

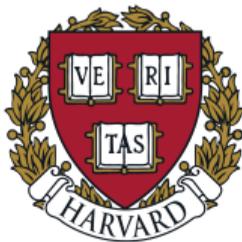
SATNet:

Bridging deep learning and logical reasoning using a differentiable satisfiability solver

Po-Wei Wang ¹ Priya L. Donti ¹ Bryan Wilder ² J. Zico Kolter ^{1,3}



¹ School of Computer Science,
Carnegie Mellon University

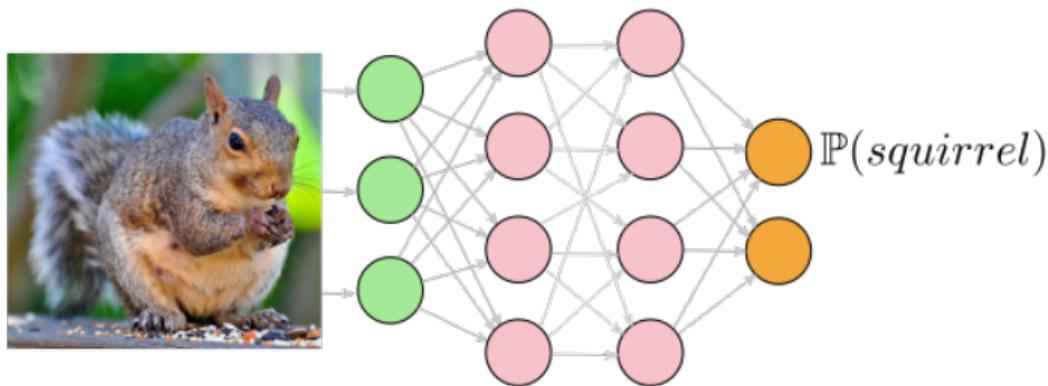


² School of Engineering and Applied Sciences,
Harvard University



³ Bosch Center for Artificial Intelligence

Integrating deep learning and logic



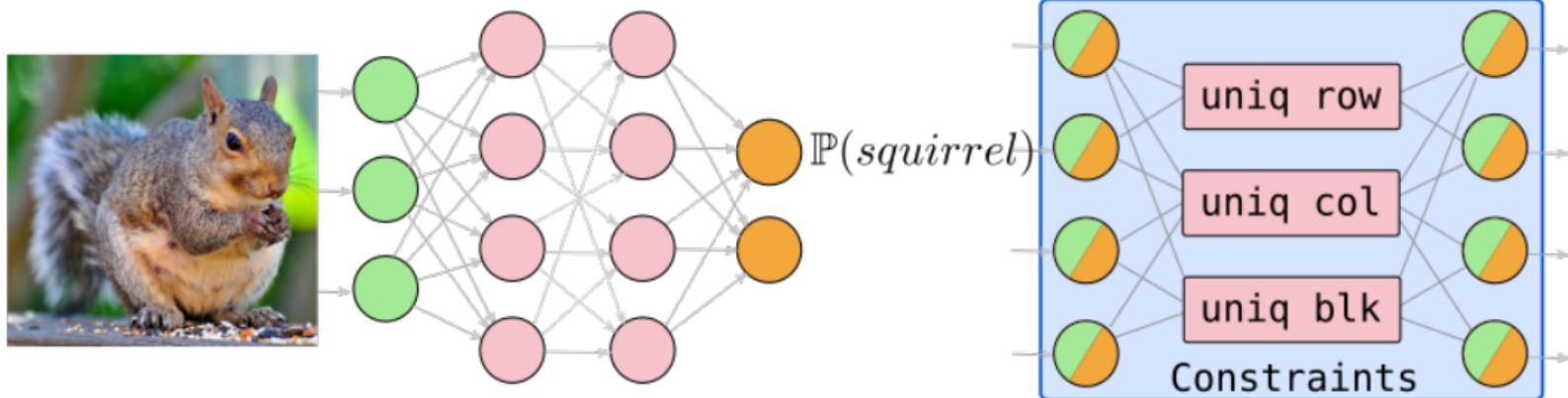
Deep Learning

No constraints on output

Differentiable

Solved via gradient optimizers

Integrating deep learning and logic



Deep Learning

No constraints on output

Differentiable

Solved via gradient optimizers

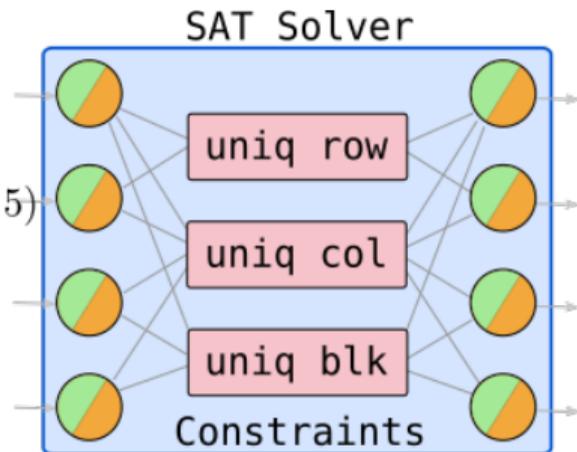
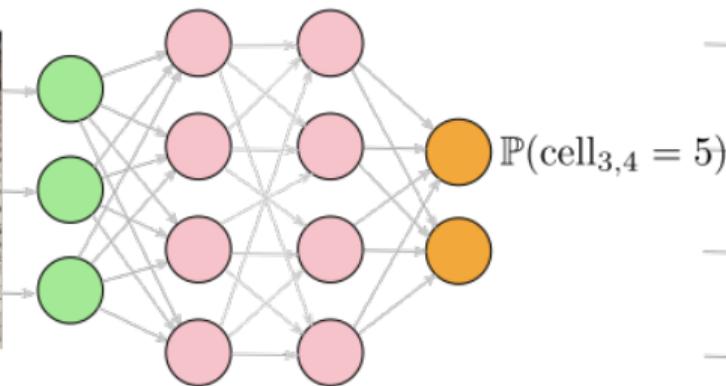
Logical Inference

Rich constraints on output

Discrete input/output

Solved via tree search

Integrating deep learning and logic



Deep Learning

No constraints on output

Differentiable

Solved via gradient optimizers



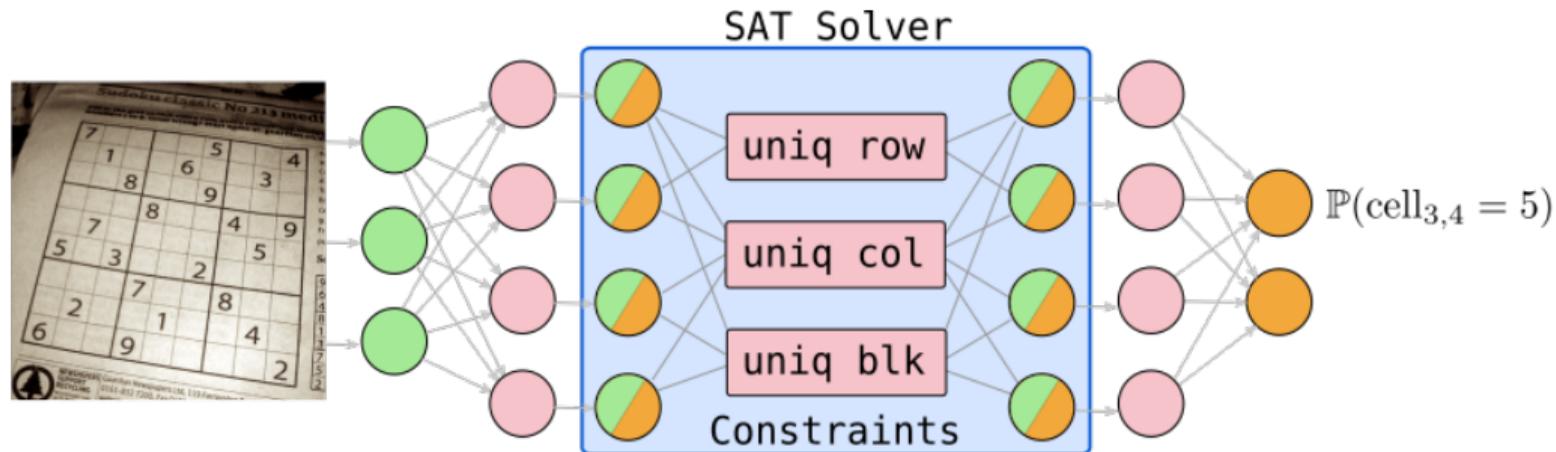
Logical Inference

Rich constraints on output

Discrete input/output

Solved via tree search

Integrating deep learning and logic



Deep Learning

No constraints on output

Differentiable

Solved via gradient optimizers

+

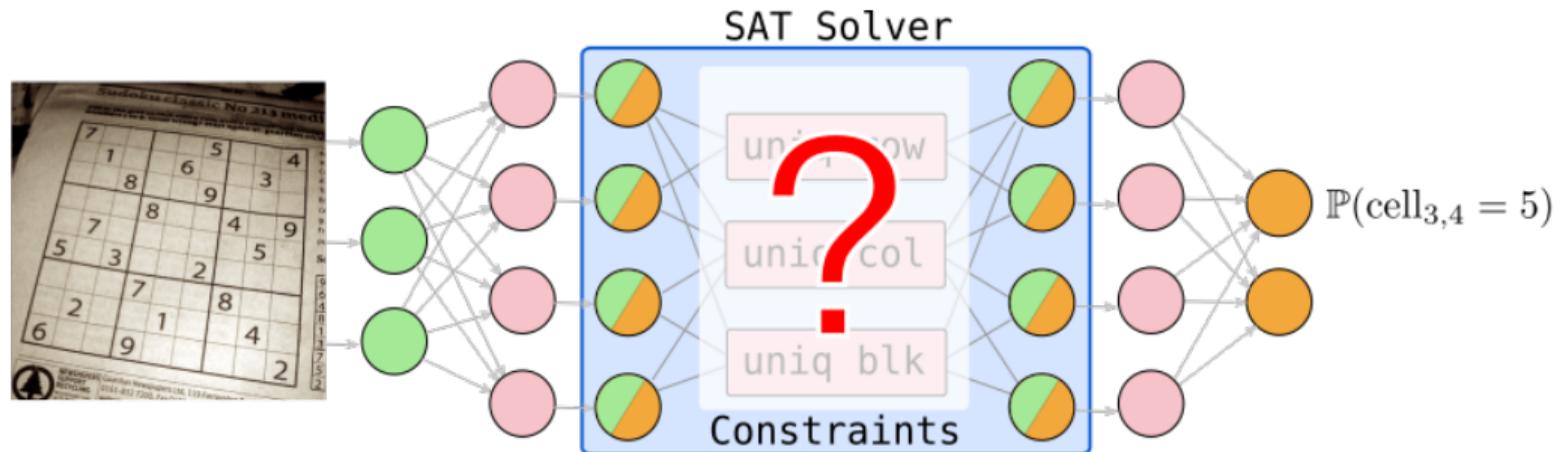
Logical Inference

Rich constraints on output

Discrete input/output

Solved via tree search

Integrating deep learning and logic



Deep Learning

No constraints on output

Differentiable

Solved via gradient optimizers

+

Logical Inference

Rich constraints on output

Discrete input/output

Solved via tree search

This talk is not about . . .

This talk is not about . . .

Not about learning to find SAT solutions [Selsam et al. 2019]

- but about learning both **constraints** and **solution** from examples

This talk is not about . . .

Not about learning to find SAT solutions [Selsam et al. 2019]

- but about learning both **constraints** and **solution** from examples

Not about using DL and SAT in a multi-staged manner

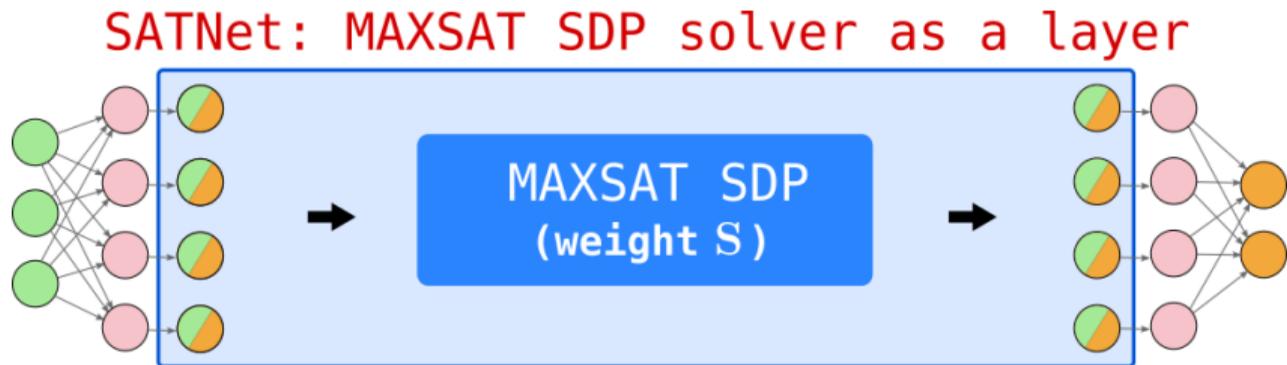
- doing so requires prior knowledge on the structure and constraints
- further, current SAT solvers cannot accept **probability inputs**

This talk is about

- A layer that enables end-to-end learning of **both** the **constraints** and **solutions** of logic problems within deep networks...

This talk is about

- A layer that enables end-to-end learning of **both** the **constraints** and **solutions** of logic problems within deep networks...
- A smoothed **differentiable (maximum) satisfiability solver** that can be integrated into the loop of deep learning systems.



Review of SAT problems

Example SAT problem:

$$v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$$

Review of SAT problems

Example SAT problem:

$$v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$$

\Downarrow

$$S = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{matrix} v_2 \\ v_1 \vee \neg v_2 \\ v_2 \vee \neg v_3 \end{matrix}$$

Review of SAT problems

Example SAT problem:

$$v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$$

\Downarrow

$$S = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{matrix} v_2 \\ v_1 \vee \neg v_2 \\ v_2 \vee \neg v_3 \end{matrix}$$

Typical SAT: Clause matrix given, find satisfying assignment

Our setting: Clause matrix is parameters of the layer (to be learned)

MAXSAT Problem

MAXSAT is the optimization variant of SAT solving

SAT: Find feasible v_i s.t. $v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$

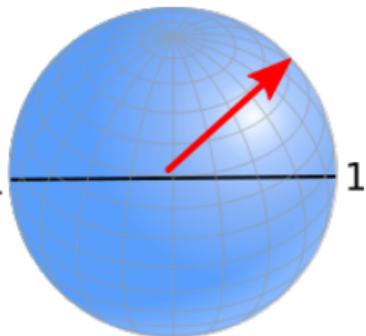
MAXSAT: maximize # of satisfiable clauses

MAXSAT Problem

MAXSAT is the optimization variant of SAT solving

SAT: Find feasible v_i s.t. $v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$ -1 1

MAXSAT: maximize # of satisfiable clauses



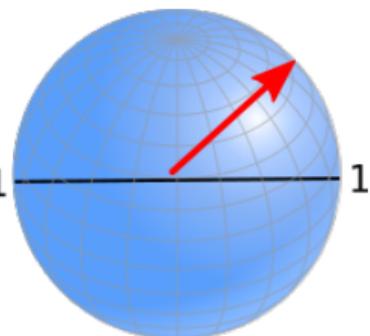
Relax the binary variables to smooth & continuous spheres

$$v_i \in \{+1, -1\} \xrightarrow{\text{equiv}} |v_i| = 1, v_i \in \mathbb{R}^1 \xrightarrow{\text{relax}} \|v_i\| = 1, v_i \in \mathbb{R}^k$$

MAXSAT Problem

MAXSAT is the optimization variant of SAT solving

SAT: Find feasible v_i s.t. $v_2 \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee \neg v_3)$ -1 1



MAXSAT: maximize # of satisfiable clauses

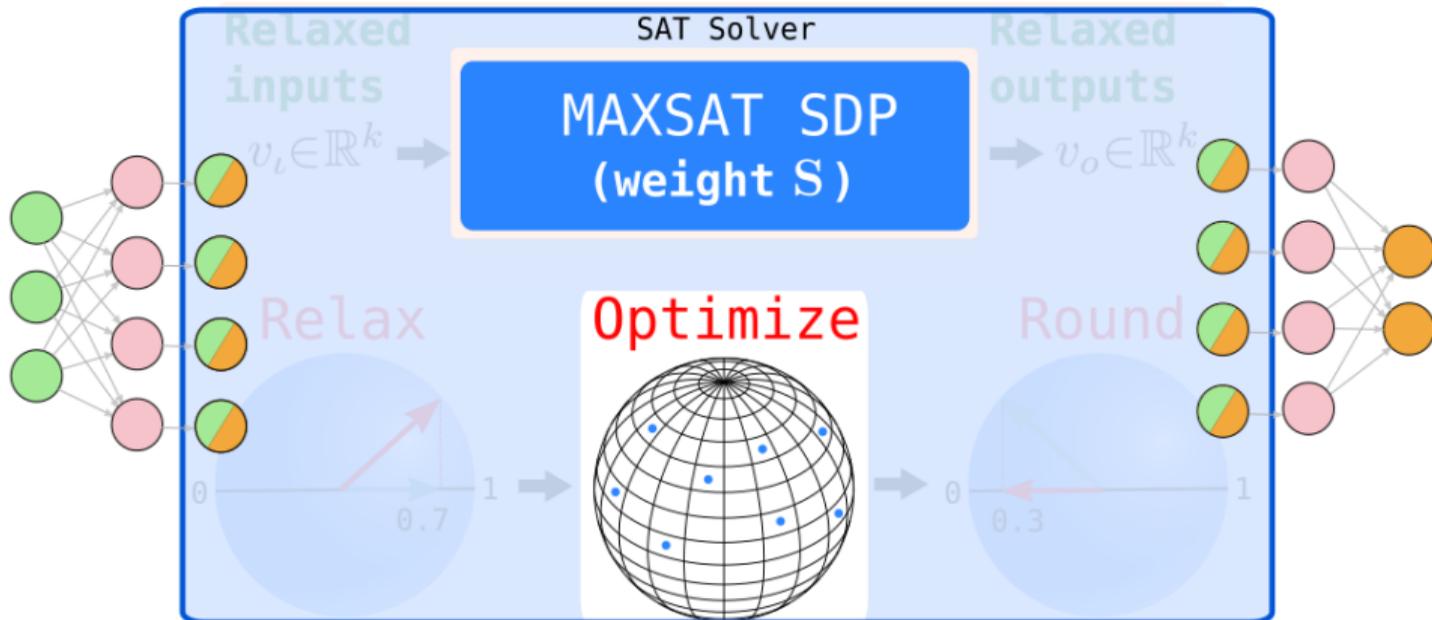
Relax the binary variables to smooth & continuous spheres

$$v_i \in \{+1, -1\} \xrightarrow{\text{equiv}} |v_i| = 1, \quad v_i \in \mathbb{R}^1 \xrightarrow{\text{relax}} \|v_i\| = 1, \quad v_i \in \mathbb{R}^k$$

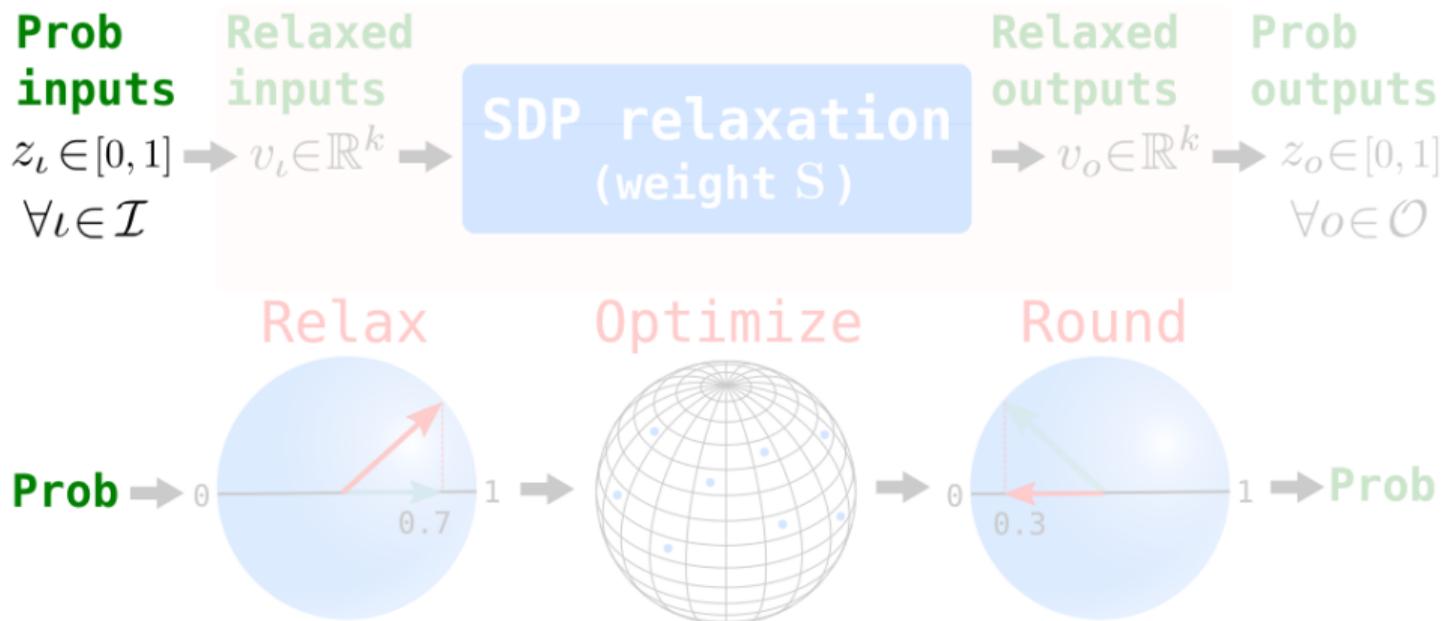
Semidefinite relaxation (Goemans-Williamson, 1995), $X = V^T V$

$$\text{minimize } \langle S^T S, X \rangle, \quad \text{s.t. } X \succeq 0, \quad \text{diag}(X) = 1.$$

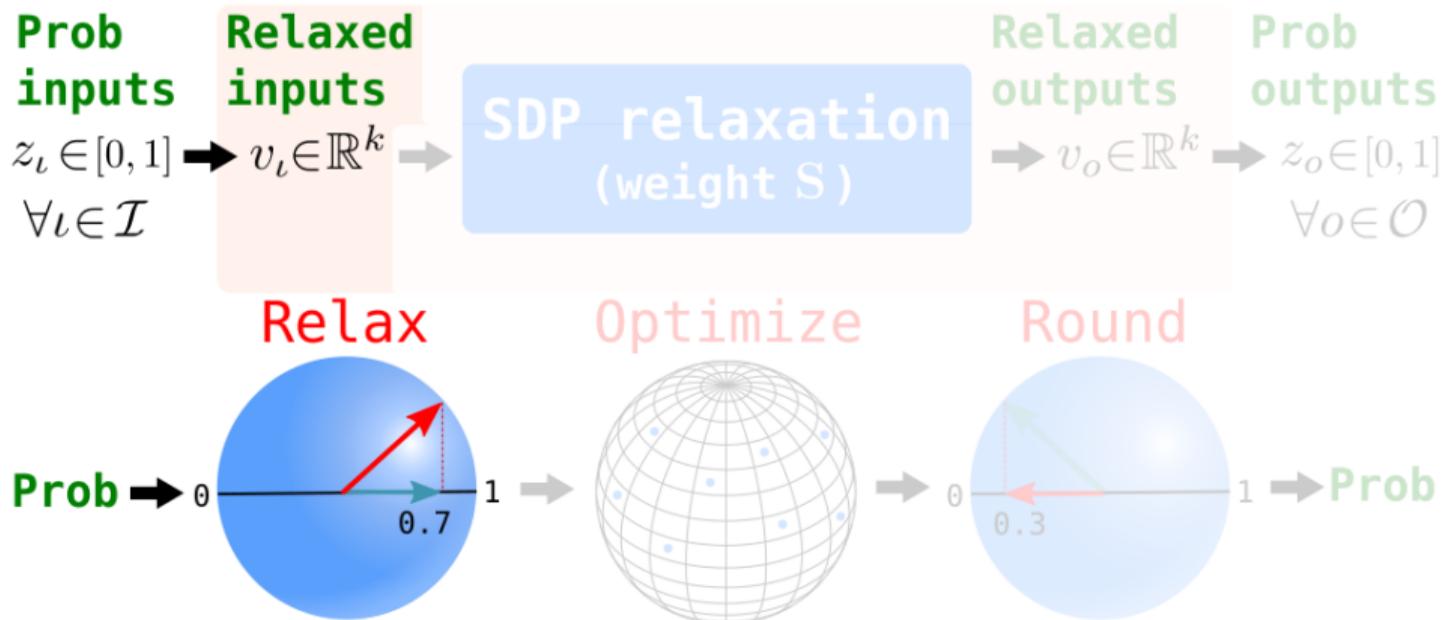
SATNet: MAXSAT SDP as a layer



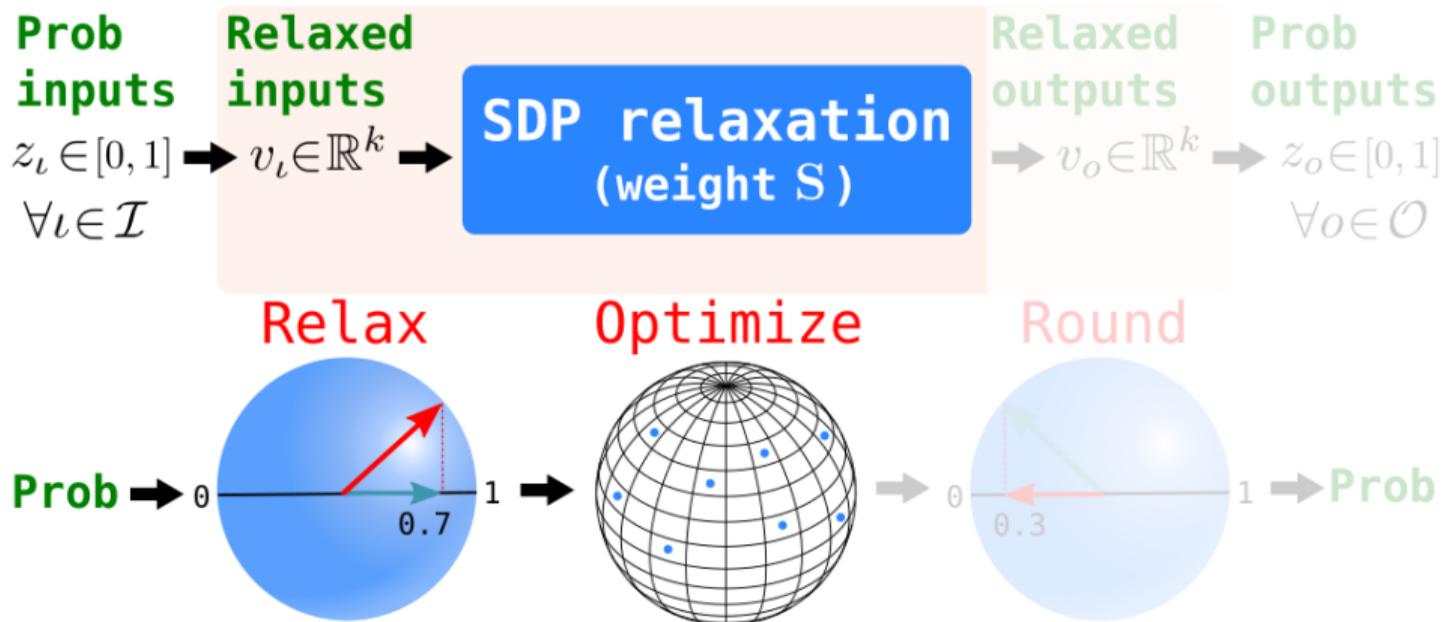
SATNet: MAXSAT SDP as a layer



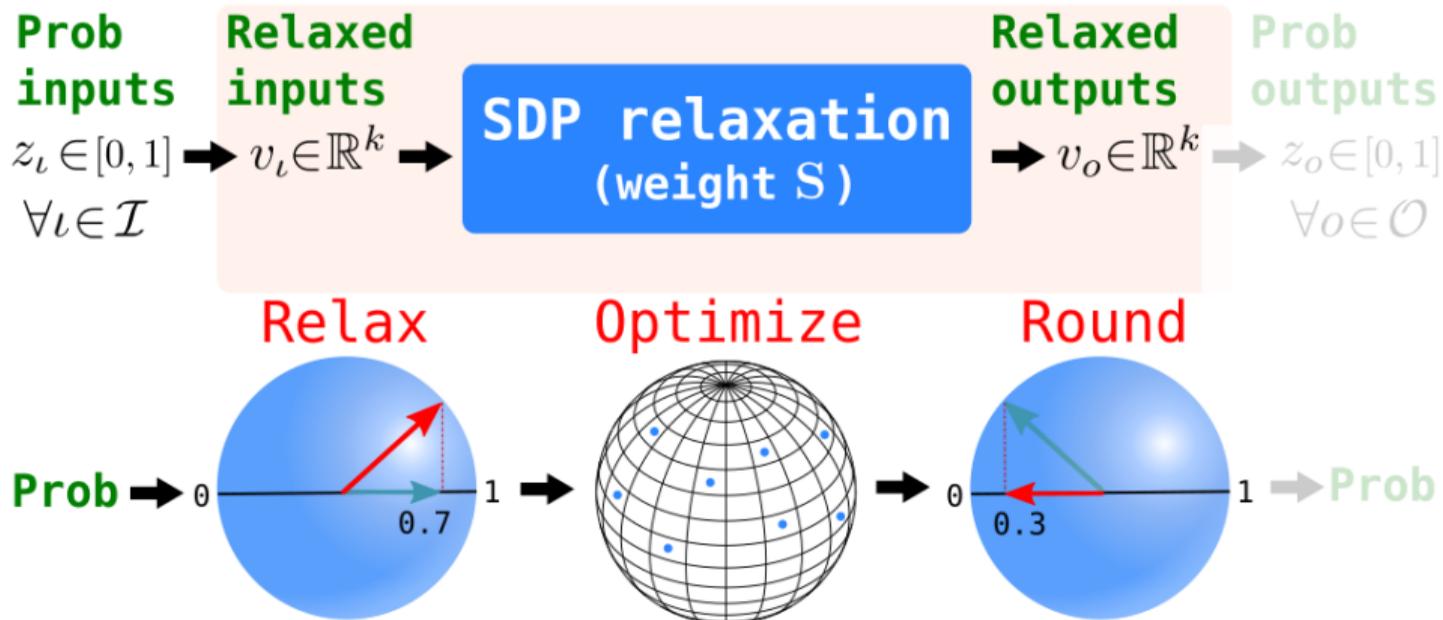
SATNet: MAXSAT SDP as a layer



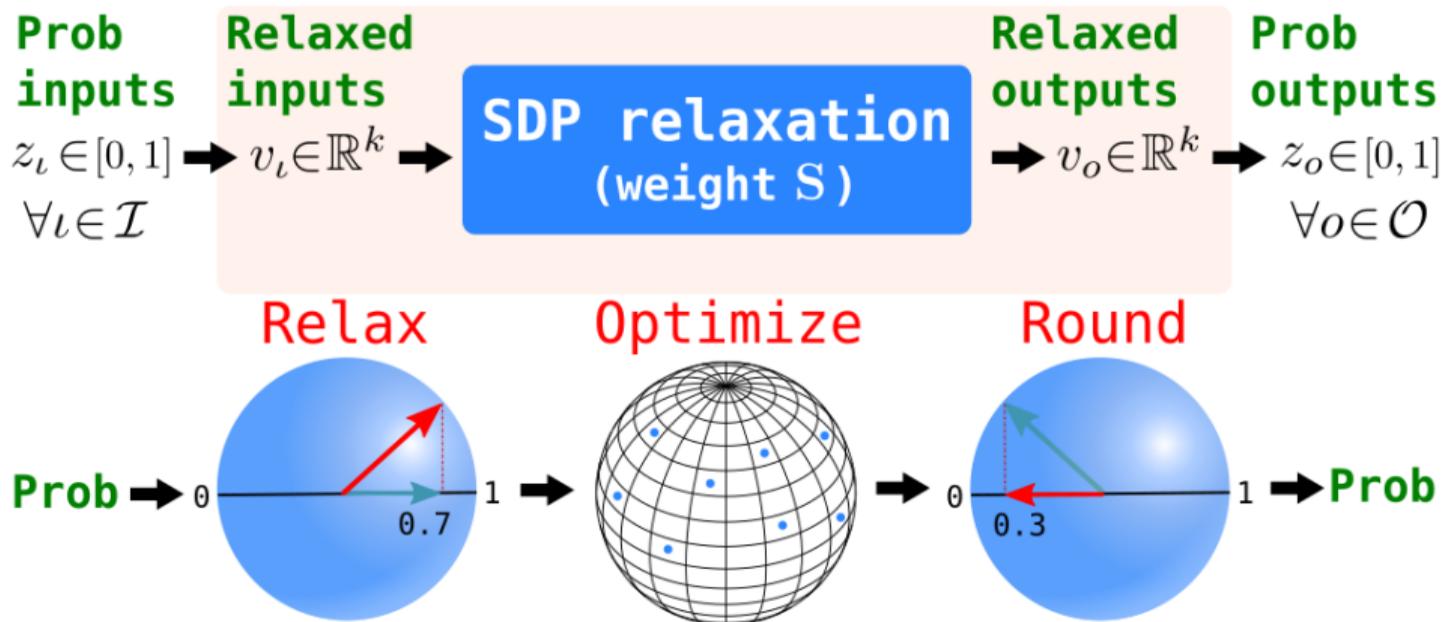
SATNet: MAXSAT SDP as a layer



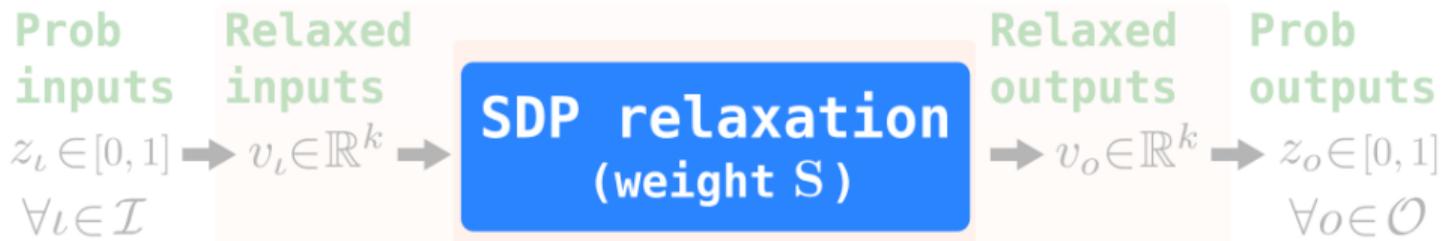
SATNet: MAXSAT SDP as a layer



SATNet: MAXSAT SDP as a layer



SATNet: MAXSAT SDP as a layer

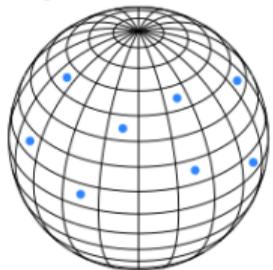


Relax

Optimize

Complexity:

Primal-dual interior point method for SDP



$O(n^6)$

Too expensive

Fast solution to MAXSAT SDP approximation

Efficiently solve via low-rank factorization $X = V^T V$, $V \in \mathbb{R}^{k \times n}$, $\|v_i\| = 1$ (a.k.a. Burer-Monteiro method), and block coordinate descent iters

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i).$$

Fast solution to MAXSAT SDP approximation

Efficiently solve via low-rank factorization $X = V^T V$, $V \in \mathbb{R}^{k \times n}$, $\|v_i\| = 1$ (a.k.a. Burer-Monteiro method), and block coordinate descent iters

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i).$$

For $k > \sqrt{2n}$, the non-convex iterates are guaranteed to converge to global optima of SDP [Wang et al., 2018; Erdogdu et al., 2018]

Fast solution to MAXSAT SDP approximation

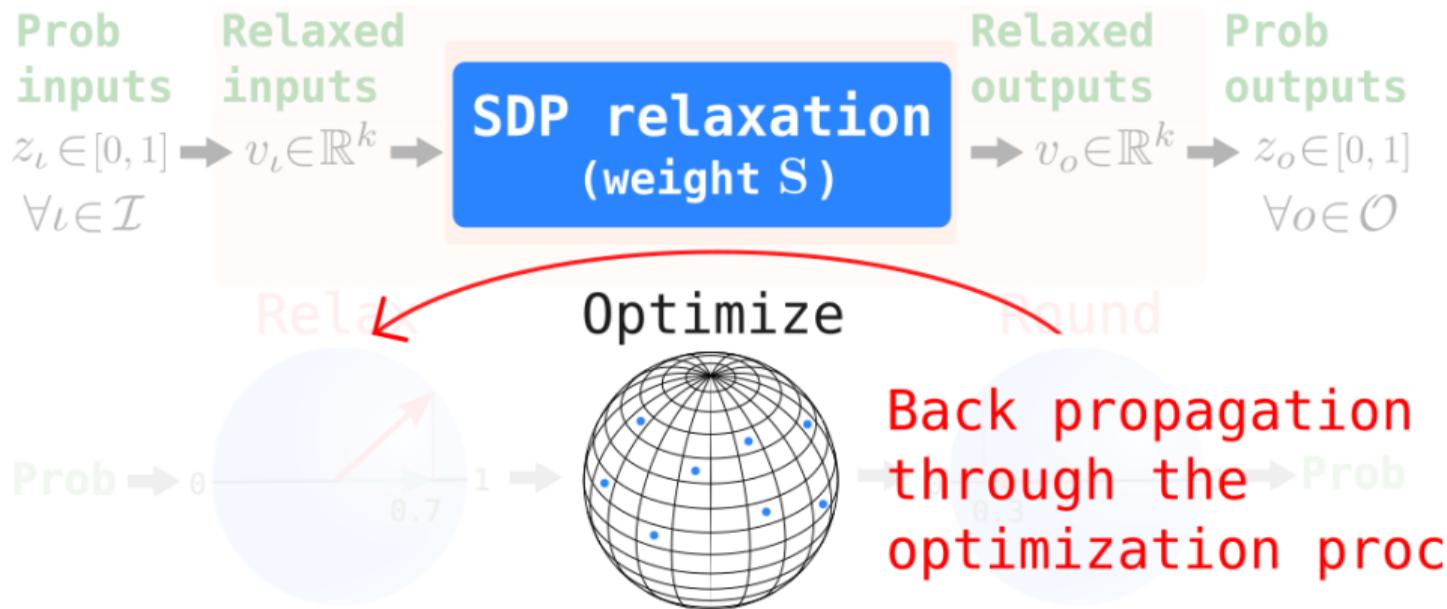
Efficiently solve via low-rank factorization $X = V^T V$, $V \in \mathbb{R}^{k \times n}$, $\|v_i\| = 1$ (a.k.a. Burer-Monteiro method), and block coordinate descent iters

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i).$$

For $k > \sqrt{2n}$, the non-convex iterates are guaranteed to converge to global optima of SDP [Wang et al., 2018; Erdogdu et al., 2018]

Complexity reduced from $O(n^6 \log \log \frac{1}{\epsilon})$ of interior point methods to $O(n^{1.5} m \log \frac{1}{\epsilon})$ of our method, where m is #clauses.

Differentiate through the optimization problem



Differentiate through the optimization problem

When converged, the procedure satisfies the fixed-point equation

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i), \forall i$$

Differentiate through the optimization problem

When converged, the procedure satisfies the fixed-point equation

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i), \forall i$$

The fixed-point equation of the block coordinate descent provides an implicit function definition of the solution [Amos et al. 2017]

$$F_i(S, V(S)) = v_i + \text{normalize}(VS^T s_i - \|s_i\|^2 v_i) = 0, \forall i$$

Differentiate through the optimization problem

When converged, the procedure satisfies the fixed-point equation

$$v_i = -\text{normalize}(VS^T s_i - \|s_i\|^2 v_i), \forall i$$

The fixed-point equation of the block coordinate descent provides an implicit function definition of the solution [Amos et al. 2017]

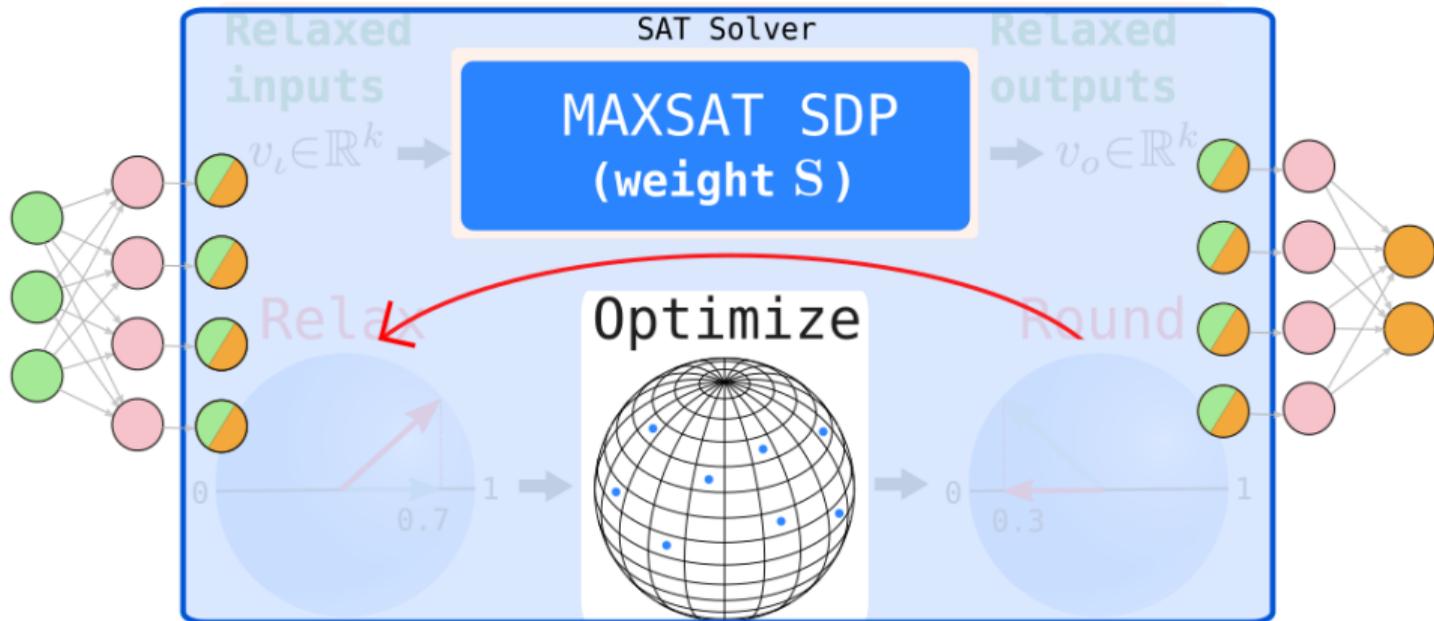
$$F_i(S, V(S)) = v_i + \text{normalize}(VS^T s_i - \|s_i\|^2 v_i) = 0, \forall i$$

Thus, can apply implicit function theorem on the total derivatives

$$\frac{\partial \vec{F}(\vec{S}, \vec{V}(S))}{\partial \vec{S}} = 0 \implies \frac{\partial \vec{F}(\vec{S}, \vec{V})}{\partial \vec{S}} + \frac{\partial \vec{F}(\vec{S}, \vec{V})}{\partial \vec{V}} \cdot \frac{\partial \vec{V}}{\partial \vec{S}} = 0$$

Solve the above **linear system** of $\partial \vec{V} / \partial \vec{S}$ to backprop

SATNet: MAXSAT SDP as a layer



Other ingredients in SATNet

Low-rank regularization on S

- Doubly-exponentially many possible Boolean functions!

Other ingredients in SATNet

Low-rank regularization on S

- Doubly-exponentially many possible Boolean functions!
- Low-rank \Rightarrow Regularize the complexity through number of clauses

Other ingredients in SATNet

Low-rank regularization on S

- Doubly-exponentially many possible Boolean functions!
- Low-rank \Rightarrow Regularize the complexity through number of clauses

Auxiliary variable (hidden nodes)

- Only SDP with diagonal constraints, limiting representation

Other ingredients in SATNet

Low-rank regularization on S

- Doubly-exponentially many possible Boolean functions!
- Low-rank \Rightarrow Regularize the complexity through number of clauses

Auxiliary variable (hidden nodes)

- Only SDP with diagonal constraints, limiting representation
- Adding auxiliary variable (gadget) increases representation power

Illustration: Learning Parity from single bit supervision

- Parity problem is surprisingly hard for most deep networks to learn [Shalev-Swartz et al., 2017]

Illustration: Learning Parity from single bit supervision

- Parity problem is surprisingly hard for most deep networks to learn [Shalev-Swartz et al., 2017]
- Chained (recurrent) SATNet-based network learns parity function for up to length 40 strings from 10K examples

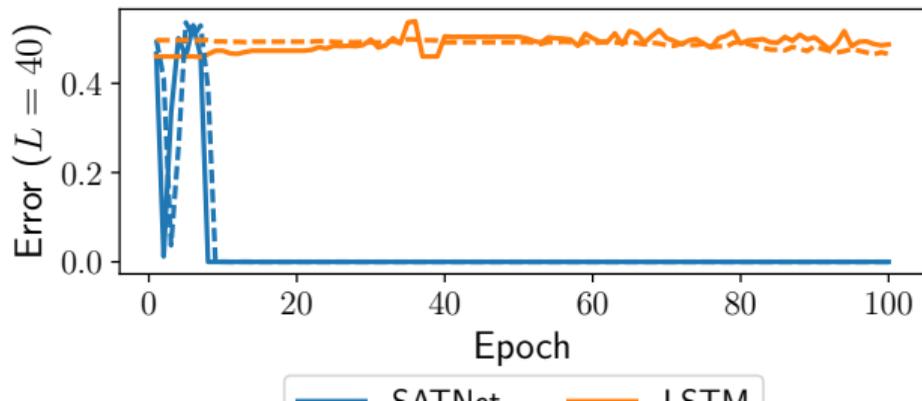


Illustration: Learning Sudoku

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 3 | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | |
| | 9 | 8 | | | | 6 | |
| 8 | | | 6 | | | | 3 |
| 4 | | 8 | | 3 | | | 1 |
| 7 | | | 2 | | | | 6 |
| | 6 | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | 5 |
| | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

Illustration: Learning Sudoku

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

- Learning 9x9 Sudoku from 9K examples
- Single SATNet layer on one-hot-encoded input puzzles

| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 72.6% | 0.04% |
| SATNet (ours) | 99.8% | 98.3% |

Original Sudoku.

Illustration: Learning Sudoku

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

- Learning 9x9 Sudoku from 9K examples
- Single SATNet layer on one-hot-encoded input puzzles
- Free parameters are S matrix of clauses, randomly initialized

| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 72.6% | 0.04% |
| SATNet (ours) | 99.8% | 98.3% |

Original Sudoku.

Illustration: Learning Sudoku

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

- Learning 9x9 Sudoku from 9K examples
- Single SATNet layer on one-hot-encoded input puzzles
- Free parameters are S matrix of clauses, randomly initialized

| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 72.6% | 0.04% |
| SATNet (ours) | 99.8% | 98.3% |

Original Sudoku.

| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 0% | 0% |
| SATNet (ours) | 99.7% | 98.3% |

Permuted Sudoku.

Illustration: MNIST Sudoku

| | | |
|-------|-------|-------|
| 0 6 2 | 1 0 7 | 0 8 0 |
| 0 3 0 | 0 0 8 | 2 5 0 |
| 8 0 0 | 0 0 4 | 0 0 0 |
| 0 0 0 | 0 8 0 | 7 0 0 |
| 4 9 1 | 0 6 0 | 0 2 8 |
| 5 0 0 | 3 4 0 | 1 0 0 |
| 0 0 3 | 0 7 9 | 0 1 0 |
| 1 7 0 | 0 0 0 | 5 0 0 |
| 0 5 0 | 0 0 0 | 9 6 0 |

Illustration: MNIST Sudoku

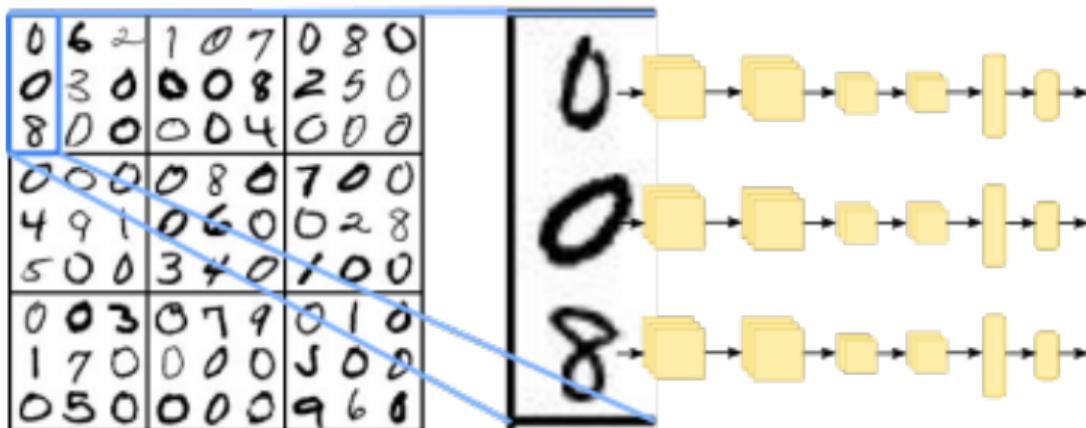


Illustration: MNIST Sudoku

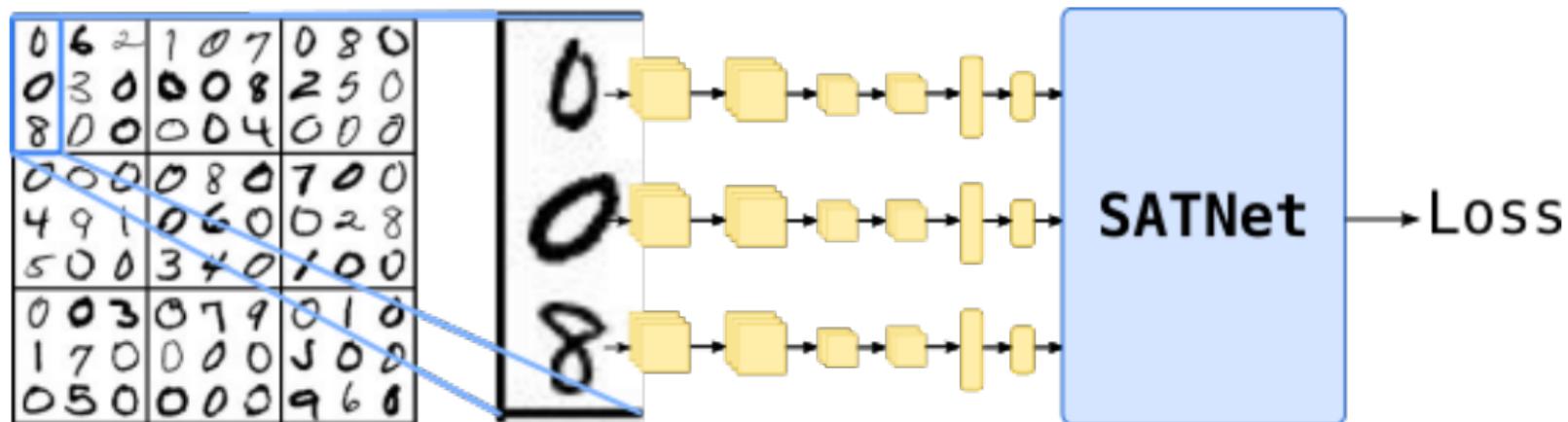
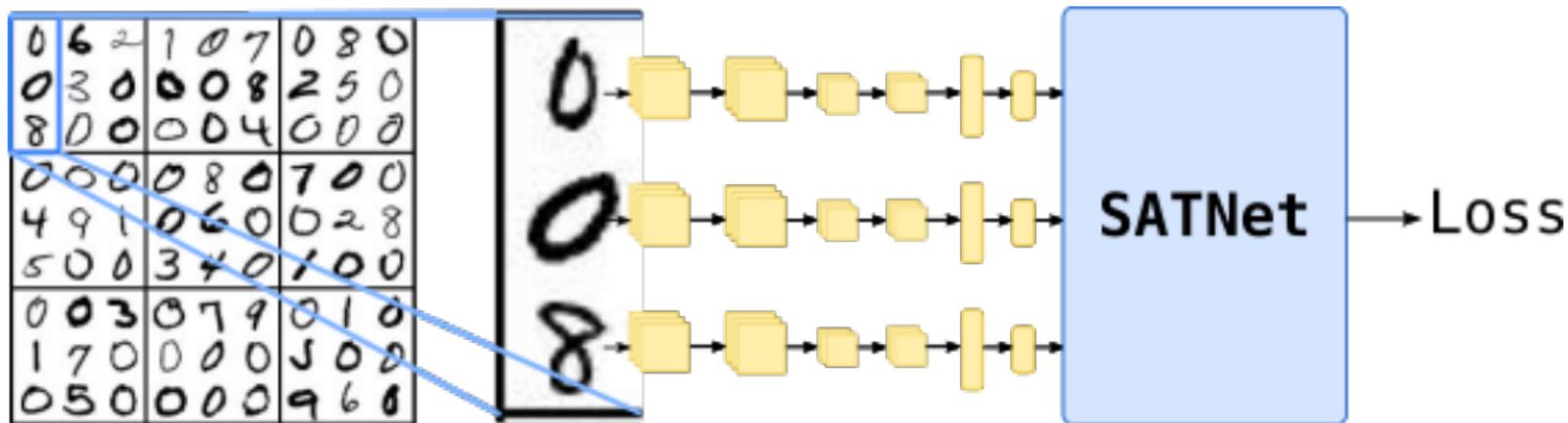


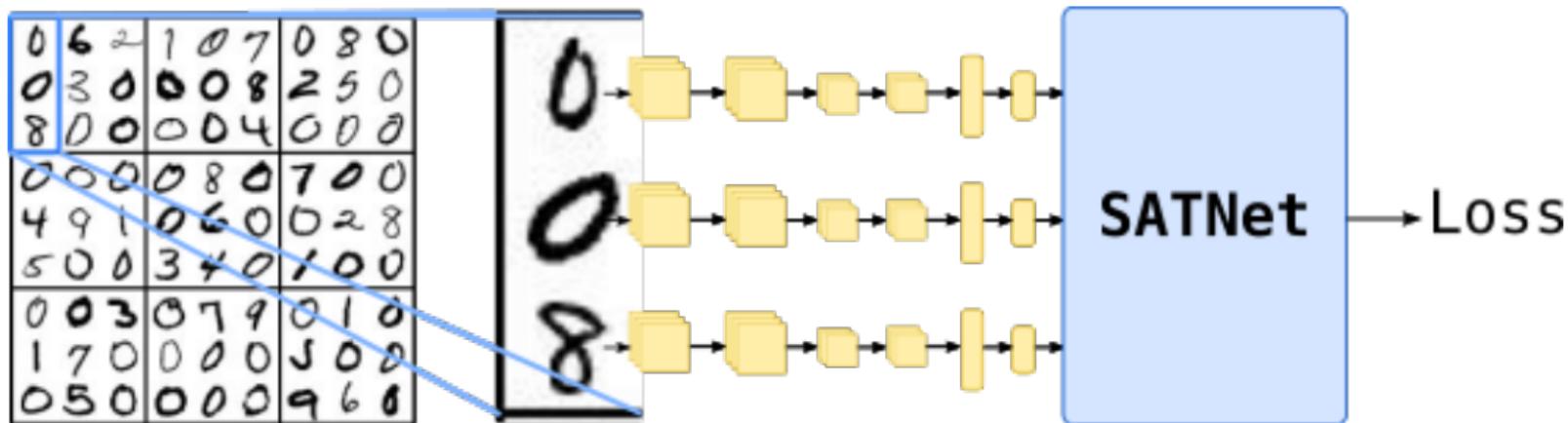
Illustration: MNIST Sudoku



| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 0.31% | 0% |
| SATNet (ours) | 93.6% | 63.2% |

- Getting example “correct” requires correct Sudoku solution *and* predicting all MNIST test digits correctly

Illustration: MNIST Sudoku



| Model | Train | Test |
|----------------------|-------|--------------|
| ConvNet | 0.31% | 0% |
| SATNet (ours) | 93.6% | 63.2% |

- Getting example “correct” requires correct Sudoku solution *and* predicting all MNIST test digits correctly
- 85% accuracy on correct ConvNet input

Code and Colab



Code available at <https://github.com/locuslab/SATNet>

The screenshot shows a Google Colab notebook titled "Learning and Solving Sudoku via SATNet.ipynb". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar are tabs for "CODE", "TEXT", "CELL", and "CELL". A "Table of contents" sidebar is visible on the left, listing sections: "Introduction to SATNet", "Building SATNet-based Models", "The Sudoku Datasets" (with sub-items "Sudoku", "One-hot encoded Boolean Sudoku", and "MNIST Sudoku"), and "The 9x9 Sudoku Experiment". The main area shows a code cell with the following terminal output:

```
!git clone https://github.com/locuslab/SATNet
%cd SATNet
!python setup.py develop > install.log 2>&1

Cloning into 'SATNet'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 47 (delta 12), reused 43 (delta 8), pack
Unpacking objects: 100% (47/47), done.
/content/SATNet

[ ] !wget -cq powei.tw/sudoku.zip && unzip -qq sudoku.zip
!wget -cq powei.tw/parity.zip && unzip -qq parity.zip

[ ] import os
import shutil
import argparse
```

Conclusion

We presented

- **SATNet**, the first **differentiable MAXSAT solver** as a layer

Conclusion

We presented

- **SATNet**, the first **differentiable MAXSAT solver** as a layer
- can be integrated into the loop of deep learning systems whenever neurons have logical constraints, and it **learns both constraints and solutions** solely from examples

Conclusion

We presented

- **SATNet**, the first **differentiable MAXSAT solver** as a layer
- can be integrated into the loop of deep learning systems whenever neurons have logical constraints, and it **learns both constraints and solutions** solely from examples

Possible extensions:

- Incorporating known rules into the system
- Exploiting structures of the clause matrix

Poster at Pacific Ballroom #26