathcal{O}}^T = 0$
4: while not converged do
5:     for $o \in \mathcal{O}$ do    // for all output variables
6:         compute $\mathsf{d}g_o = \Psi s_o - \|s_o\|^2 u_o - \partial\ell/\partial v_o$.
7:         compute $u_o = -P_o \mathsf{d}g_o/\|g_o\|$.
8:         update $\Psi = \Psi + (u_o - u_o^{\text{prev}})s_o^T$
9: output $U_{\mathcal{O}}$
```

To solve Equation (9), we use a coordinate descent approach that closely mirrors the coordinate descent procedure employed in the forward pass, and which has similar fast convergence properties. This procedure, described in Algorithm 3, enables us to compute the desired gradients without needing to maintain intermediate Jacobians explicitly. Mirroring the forward pass, we use rank-one updates to maintain and modify the term $\Psi = US^T$ needed to compute $\mathsf{d}g_o$, which again enables our algorithm to run in $O(nmk)$ time. We defer the derivation of Algorithm 3 to Appendix D.

3.3.3. From relaxed to original inputs

As a final step, we must use the gradient $\partial\ell/\partial V_{\mathcal{I}}$ (with respect to the input relaxations) to derive the gradient $\partial\ell/\partial Z_{\mathcal{I}}$ (with respect to the actual inputs) by pushing gradients through the input relaxation procedure described in Section 3.2.2. For each $\iota \in \mathcal{I}$, we see that

$$\begin{aligned}\frac{\partial\ell}{\partial z_\iota} &= \frac{\partial\ell}{\partial z_\iota^\star} + \left(\frac{\partial\ell}{\partial v_\iota}\right)^T \frac{\partial v_\iota}{\partial z_\iota} \\ &= \frac{\partial\ell}{\partial z_\iota^\star} - \left(\frac{\partial v_\iota}{\partial z_\iota}\right)^T \left(\sum_{o\in\mathcal{O}} u_o s_o^T\right) s_\iota\end{aligned} \quad (12)$$

where

$$\frac{\partial v_\iota}{\partial z_\iota} = \pi\left(\sin(\pi z_\iota)v_\top + \cos(\pi z_\iota)(I_k - v_\top v_\top^T)v_\iota^{\text{rand}}\right), \quad (13)$$

and where $\partial\ell/\partial z_\iota^\star$ captures any direct dependence of ℓ on $z_\iota^\star$ (as opposed to dependence through v_ι). Here, the expression for $\partial\ell/\partial v_\iota$ comes from Equation (10), and we obtain $\partial v_\iota/\partial z_\iota$ by differentiating Equation (5).

3.4. An efficient GPU implementation

The coordinate descent updates in Algorithms 2 and 3 dominate the computational costs of the forward and backward passes, respectively. We thus present an efficient, parallel GPU implementation of these algorithms to speed up training and inference. During the inner loop of coordinate descent, our implementation parallelizes the computation of all g_o (dg_o) terms by parallelizing the computation of Ω (Ψ), as well as of all rank-one updates of Ω (Ψ). This underscores the benefit of using a low-rank SDP formulation in our MAXSAT layer, as traditional full-rank coordinate descent cannot be efficiently parallelized. We find in our preliminary benchmarks that our GPU CUDA-C implementation is up to $18-30$x faster than the corresponding OpenMP implementation run on Xeon CPUs. Source code for our implementation is available at `https://github.com/locuslab/SATNet`.

4. Experiments

We test our MAXSAT layer approach in three domains that are traditionally difficult for neural networks: learning the parity function with single-bit supervision, learning 9×9 Sudoku solely from examples, and solving a “visual Sudoku” problem that generates the logical Sudoku solution given an input image of a Sudoku puzzle. We find that in all cases, we are able to perform substantially better on these tasks than previous deep learning-based approaches.

4.1. Learning parity (chained XOR)

This experiment tests SATNet’s ability to differentiate through many successive SAT problems by learning to compute the parity function. The parity of a bit string is defined as one if there is an odd number of ones in the sequence and zero otherwise. The task is to map input sequences to their parity, given a dataset of example sequence/parity pairs. Learning parity functions from such single-bit supervision is known to pose difficulties for conventional deep learning approaches (Shalev-Shwartz et al., 2017). However, parity is simply a logic function – namely, a sequence of XOR operations applied successively to the input sequence.

Hence, for a sequence of length L, we construct our model to contain a sequence of $L-1$ SATNet layers with tied weights (similar to a recurrent network). The first layer receives the first two binary values as input, and layer d receives value d along with the rounded output of layer $d-1$. If each layer learns to compute the XOR function, the combined system will correctly compute parity. However, this requires the model to coordinate a long series of SAT problems without any intermediate supervision.

Figure 2 shows that our model accomplishes this task for input sequences of length $L = 20$ and $L = 40$. For each sequence length, we generate a dataset of 10K random examples (9K training and 1K testing). We train our model using cross-entropy loss and the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-1}. We compare to an LSTM sequence classifier, which uses 100 hidden units and a learning rate of 10^{-3} (we tried varying the architecture and learning rate but did not observe any improvement). In each case, our model quickly learns the target function, with error on the held-out set converging to zero within 20 epochs. In contrast, the LSTM is unable to learn an appropriate representation, with only minor improvement over the course of 100 training epochs; across both input lengths, it achieves a testing error rate of at best 0.476 (where a random guess achieves value 0.5).

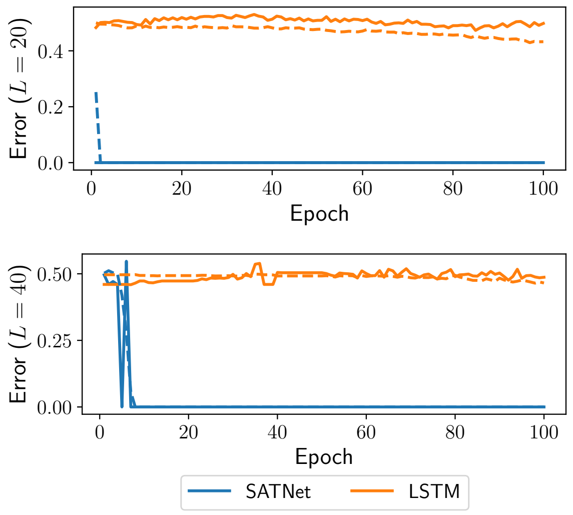


Figure 2. Error rate for the parity task with $L = 20$ (top) and $L = 40$ (bottom). Solid lines denote test values, while dashed lines represent training values.

4.2. Sudoku (original and permuted)

In this experiment, we test SATNet’s ability to infer and recover constraints simply from bit supervision (i.e. without any hard-coded specification of how bits are related). We demonstrate this property via Sudoku. In Sudoku, given a (typically) 9×9 partially-filled grid of numbers, a player must fill in the remaining empty grid cells such that each row, each column, and each of nine 3×3 subgrids contains exactly one of each number from 1 through 9. While this constraint satisfaction problem is computationally easy to solve once the rules of the game are specified, actually *learning the rules of the game*, i.e. the hard constraints of the puzzle, has proved challenging for traditional neural network architectures. In particular, Sudoku problems are often solved computationally via tree search, and while tree search cannot be easily performed by neural networks, it is easily expressible using SAT and MAXSAT problems.

We construct a SATNet model for this task that takes as input a logical (bit) representation of the initial Sudoku board

Model	Train	Test
ConvNet	72.6%	0.04%
ConvNetMask	91.4%	15.1%
SATNet (ours)	99.8%	**98.3%**

(a) Original Sudoku.

Model	Train	Test
ConvNet	0%	0%
ConvNetMask	0.01%	0%
SATNet (ours)	99.7%	**98.3%**

(b) Permuted Sudoku.

Model	Train	Test
ConvNet	0.31%	0%
ConvNetMask	89%	0.1%
SATNet (ours)	93.6%	**63.2%**

(c) Visual Sudoku. (Note: the theoretical "best" test accuracy for our architecture is 74.7%.)

Table 1. Results for 9×9 Sudoku experiments with 9K train/1K test examples. We compare our SATNet model against a vanilla convolutional neural network (ConvNet) as well as one that receives a binary mask indicating which bits need to be learned (ConvNetMask).

along with a mask representing which bits must be learned (i.e. all bits in empty Sudoku cells). This input is vectorized, which means that our SATNet model cannot exploit the locality structure of the input Sudoku grid when learning to solve puzzles. Given this input, the SATNet layer then outputs a bit representation of the Sudoku board with guesses for the unknown bits. Our model architecture consists of a single SATNet layer with 300 auxiliary variables and low rank structure $m = 600$, and we train it to minimize a digit-wise negative log likelihood objective (optimized via Adam with a 2×10^{-3} learning rate).

We compare our model to a convolutional neural network baseline modeled on that of Park (2016), which interprets the bit inputs as 9 input image channels (one for each square in the board) and uses a sequence of 10 convolutional layers (each with 512 3×3 filters) to output the solution. The ConvNet makes explicit use of locality in the input representation since it treats the nine cells within each square as a single image. We also compare to a version of the ConvNet which receives a binary mask indicating which bits need to be learned (ConvNetMask). The mask is input as a set of additional image channels in the same format as the board. We trained both architectures using mean squared error (MSE) loss (which gave better results than negative log likelihood for this architecture). The loss was optimized using Adam (learning rate 10^{-4}). We additionally tried to train an OptNet (Amos & Kolter, 2017) model for comparison, but this model made little progress even after a few days of training. (We compare our method to OptNet on a simpler 4×4 version of the Sudoku problem in Appendix E.)

Our results for the traditional 9×9 Sudoku problem (over 9K training examples and 1K test examples) are shown in Table 1. (Convergence plots for this experiment are shown in Appendix F.) Our model is able to learn the constraints of the Sudoku problem, achieving high accuracy early in the training process (95.0% test accuracy in 22 epochs/37 minutes on a GTX 1080 Ti GPU), and demonstrating 98.3% board-wise test accuracy after 100 training epochs (172 minutes). On the other hand, the ConvNet baseline does poorly. It learns to correctly solve 72.6% of puzzles in the training set but fails altogether to generalize: accuracy on the held-out set reaches at most 0.04%. The ConvNetMask baseline, which receives a binary mask denoting which entries must be completed, performs only somewhat better, correctly solving 15.1% of puzzles in the held-out set. We note that our test accuracy is qualitatively similar to the results obtained in Palm et al. (2017), but that our network is able to learn the structure of Sudoku *without explicitly encoding the relationships between variables*.

To underscore that our architecture truly learns the rules of the game, as opposed to overfitting to locality or other structure in the inputs, we test our SATNet architecture on *permuted* Sudoku boards, i.e. boards for which we apply a fixed permutation of the underlying bit representation (and adjust the corresponding input masks and labels accordingly). This removes any locality structure, and the resulting Sudoku boards do not have clear visual analogues that can be solved by humans. However, the relationships between bits are unchanged (modulo the permutation) and should therefore be discoverable by architectures that can truly learn the underlying logical structure. Table 1 shows results for this problem in comparison to the convolutional neural network baselines. Our architecture is again able to learn the rules of the (permuted) game, demonstrating the same 98.3% board-wise test accuracy as in the original game. In contrast, the convolutional neural network baselines perform even more poorly than in the original game (achieving 0% test accuracy even with the binary mask as input), as there is little locality structure to exploit. Overall, these results demonstrate that SATNet can truly learn the logical relationships between discrete variables.

4.3. Visual Sudoku

In this experiment, we demonstrate that SATNet can be integrated into larger deep network architectures for end-to-end training. Specifically, we solve the visual Sudoku problem: that is, given an *image representation* of a Sudoku board (as opposed to a one-hot encoding or other logical representation) constructed with MNIST digits, our network must output a *logical solution* to the associated Sudoku problem.

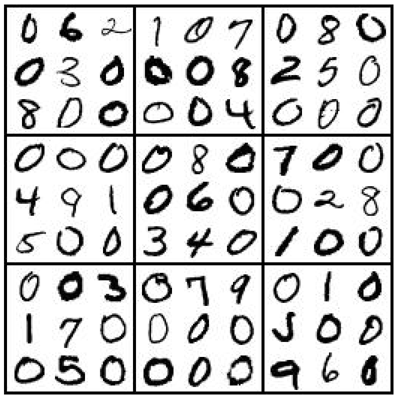


Figure 3. An example visual Sudoku image input, i.e. an image of a Sudoku board constructed with MNIST digits. Cells filled with the numbers 1-9 are fixed, and zeros represent unknowns.

An example input is shown in Figure 3. This problem cannot traditionally be represented well by neural network architectures, as it requires the ability to combine multiple neural network layers *without* hard-coding the logical structure of the problem into intermediate logical layers.

Our architecture for this problem uses a convolutional neural network connected to a SATNet layer. Specifically, we apply a convolutional layer for digit classification (which uses the LeNet architecture (LeCun et al., 1998)) to each cell of the Sudoku input. Each cell-wise probabilistic output of this convolutional layer is then fed as logical input to the SATNet layer, along with an input mask (as in Section 4.2). This SATNet layer employs the same architecture and training parameters as described in the previous section. The whole model is trained end-to-end to minimize cross-entropy loss, and is optimized via Adam with learning rates 2×10^{-3} for the SATNet layer and 10^{-5} for the convolutional layer.

We compare our approach against a convolutional neural network which combines two sets of convolutional layers. First, the visual inputs are passed through the same convolutional layer as in our SATNet model, which outputs a probabilistic bit representation. Next, this representation is passed through the convolutional architecture that we compared to for the original Sudoku problem, which outputs a solution. We use the same training approach as above.

Table 1 summarizes our experimental results (over 9K training examples and 1K test examples); additional plots are shown in Appendix F. We contextualize these results against the theoretical "best" testing accuracy of 74.7%, which accounts for the Sudoku digit classification accuracy of our specific convolutional architecture; that is, assuming boards with 36.2 out of 81 filled cells on average (as in our test set) and an MNIST model with 99.2% test accuracy (LeCun et al., 1998), we would expect a perfect Sudoku solver to output the correct solution 74.7% ($= 0.992^{36.2}$) of the time. In 100 epochs, our model learns to correctly solve 63.2% of boards at test time, reaching 85% of this theoretical "best." Hence, our approach demonstrates strong performance in solving visual Sudoku boards end-to-end. On the other hand, the baseline convolutional networks make only minuscule improvements to the training loss over the course of 100 epochs, and fail altogether to improve out-of-sample performance. Accordingly, our SATNet architecture enables end-to-end learning of the "rules of the game" directly from pictorial inputs in a way that was not possible with previous architectures.

5. Conclusion

In this paper, we have presented a low-rank differentiable MAXSAT layer that can be integrated into neural network architectures. This layer employs block coordinate descent methods to efficiently compute the forward and backward passes, and is amenable to GPU acceleration. We show that our SATNet architecture can be successfully used to learn logical structures, namely the parity function and the rules of 9×9 Sudoku. We also show, via a visual Sudoku task, that our layer can be integrated into larger deep network architectures for end-to-end training. Our layer thus shows promise in allowing deep networks to learn logical structure *without hard-coding of the relationships between variables*.

More broadly, we believe that this work fills a notable gap in the regime spanning deep learning and logical reasoning. While many "differentiable logical reasoning" systems have been proposed, most of them still require fairly hand-specified logical rules and groundings, and thus are somewhat limited in their ability to operate in a truly end-to-end fashion. Our hope is that by wrapping a powerful yet generic primitive such as MAXSAT solving within a differentiable framework, our solver can enable "implicit" logical reasoning to occur where needed within larger frameworks, even if the precise structure of the domain is unknown and must be learned from data. In other words, we believe that SATNet provides a step towards integrating symbolic reasoning and deep learning, a long-standing goal in artificial intelligence.

Acknowledgments

Po-Wei Wang is supported by a grant from the Bosch Center for AI; Priya Donti is supported by the Department of Energy's Computational Science Graduate Fellowship under grant number DE-FG02-97ER25308; and Bryan Wilder is supported by the National Science Foundation Graduate Research Fellowship.

References

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017.

Barvinok, A. I. Problems of distance geometry and convex properties of quadratic maps. *Discrete & Computational Geometry*, 13(2):189–202, 1995.

Cingillioglu, N. and Russo, A. Deeplogic: End-to-end logical reasoning. *arXiv preprint arXiv:1805.07433*, 2018.

Dai, W.-Z., Xu, Q.-L., Yu, Y., and Zhou, Z.-H. Tunneling neural perception and logic reasoning through abductive learning. *arXiv preprint arXiv:1802.01173*, 2018.

Djolonga, J. and Krause, A. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pp. 1013–1023, 2017.

Donti, P. L., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. *arXiv preprint arXiv:1703.04529*, 2017.

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

Garcez, A., Besold, T. R., De Raedt, L., Földiak, P., Hitzler, P., Icard, T., Kühnberger, K.-U., Lamb, L. C., Miikkulainen, R., and Silver, D. L. Neural-symbolic learning and reasoning: contributions and challenges. In *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches, Stanford*, 2015.

Goemans, M. X. and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

Gomes, C. P., van Hoeve, W.-J., and Leahu, L. The power of semidefinite programming relaxations for max-sat. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pp. 104–118. Springer, 2006.

Hu, Z., Ma, X., Liu, Z., Hovy, E., and Xing, E. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 2410–2420, 2016.

Kingma, D. P. and Ba, J. L. Adam: Amethod for stochastic optimization. In *International Conference on Learning Representations*, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Ling, C. K., Fang, F., and Kolter, J. Z. What game are we playing? end-to-end learning in normal and extensive form games. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 396–402, 2018. doi: 10.24963/ijcai.2018/55.

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

Palm, R. B., Paquet, U., and Winther, O. Recurrent relational networks. *arXiv preprint arXiv:1711.08028*, 2017.

Park, K. Can neural networks crack sudoku?, 2016. URL `https://github.com/Kyubyong/sudoku`.

Pataki, G. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358, 1998.

Selsam, D., Lamm, M., Bunz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Shalev-Shwartz, S., Shamir, O., and Shammah, S. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 3067–3075, 2017.

Sourek, G., Aschenbrenner, V., Zelezny, F., Schockaert, S., and Kuzelka, O. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

Tschiatschek, S., Sahin, A., and Krause, A. Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*, 2018.

Wang, P.-W. and Kolter, J. Z. Low-rank semidefinite programming for the max2sat problem. In *AAAI Conference on Artificial Intelligence*, 2019.

Wang, P.-W., Chang, W.-C., and Kolter, J. Z. The mixing method: coordinate descent for low-rank semidefinite programming. *arXiv preprint arXiv:1706.00476*, 2017.

Wilder, B., Dilkina, B., and Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI Conference on Artificial Intelligence*, 2018.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and den Broeck, G. V. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5498–5507, 2018. URL http://proceedings.mlr.press/v80/xu18h.html.

Yang, F., Yang, Z., and Cohen, W. W. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, pp. 2319–2328, 2017.

A. Derivation of the forward pass coordinate descent update

Our MAXSAT SDP relaxation (described in Section 3.1) is given by

$$\begin{aligned} \underset{V \in \mathbb{R}^{k \times (n+1)}}{\text{minimize}} \quad & \langle S^T S, V^T V \rangle, \\ \text{subject to} \quad & \|v_i\| = 1, \quad i = 0, \ldots, n, \end{aligned} \tag{A.1}$$

where $S \in \mathbb{R}^{m \times (n+1)}$ and v_i is the ith column vector of V.

We rewrite the objective of (A.1) as $\langle S^T S, V^T V \rangle \equiv \operatorname{tr}((S^T S)^T (V^T V)) = \operatorname{tr}(V^T V S^T S)$ by noting that $S^T S$ is symmetric and by cycling matrices within the trace. We then observe that the objective terms that depend on any given v_i are given by

$$v_i^T \sum_{j=0}^{n} s_j^T s_i v_j = v_i^T \sum_{\substack{j=0 \\ (j \neq i)}}^{n} s_j^T s_i v_j + v_i^T s_i^T s_i v_i, \tag{A.2}$$

where s_i is the ith column vector of S. Observe $v_i^T v_i$ in the last term cancels to 1, and the remaining coefficient

$$g_i \equiv \sum_{\substack{j=0 \\ (j \neq i)}}^{n} s_j^T s_i v_j = V S^T s_i - \|s_i\|^2 v_i \tag{A.3}$$

is constant with respect to v_i. Thus, (A.2) can be simply rewritten as

$$v_i^T g_i + s_i^T s_i. \tag{A.4}$$

Minimizing this expression over v_i with respect to the constraint $\|v_i\| = 1$ yields the block coordinate descent update

$$v_i = -g_i / \|g_i\|. \tag{A.5}$$

B. Details on backpropagation through the MAXSAT SDP

Given the result $\partial \ell / \partial V_{\mathcal{O}}$, we next seek to compute $\partial \ell / \partial V_{\mathcal{I}}$ and $\partial \ell / \partial S$ by pushing gradients through the SDP solution procedure described in Section 3.1. We do this by taking the total differential through our coordinate descent updates (3) for each output $o \in \mathcal{O}$ at the optimal fixed-point solution to which these updates converge.

Computing the total differential. Computing the total differential of the updates (3) and rearranging, we see that for every $o \in \mathcal{O}$,

$$\left(\|g_o\| I_k - \|s_o\|^2 P_o\right) \mathsf{d} v_o + P_o \sum_{j \in \mathcal{O}} s_o^T s_j \mathsf{d} v_j = -P_o \xi_o, \tag{B.1}$$

where

$$\xi_o \equiv \Big(\sum_{j \in \mathcal{I}'} s_o^T s_j \mathsf{d} v_j + V \mathsf{d} S^T s_o + V S^T \mathsf{d} s_o - 2 \mathsf{d} s_o^T s_o v_o \Big), \tag{B.2}$$

and where $P_o \equiv I_k - v_o v_o^T, o \in \mathcal{O}$ and $\mathcal{I}' \equiv \{\top\} \cup \mathcal{I}$.

Rewriting as a linear system. Rewriting Equation B.1 over all $o \in \mathcal{O}$ as a linear system, we obtain

$$\begin{aligned} &\Big(\operatorname{diag}(\|g_o\|) \otimes I_k + PC \otimes I_k\Big) \operatorname{vec}(\mathsf{d} V_{\mathcal{O}}) = -P \operatorname{vec}(\xi_o) \\ &\Rightarrow \operatorname{vec}(\mathsf{d} V_{\mathcal{O}}) = -\Big(P\big(\big(\operatorname{diag}(\|g_o\|) + C\big) \otimes I_k\big) P\Big)^{\dagger} \operatorname{vec}(\xi_o), \end{aligned} \tag{B.3}$$

where $C = S_{\mathcal{O}}^T S_{\mathcal{O}} - \operatorname{diag}(\|s_o\|^2)$, $P = \operatorname{diag}(P_o)$, and the second step follows from the lemma presented in Appendix C.

We then see that by the chain rule, the gradients $\partial \ell / \partial V_{\mathcal{I}}$ and $\partial \ell / \partial S$ are given by the left matrix-vector product

$$\begin{aligned} &\left(\frac{\partial \ell}{\partial \operatorname{vec}(V_{\mathcal{O}})}\right)^T \operatorname{vec}(\mathsf{d} V_{\mathcal{O}}) \\ &= -\left(\frac{\partial \ell}{\partial \operatorname{vec}(V_{\mathcal{O}})}\right)^T \Big(P\big(\big(\operatorname{diag}(\|g_o\|) + C\big) \otimes I_k\big) P\Big)^{\dagger} \operatorname{vec}(\xi_o) \end{aligned} \tag{B.4}$$

where the second equality comes from plugging in the result of (B.3).

Now, define $U \in \mathbb{R}^{k \times n}$, where the columns $U_{\mathcal{I}} = 0$ and the columns $U_{\mathcal{O}}$ are given by

$$\operatorname{vec}(U_{\mathcal{O}}) = \Big(P\big(\big(\operatorname{diag}(\|g_o\|) + C\big) \otimes I_k\big) P\Big)^{\dagger} \operatorname{vec}\left(\frac{\partial \ell}{\partial \operatorname{vec}(V_{\mathcal{O}})}\right). \tag{B.5}$$

Then, we see that (B.4) can be written as

$$\left(\frac{\partial \ell}{\partial \operatorname{vec}(V_{\mathcal{O}})}\right)^T \operatorname{vec}(\mathsf{d} V_{\mathcal{O}}) = -\operatorname{vec}(U_{\mathcal{O}})^T \operatorname{vec}(\xi_o), \tag{B.6}$$

which is the implicit linear form for our gradients.

Computing desired gradients from implicit linear form. Once we have obtained $U_{\mathcal{O}}$ (via coordinate descent), we can explicitly compute the desired gradients $\partial \ell / \partial V_{\mathcal{I}}$ and $\partial \ell / \partial S$ from the implicit form (B.6). For instance, to compute the gradient $\partial \ell / \partial v_\iota$ for some $\iota \in \mathcal{I}$, we would set $\mathsf{d} v_\iota = 1$ and all other gradients to zero in Equation (B.6) (where these gradients are captured within the terms ξ_o).

Explicitly, we compute each $\partial \ell / \partial v_{\iota j}$ by setting $\mathsf{d} v_{\iota j} = 1$ and all other gradients to zero, i.e.

$$\begin{aligned} \frac{\partial \ell}{\partial v_{\iota j}} &= -\operatorname{vec}(U_{\mathcal{O}})^T \operatorname{vec}(\xi_o) = -\sum_{o \in \mathcal{O}} u_o^T e_j s_\iota^T s_o \\ &= -e_j^T \left(\sum_{o \in \mathcal{O}} u_o s_o^T\right) s_\iota. \end{aligned} \tag{B.7}$$

Similarly, we compute each $\partial \ell / \partial S_{i,j}$ by setting $\mathsf{d} S_{i,j} = 1$

and all other gradients to zero, i.e.

$$\begin{aligned}\frac{\partial \ell}{\partial S_{i,j}} &= -\sum_{o\in\mathcal{O}} u_o^T \xi_o \\ &= -\sum_{o\in\mathcal{O}} u_o^T v_i s_{oj} - u_i^T (VS^T)_j + u_i^T (s_{ij} P_i v_i) \\ &= -v_i^T (\sum_{o\in\mathcal{O}} u_o s_{oj}) - u_i^T (VS^T)_j.\end{aligned} \tag{B.8}$$

In matrix form, these gradients are

$$\frac{\partial \ell}{\partial V_{\mathcal{I}}} = -\left(\sum_{o\in\mathcal{O}} u_o s_o^T\right) S_{\mathcal{I}}, \tag{B.9}$$

$$\frac{\partial \ell}{\partial S} = -\left(\sum_{o\in\mathcal{O}} u_o s_o^T\right)^T V - (SV^T)U, \tag{B.10}$$

where u_i is the ith column of U, and where $S_{\mathcal{I}}$ denotes the $\mathcal{I}$-indexed column subset of S.

C. Proof of pseudoinverse computations

We prove the following lemma, used to derive the implicit total differential for $\mathrm{vec}(\mathsf{d}V_{\mathcal{O}})$.

Lemma C.1. *The quantity*

$$\mathrm{vec}(\mathsf{d}V_{\mathcal{O}}) = (P\left((D+C)\otimes I_k\right)P)^\dagger\, \mathrm{vec}(\xi_o) \tag{C.1}$$

is the solution of the linear system

$$(D\otimes I_k + PC\otimes I_k)\,\mathrm{vec}(\mathsf{d}V_{\mathcal{O}}) = P\,\mathrm{vec}(\xi_o), \tag{C.2}$$

where $P = \mathrm{diag}(I_k - v_o v_o^T)$, $C = S_{\mathcal{O}}^T S_{\mathcal{O}} - \mathrm{diag}(\|s_o\|^2)$, $D = \mathrm{diag}(\|g_i\|)$, *and* ξ_o *is as defined in Equation* (B.2).

Proof. Examining the equation with respect to $\mathsf{d}v_i$ gives

$$\|g_i\|\mathsf{d}v_i + P_i\left(\sum_j c_{ij}\mathsf{d}v_j - \xi_j\right) = 0, \tag{C.3}$$

which implies that for all i, $\mathsf{d}v_i = P_i y_i$ for some y_i. Substituting y_i into the equality gives

$$(D\otimes I_k + PC\otimes I_k)P\,\mathrm{vec}(y_i) \tag{C.4}$$
$$=P((D+C)\otimes I_k)P\,\mathrm{vec}(y_i) = P\,\mathrm{vec}(\xi_o). \tag{C.5}$$

Note that the last equation comes form $D\otimes I_k P = D\otimes I_k PP = P(D\otimes I_k)P$ due to the block diagonal structure of the projection P. Thus, by the properties of projectors and the pseudoinverse,

$$\mathrm{vec}(Y) = (P((D+C)\otimes I_k)P)^\dagger P\,\mathrm{vec}(\xi_o) \tag{C.6}$$
$$= (P((D+C)\otimes I_k)P)^\dagger\,\mathrm{vec}(\xi_o). \tag{C.7}$$

Note that the first equation comes from the idempotence property of P (that is, $PP = P$). Substituting $\mathrm{vec}(\mathsf{d}V_{\mathcal{O}}) = P\,\mathrm{vec}(Y)$ back gives the solution of $\mathsf{d}V_{\mathcal{O}}$. □

D. Derivation of the backward pass coordinate descent algorithm

Consider solving for $U_{\mathcal{O}}$ as mentioned in Equation (B.5):

$$\Big(P\big(\big(\mathrm{diag}(\|g_o\|)+C\big)\otimes I_k\big)P\Big)\,\mathrm{vec}(U_{\mathcal{O}}) = \mathrm{vec}\left(\frac{\partial \ell}{\partial\,\mathrm{vec}(V_{\mathcal{O}})}\right),$$

where $C = S_{\mathcal{O}}^T S_{\mathcal{O}} - \mathrm{diag}(\|s_o\|^2)$. The linear system can be computed using block coordinate descent. Specifically, observe this linear system with respect to only the u_o variable. Since we start from $U_{\mathcal{O}} = 0$, we can assume that $P\,\mathrm{vec}(U_o) = \mathrm{vec}(U_o)$. This yields

$$\|g_o\|P_o u_o + P_o\Big(U_{\mathcal{O}} S_{\mathcal{O}}^T s_o - \|s_o\|^2 u_o\Big) = P_o\left(\frac{\partial \ell}{\partial v_o}\right). \tag{D.1}$$

Let $\Psi = (U_{\mathcal{O}})S_{\mathcal{O}}^T$. Then we have

$$\|g_o\|P_o u_o = -P_o(\Psi s_o - \|s_o\|^2 u_o - \partial\ell/\partial v_o). \tag{D.2}$$

Define $-\mathsf{d}g_i$ to be the terms contained in parentheses in the right-hand side of the above equation. Note that $\mathsf{d}g_i$ does not depend on the variable u_o. Thus, we have the closed-form feasible solution

$$u_o = -P_o \mathsf{d}g_o/\|g_o\|. \tag{D.3}$$

After updating u_o, we can maintain the term Ψ by replacing the old u_o^{prev} with the new u_o. This yields the rank 1 update

$$\Psi := \Psi + (u_o - u_o^{\text{prev}})s_o^T. \tag{D.4}$$

The above procedure is summarized in Algorithm 3. Further, we can verify that the assumption $P\,\mathrm{vec}(U_{\mathcal{O}}) = \mathrm{vec}(U_{\mathcal{O}})$ still holds after each update by the projection P_o.

E. Results for the 4×4 Sudoku problem

We compare the performance of our SATNet architecture on a 4×4 reduced version of the Sudoku puzzle against OptNet (Amos & Kolter, 2017) and a convolutional neural network architecture. These results (over 9K training and 1K testing examples) are shown in Figure E.1. We note that our architecture converges quickly – in just two epochs – to *100% board-wise test accuracy*.

OptNet takes slightly longer to converge to similar performance, in terms of both time and epochs. In particular, we see that OptNet takes 3-4 epochs to converge (as opposed to 1 epoch for SATNet). Further, in our preliminary benchmarks, OptNet required 12 minutes to run 20 epochs on a GTX 1080 Ti GPU, whereas SATNet took only 2 minutes to run the same number of epochs. In other words, we see that SATNet requires fewer epochs to converge *and* takes less time per epoch than OptNet.

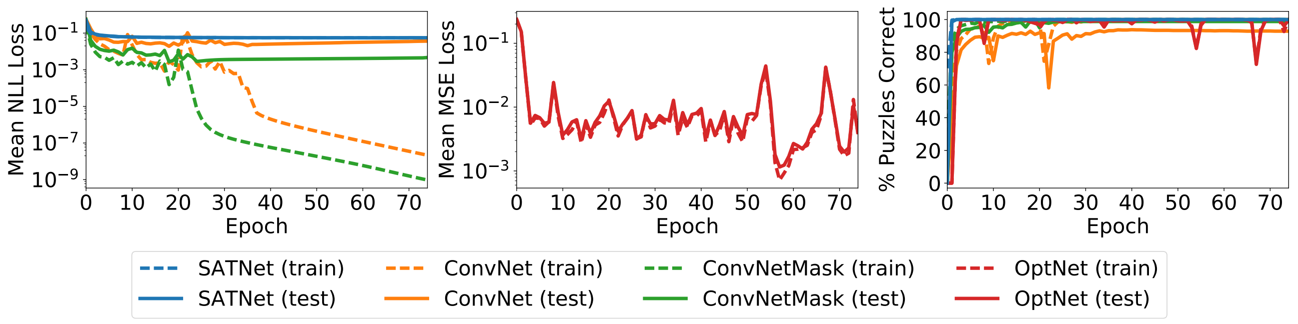


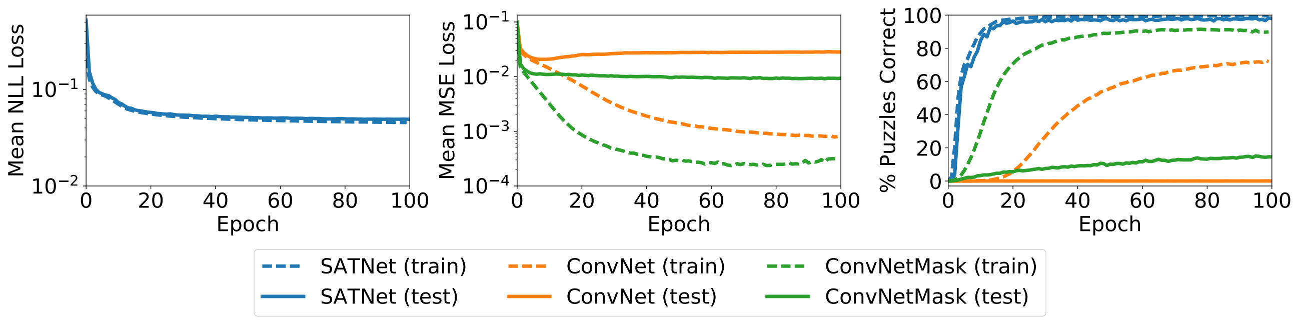
Figure E.1. Results for 4×4 Sudoku. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better.

Both our SATNet architecture and OptNet outperform the traditional convolutional neural network in this setting, as the ConvNet somewhat overfits to the training set and therefore does not generalize as well to the test set (achieving 93% accuracy). The ConvNetMask, which additionally receives a binary input mask, performs much better (99% test accuracy) but does not achieve perfect performance as in the case of OptNet and SATNet.

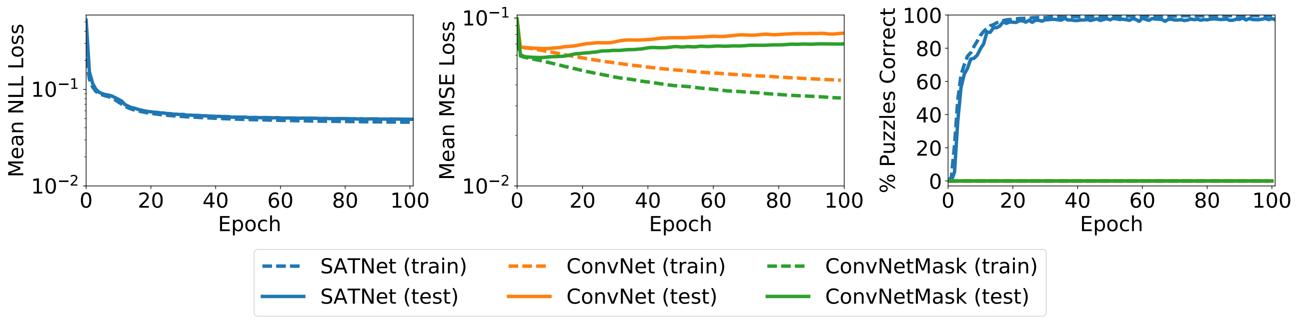
F. Convergence plots for 9×9 Sudoku experiments

Convergence plots for our 9×9 Sudoku experiments (original and permuted) are shown in Figure F.1. SATNet performs nearly identically in both the original and permuted settings, generalizing well to the test set at every epoch without overfitting to the training set. The ConvNet and ConvNetMask, on the other hand, do not generalize well. In the original setting, both architectures overfit to the training set, showing little-to-no improvement in generalization performance over the course of training. In the permuted setting, both ConvNet and ConvNetMask make little progress even on the training set, as they are not able to rely on spatial locality of inputs.

Convergence plots for the visual Sudoku experiments are shown in Figure F.2. Here, we see that SATNet generalizes well in terms of loss throughout the training process, and generalizes somewhat well in terms of whole-board accuracy. The difference in generalization performance between the logical and visual Sudoku settings can be attributed to the generalization performance of the MNIST classifier trained end-to-end with our SATNet layer. The ConvNetMask architecture overfits to the training set, and the ConvNet architecture makes little-to-no progress even on the training set.



(a) Original 9×9 Sudoku



(b) Permuted 9×9 Sudoku

Figure F.1. Results for our 9×9 Sudoku experiments. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better.

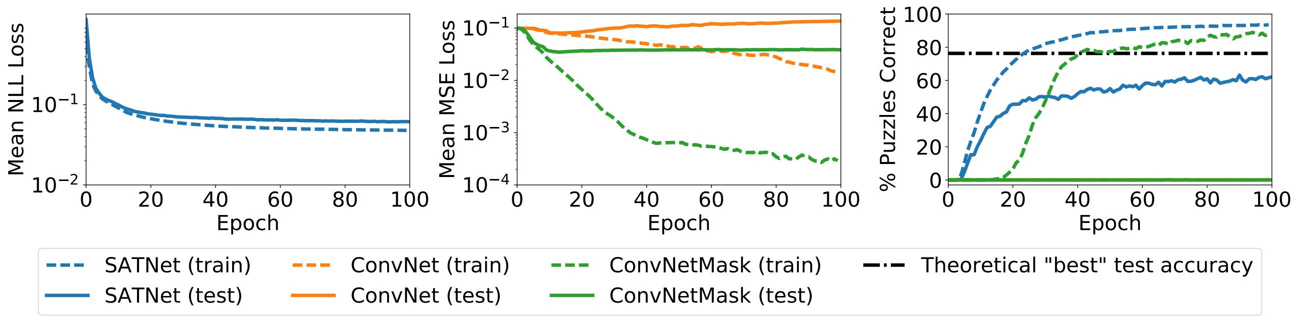


Figure F.2. Results for our visual Sudoku experiments. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better. The theoretical "best" test accuracy plotted is for our specific choice of MNIST classifier architecture.