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*

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

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

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

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.

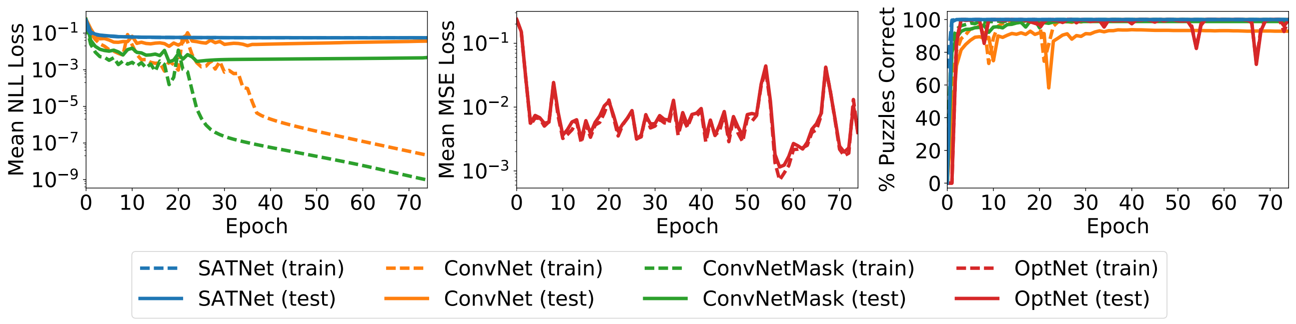


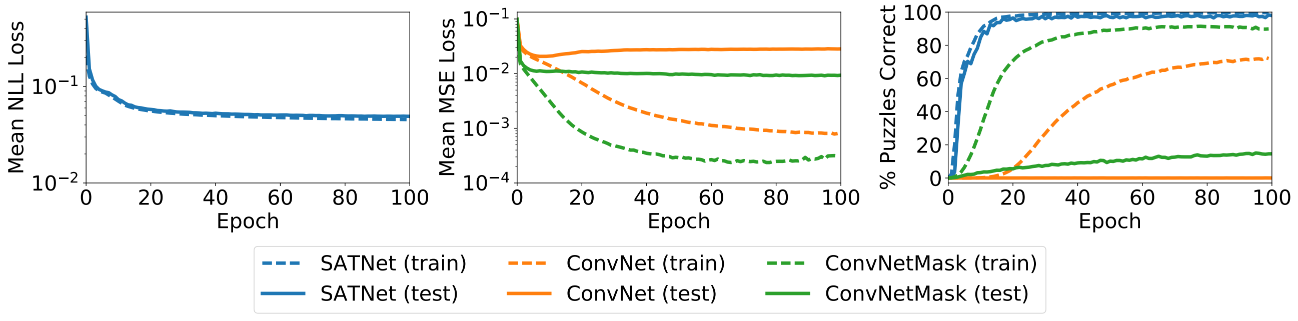
Figure E.1. Results for 4×4 Sudoku. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better.

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.

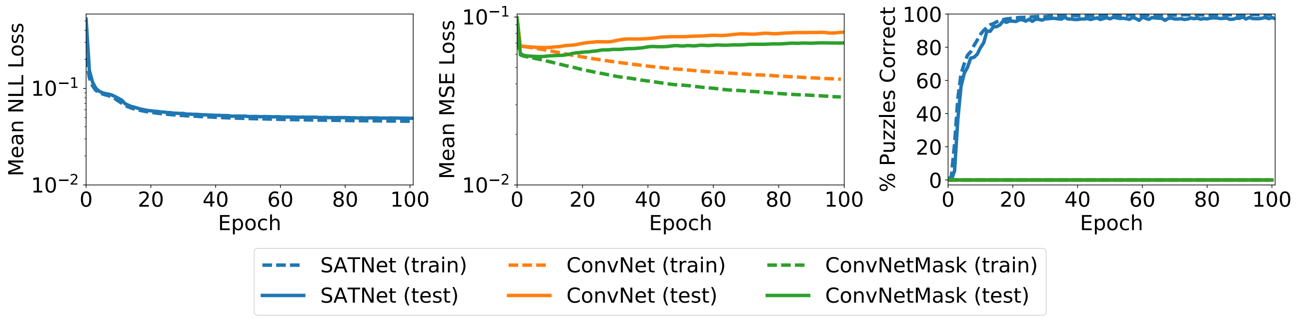
F. Convergence plots for 9×9 Sudoku experiments

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.

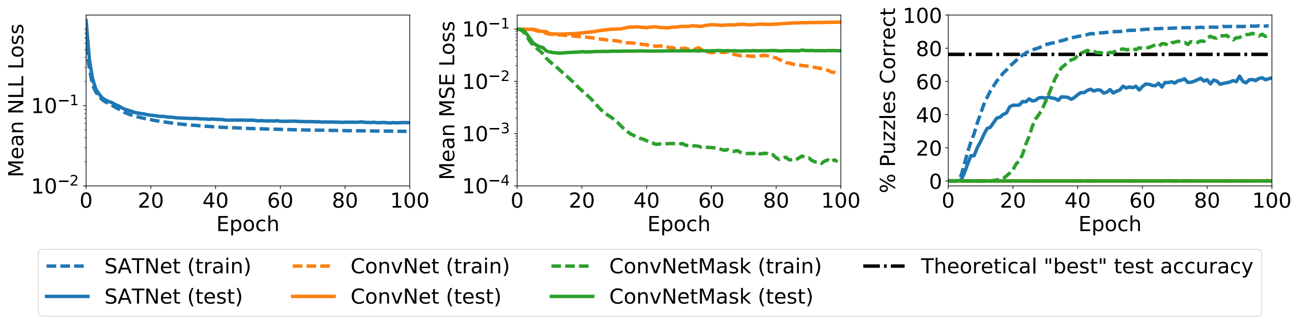


Figure F.2. Results for our visual Sudoku experiments. Lower loss (mean NLL loss and mean MSE loss) and higher whole-board accuracy (% puzzles correct) are better. The theoretical "best" test accuracy plotted is for our specific choice of MNIST classifier architecture.