SATNet
Overview

• Background
• Framework
• Issues with SAT Net
  • Integer programming approach
  • End-to-end training results in very poor performance
• Symbol grounding
• Beyond SAT Net
• Concluding Thoughts
SAT Net

Background
The MAX SAT problem

INPUT:
Given Boolean variables $x_1, ..., x_m$ and causes $C_1, ..., C_n$ where each clause is a disjunction of literals (atoms or negations). (Conjunctive normal form)

OUTPUT:
An assignment of Boolean variables such that the number of satisfied clauses is maximized.

Notes:
• Known to be NP-hard (even when each clause has just two literals)
• The clauses can be numerically represented by matrix $(M)$ of dimensions $n \times m$ (in the original paper, $S$ is used instead of $M$)
• Often framed as an optimization problem
Partial Knowledge on Boolean Variables

An extension to the problem is to partition the $m$ Boolean variables into two groups: input ($a_{1,...,k}^{in}$) and output ($a_{k+1,...,m}^{out}$).

Hence, we can think of MAX SAT as the following problem:

$$a_{k+1,...,m}^{out} = S(a_{1,...,k}^{in}, M)$$

Where $S$ is an oracle and $M$ is the matrix representing the clauses.

A **visual** variant of the problem is one in which the input is presented as images instead of text.
Problems

• Solving MAX SAT given the clauses:
  • SAT solvers, Integer programming, Semi definite programming

• Solving visual variant of MAX SAT problem (with known clauses) in an end-to-end fashion
  • CNN + differentiable architecture
  • NeurASP (in a few lessons)

• Solving the MAX SAT problem when the clauses are not known / solving visual variant is processed by a pre-trained CNN
  • This talk

• Solving the visual MAX SAT problem when the clauses are not known and no supervision on images
  • This talk
Sudoku Puzzle

Sudoku is a popular number puzzle that requires you to fill blanks in a 9X9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids contains all of the digits from 1 to 9.

From [https://github.com/Kyubyong/sudoku](https://github.com/Kyubyong/sudoku)

Can be treated as an instance of MAX SAT

Above GitHub site is a good source for training and testing data

Image from Topan et al., NeurIPS 2021.
SAT Net

Framework
SAT Net Framework

Notes:

• For Sudoku, MSE loss is used; cross-entropy loss for visual variant
• A relaxation via Semi Definite programming is used to solve the MAX SAT instance
• Coordinate descent is used instead of gradient descent (both forward and backward passes) as the authors found improved performance with their semidefinite approach to MAX SAT
  • Note that this is for optimization within the forward and backward pass, SGD is used in the overall training
• For the visual Sudoku problem, LeNet is used to classify the digits (pre-trained)
Intuition: Overall Approach

- $Z$ vectors are actual inputs and outputs
- $V$ vectors are relaxations
- $S$ is the weight matrix
- $U$ is a matrix derived from the gradient wrt the relaxed output and used to compute the gradient for the weights

---

Algorithm 1 SATNet Layer

1: **procedure** INIT()
2:  // rank, num aux vars, initial weights, rand vectors
3:   init $m$, $n_{aux}$, $S$
4:   init random unit vectors $v_\mathcal{T}$, $v_{i}^{\text{rand}} \forall i \in \{1, \ldots, n\}$
5:   // smallest $k$ for which (2) recovers SDP solution
6:   set $k = \sqrt{2n} + 1$

7:   
8: **procedure** FORWARD($Z_\mathcal{T}$)
9:   compute $V_\mathcal{T}$ from $Z_\mathcal{T}$ via (5)
10:  compute $V_\mathcal{O}$ from $V_\mathcal{T}$ 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 $\partial \ell / \partial z_\mathcal{T}$, $\partial \ell / \partial s$ from $U$ via (12), (11)
18:   return $\partial \ell / \partial z_\mathcal{T}$

---

Algorithm from Wang et al., ICML 2019.
Intuition: Forward Pass

- Key thing to remember: the forward pass is itself solving another optimization problem (using the SDP relaxation) – this is just a subroutine in the overall optimization process.
- The outputs of the forward pass are evaluated against a standard loss function.
Intuition: Backward Pass

- Key result to derive the gradients w.r.t. weights and inputs
- $U$ is the key item to compute and is done via coordinate descent

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

$$\text{vec}(U_O) = (P((C + D) \otimes I_k)P)^\dagger \text{vec} \left( \frac{\partial \ell}{\partial V_O} \right),$$

(9)

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

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

(10)

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

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

(11)

Theorem 1 is from Wang et al., ICML 2019.
Sudoku Experimental Results

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>72.6%</td>
<td>0.04%</td>
</tr>
<tr>
<td>ConvNetMask</td>
<td>91.4%</td>
<td>15.1%</td>
</tr>
<tr>
<td>SATNet (ours)</td>
<td>99.8%</td>
<td>98.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>ConvNetMask</td>
<td>0.01%</td>
<td>0%</td>
</tr>
<tr>
<td>SATNet (ours)</td>
<td>99.7%</td>
<td>98.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>0.31%</td>
<td>0%</td>
</tr>
<tr>
<td>ConvNetMask</td>
<td>89%</td>
<td>0.1%</td>
</tr>
<tr>
<td>SATNet (ours)</td>
<td>93.6%</td>
<td>63.2%</td>
</tr>
</tbody>
</table>

(a) Original Sudoku. (b) Permutated Sudoku. (c) Visual Sudoku. (Note: the theoretical “best” test accuracy for our architecture is 74.7%.)

- Uses approach of [https://github.com/Kyubbyong/sudoku](https://github.com/Kyubbyong/sudoku) as a baseline (note that is only a GitHub site, not a paper, and was only evaluating Sudoku on CNN’s compared to training data)
- The Visual Sudoku tests used LeNet architecture into the end-to-end framework – the authors claimed end-to-end training
- Permutated rows were to illustrate that SAT Net was not overfitting based on row position
- The number of clauses learned by SAT Net was limited to avoid overfitting (m=600)
- Why does the CNN fail and end up overfitting? This is a combinatorial problem, so the amount of required training data could be exponential
Sudoku Experimental Results

### Results of visual case highly sensitive to initial conditions (~80% reduction in accuracy)

<table>
<thead>
<tr>
<th>Model</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>ConvNetMask</td>
<td>0.01%</td>
<td>0%</td>
</tr>
<tr>
<td>SATNet (ours)</td>
<td>99.7%</td>
<td>98.3%</td>
</tr>
</tbody>
</table>

### The output labels included the symbolic representation of the input digits – so there was label leakage (which actually causes total failure)

- The Visual Sudoku tests used LeNet architecture into the end-to-end framework – the authors claimed end-to-end training.
- Arbitrarily increasing number of parameters led to model failure (unlike standard DL).
- Why does the CNN fail and end up overfitting? This is a combinatorial problem, so the amount of required training data could be exponential.

### Chang et al., NeurIPS 2020 showed:

- Uses approach of [https://github.com/Kyubyong/sudoku](https://github.com/Kyubyong/sudoku) as a baseline (note that is only a GitHub site, not a paper, and was only evaluating Sudoku on CNN’s compared to training data).
- The Visual Sudoku tests used LeNet architecture into the end-to-end framework – the authors claimed end-to-end training.
- The number of clauses learned by SAT Net was limited to avoid overfitting (m=600).
- Why does the CNN fail and end up overfitting? This is a combinatorial problem, so the amount of required training data could be exponential.

### Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>0.31%</td>
<td>0%</td>
</tr>
<tr>
<td>ConvNetMask</td>
<td>89%</td>
<td>0.1%</td>
</tr>
<tr>
<td>SATNet (ours)</td>
<td>93.6%</td>
<td>63.2%</td>
</tr>
</tbody>
</table>

*Note: the theoretical “best” test accuracy for our architecture is 74.7%.*
Despite its shortcomings, SAT Net has significance:

- It successfully could learn constraints in a differentiable framework.

- Combinatorial forward pass and ability to derive gradients for backpropagation.

- Significantly outperformed standard DL architectures:
  - (this was significant at the time of publication)
SAT Net

Issues with SAT Net
Integer Programming Approach Outperforms SAT Net on Sudoku

Strategy: derive constraints from the inner and outer polytopes based on training data. Squeeze polytopes until convergence.

Outperformed SAT Net on Sudoku.

However, this approach is not differentiable.

But, if SAT Net cannot be used in end-to-end training, then why not just use an IP approach and get better results

Table from Meng and Chang, ICML 2021.
The Case Against SAT Net

At NeurIPS 2020, Chang et al., reviewed SAT Net in a paper and found major issues with the results around Visual Sudoku

• Label Leakage

• Initialization / hyperparameters / training decisions

• Symbol grounding
Label Leakage

- Wang et al. claimed SAT Net for Visual Sudoku was trained in an end-to-end fashion.
- However, while training samples were visual, the ground truth labels included the symbolic representation of the visual input.

\[ \begin{array}{c}
\alpha_{\text{visual}}^\text{in} \\
\end{array} \quad \rightarrow \quad \begin{bmatrix}
1 & 8 & 5 \\
2 & 6 & 9 \\
3 & 4 & 7 \\
\end{bmatrix} \]

\[ \alpha_{\text{in}} \quad \alpha_{\text{out}} \]

Image from Topan et al., NeurIPS 2021.
Overall accuracy only surpasses zero when the recognition of digits is properly occurs.
The relationship between symbols and perception

- Transduction problem
  - If they exist, how then, are the perceptual states mapped into amodal symbols? (Barsalou, 1999)

- Symbol grounding problem
  - The reverse of the transduction problem
  - How are amodal symbols grounded in perception? (Barsalou, 1999)

- Chang et al. argue that SAT Net did not adequately solve the symbol grounding problem.
  - They probably really mean the transduction problem – as the issue was the transduction of perception into symbols
  - Note: Symbol grounding does come up in ML verification – ensuring that a symbol maps back properly to perception
Failure of symbol grounding

Challenges in symbol grounding are somewhat different than the problem of SAT Net – i.e., how do we determine (without human inspection) that symbols are properly grounded in perception.

Source: Zhang et al., 2019: https://openreview.net/pdf?id=SJgEl3A5tm
Perception-Cognition: A view from cognitive science

Transduction deals with mapping perception to symbolic representations
Testing SAT Net’s ability to address transduction

Chang et al. re-evaluate SAT Net by masking the output

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Non-Visual Sudoku</th>
<th>Visual Sudoku</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Masked Outputs</td>
</tr>
<tr>
<td>Train</td>
<td>99.7±0.0%</td>
<td>99.7±0.0%</td>
</tr>
<tr>
<td>Test</td>
<td>97.6±0.1%</td>
<td>97.6±0.1%</td>
</tr>
</tbody>
</table>

The argument presented by Chang et al. is that SAT Net does not solve the transduction* but only appeared to due to a combination of label leakage and tuned training/hyperparameter settings.

They further explore the limits of SAT Net in their paper for visual problems

However, the non-visual results largely held

*The authors use the term “symbol grounding”
Unsupervised Learning to address transduction

• Topan et al. seek to directly address the shortcomings of SAT Net:
  • Use of unsupervised learning for digit recognition
  • Additional loss term to account for in accurate digits
  • Addition of proofreader layer improved performance (an extra boost, but not directly related to the problem of transduction)
Use of permutation matrix

\[ \hat{y}_{in}^{PTE} P = \hat{y}_{in}^{LE} \]

• The pre-trained encodings (PTE) of input digits is directly related to the (correct) label encodings (LE) by some permutation matrix P.

\[ \hat{y}_{out}^{PTE} P \approx y^{LE} \]

• We can assume that results from the supervised training is related to label encoding via the same matrix P.

Image from Topan et al., NeurIPS 2021.
Two-phased training

1. Freeze digit classifier (the unsupervised method) and train SAT Net weights with a special loss function that allows us to also learn permutation matrix P. Stop training upon convergence of P.

2. Freeze P and switch to traditional loss (cross-entropy) and learn weights; the digit classifier is also unfrozen at this point as well.
Table from Topan et al., NeurIPS 2021.

<table>
<thead>
<tr>
<th>Model Configuration</th>
<th>Grounded vs. Ungrounded Data</th>
<th>Total Board Accuracy (%)</th>
<th>Per-Cell Accuracy (%)</th>
<th>Visual Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original SATNet</td>
<td>grounded</td>
<td>66.5 ± 1.0</td>
<td>98.8 ± 0.1</td>
<td>99.0 ± 0.0</td>
</tr>
<tr>
<td>Original SATNet</td>
<td>ungrounded</td>
<td>0 ± 0.0</td>
<td>11.2 ± 0.1</td>
<td>11.6 ± 0.0</td>
</tr>
<tr>
<td>Our Method</td>
<td>ungrounded</td>
<td>64.8 ± 3.0</td>
<td>98.4 ± 0.2</td>
<td>98.9 ± 0.1</td>
</tr>
</tbody>
</table>
Concluding Thoughts

• The original SAT Net paper studied how to learn logical constraints, do back propagation where the forward function is a combinatorial problem and address transduction – was this too much of a leap for a single study?

• Practicalities of training significantly altered results – how often does this happen in DL papers?

• Topan et al. do not fully solve transduction – they just do it for Sudoku. They essentially leverage the fact that the solution still has numbers (even when unmasked) – will this be the case in other, more real-world problems?

• There are other approaches to learning of constraints (papers posted on today’s lesson)