Learning a SAT Solver from Single-Bit Supervision

Daniel Selsman, Matthew Lamm, Benedikt Bunz, Percy Liang, Leonardo de Moura and David L. Dill

Presented By Aditya Sanghi

Overview

- NeuroSAT
- Background:
  - SAT Problem
  - Message Passing Neural Networks
- Model
- Training Details
- Model Results:
  - Predicting Satisfiability
  - Decoding Satisfying Assignments
  - Generalizing to other Problem Distribution
- NeuroUNSAT

NeuroSAT

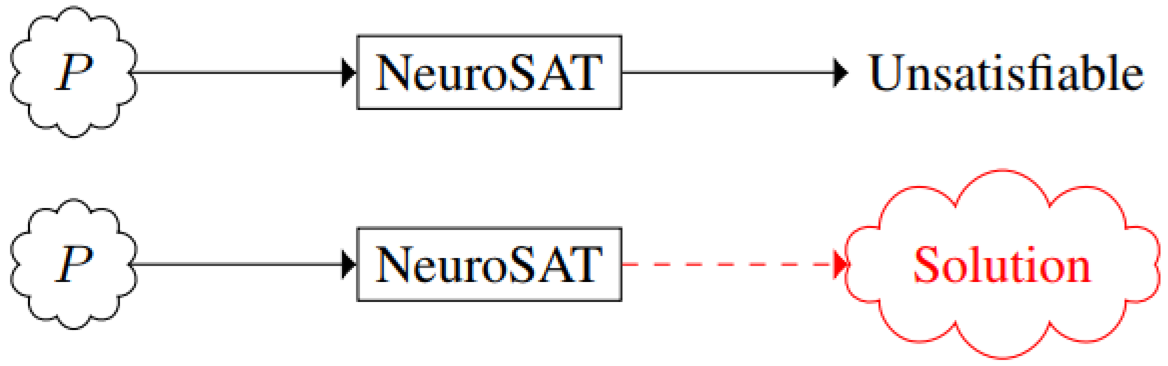
- **Goal**: Can a neural network learn to solve SAT problems?
- Use MPNN framework that learns to solve SAT problem after only trained as a classifier to predict satisfiability on a dataset of random SAT problems
- Train with one bit supervision for each SAT problem to indicate whether or not the problem is satisfiable
- During Test Time, if the solution is satisfiable, the network initially guesses unsatisfiability with low confidence until it finds a solution at which point it converges and guesses satisfiability with high confidence

NeuroSAT

Train:

$$\left\{ \begin{array}{ll} \text{Input:} & \text{SAT problem P} \\ \text{Output:} & \mathbb{1}\{P \text{ is satisfiable}\} \end{array} \right\}$$

Test:



SAT Problem – Satisfiable

- **Formula of propositional logic**: A Boolean expression built using constants true and false, variables, negations, conjunctions, and disjunctions
- Example:
  - $(x_1 \wedge \neg x_2)$
  - $(x_1 \wedge \neg x_1)$
  - $(x_1 \vee x_2 \vee x_3) \wedge \neg (x_1 \wedge x_2 \wedge x_3)$
- **Satisfiable**: A formula is satisfiable if there exists a assignment of Boolean values to its variables such that the formula evaluated to 1
- Example:
  - $(x_1 \wedge \neg x_2)$ is true when x_1 is true and x_2 is false. This is satisfiable
  - $(x_1 \wedge \neg x_1)$ is always false for any assignment of x_1. This is unsatisfiable.

SAT Problem

- **Conjunctive Normal Form (CNF):** A formula expressed in a conjunction of disjunctions of (possibly negated) variables
- Example:
  - $(x_1 \lor x_2 \lor x_3) \land \neg (x_1 \land x_2 \land x_3)$ CNF form is $(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3)$
- Every formula has a equisatisfiable Conjunctive Normal Form
- **Clause**: Each conjunct of a formula in CNF
- **Literal**: Each variable within a clause
- **SAT Problem**: Is a formula in CNF, where the goal is to determine if the formula is satisfiable, and if so, to produce a satisfying assignment of truth values to variables

Message Passing Neural Networks

- **Message Passing Neural Networks (MPNNs)**: A general framework for supervised learning on graphs
- MPNNs can be used to operate on undirected graphs G with node features x_v and edge features e_{vw}.
- The forward pass has two phases:
  - Message Passing Phase:
    - Runs for T time steps
    - Defined in terms of Message function M and vertex update function U

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$
$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

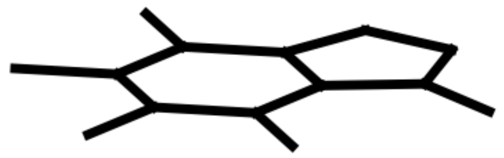
  - Readout Phase:
    - Feature vector for the whole graph using some readout function R

$$\hat{y} = R(\{h_v^T \mid v \in G\}).$$

source: https://arxiv.org/abs/1704.01212

Message Passing Neural Networks – example

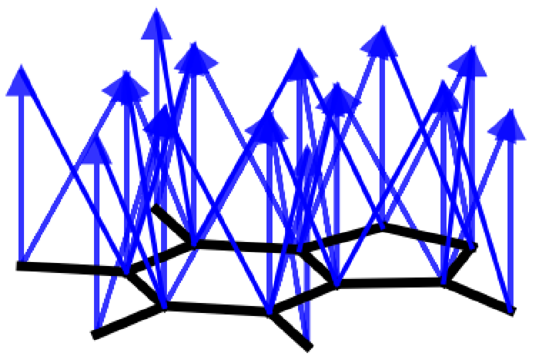
- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



source: Glem et al., 2006

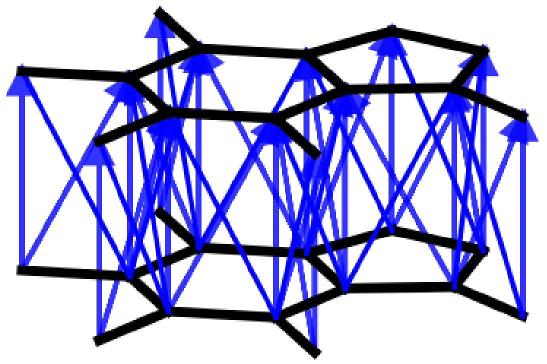
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



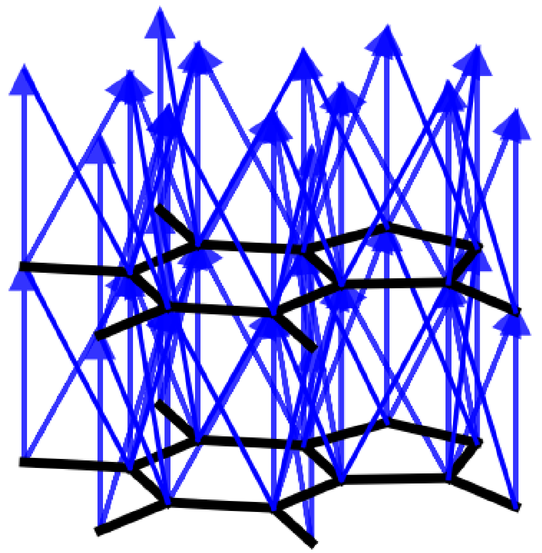
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



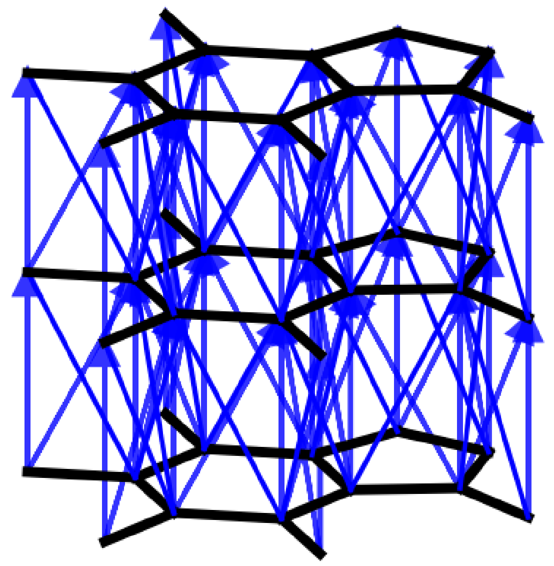
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



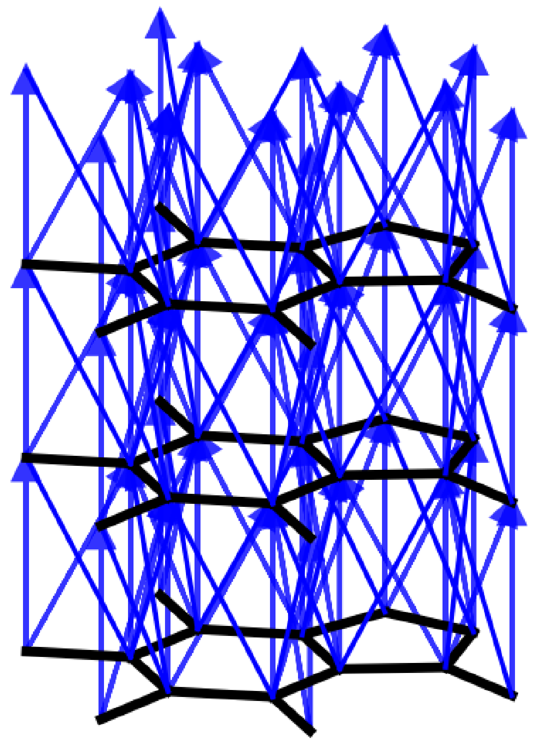
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



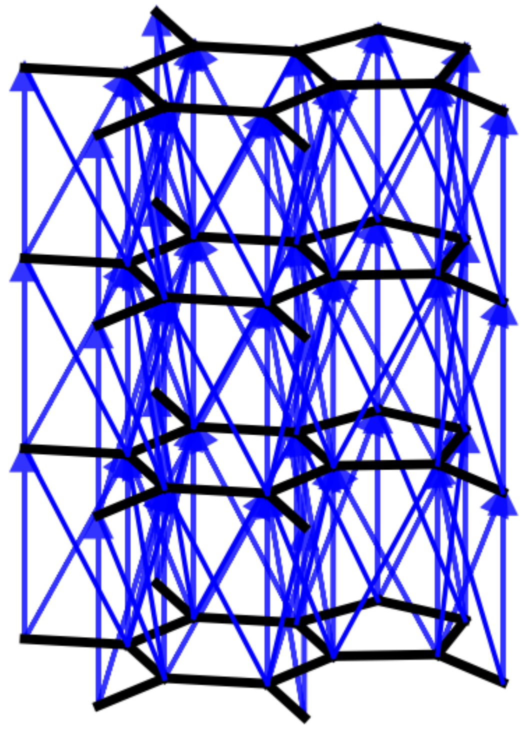
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



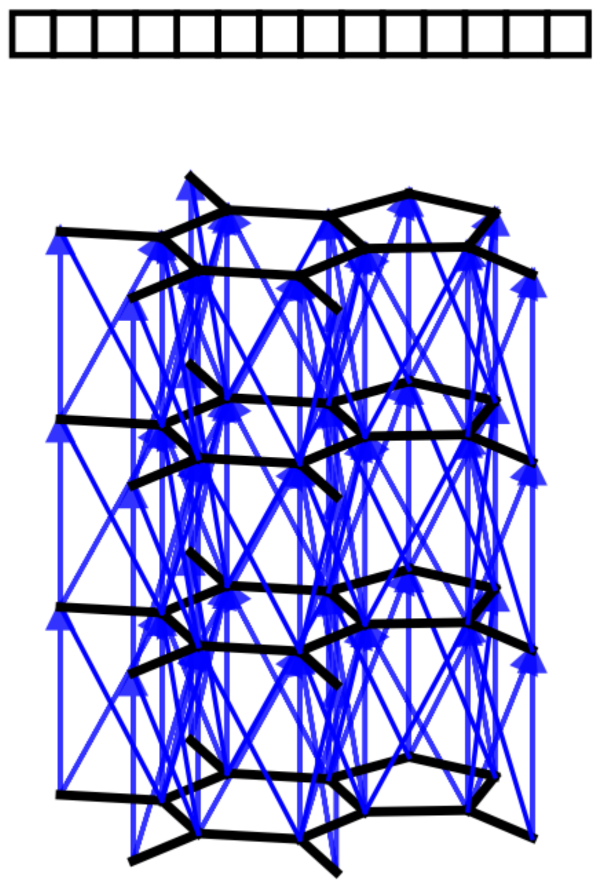
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



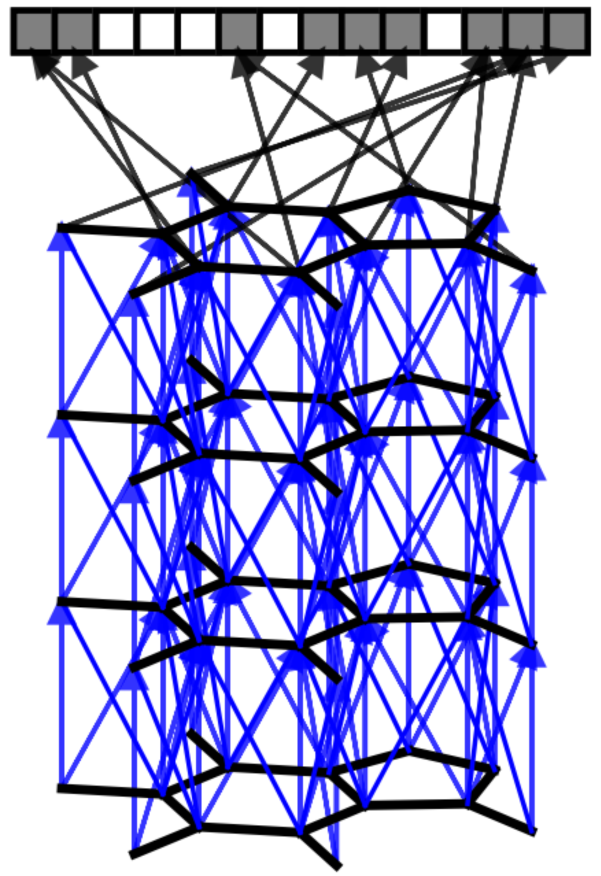
Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector



Message Passing Neural Networks – example

- A graph is constructed matching the topology of molecule being fingerprinted
- Each node represents atom and edge represents bonds
- At each iteration, information flows between neighbors in the graph
- Finally, each node in the graph turns on one bit in the fixed-length fingerprint vector

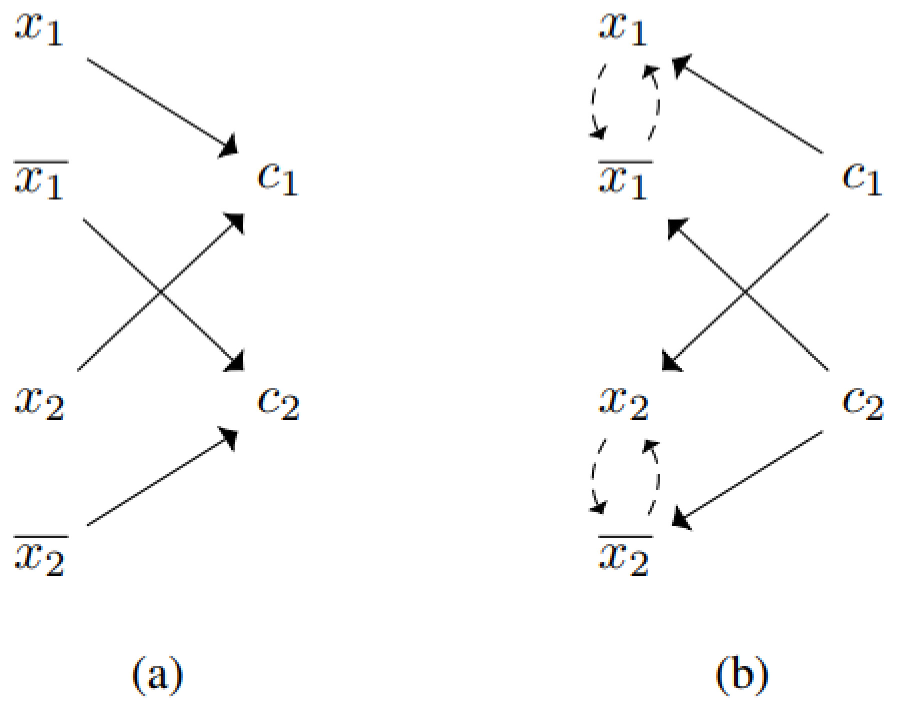


source: Glem et al., 2006

Model

- Encode the SAT problem as an undirected graph
  - One node for every clause and every literal
  - Embedding for every literal and every clause
  - An edge between every literal and every clause it appears in
  - An edge between each pair of complementary literals
- NeuroSAT iteratively refines a vector space embedding for each node by passing messages back and forth along the edges of the graph
- An iteration consists of two stages:
  - Each clause receives messages from its neighboring literals and updates its embedding accordingly
  - Each literal receives its message from its neighboring clauses as well from its complement and update its embedding

Model



(a) Stage 1 (b) Stage 2

Graph representation of $\{1|2, \bar{1}|\bar{2}\}$

Model

- The embeddings for literal and clause at each time step are represented by

$$L^{(t)} \in \mathbb{R}^{2n \times d} \text{ and } C^{(t)} \in \mathbb{R}^{m \times d}$$

- Parametrized by three multilayer perceptrons: $\mathbf{L}_{\text{msg}}$, $\mathbf{C}_{\text{msg}}$ and $\mathbf{L}_{\text{vote}}$
- Also parametrized by two-layer LSTMs: $\mathbf{L}_{\mathbf{u}}$ and $\mathbf{C}_{\mathbf{u}}$
- Hidden states of these LSTM are

$$L_h^{(t)} \in \mathbb{R}^{2n \times d} \text{ and } C_h^{(t)} \in \mathbb{R}^{m \times d}$$

- *M* is the adjacency matrix

$$M(i,j) = \mathbb{1}\left\{\ell_i \in c_j\right\}$$

- F is the operator that takes a matrix *L* and swaps each row of *L* with the corresponding to the literal's negation

Model

- A single iteration consist of applying –

$$(C^{(t+1)}, C_h^{(t+1)}) \leftarrow \mathbf{C}_\mathrm{u}([C_h^{(t)}, M^\top \mathbf{L}_\mathrm{msg}(L^{(t)})])$$
$$(L^{(t+1)}, L_h^{(t+1)}) \leftarrow \mathbf{L}_\mathrm{u}([L_h^{(t)}, \mathrm{F}(L^{(t)}), M\mathbf{C}_\mathrm{msg}(C^{(t+1)})])$$

- After T iterations, we compute

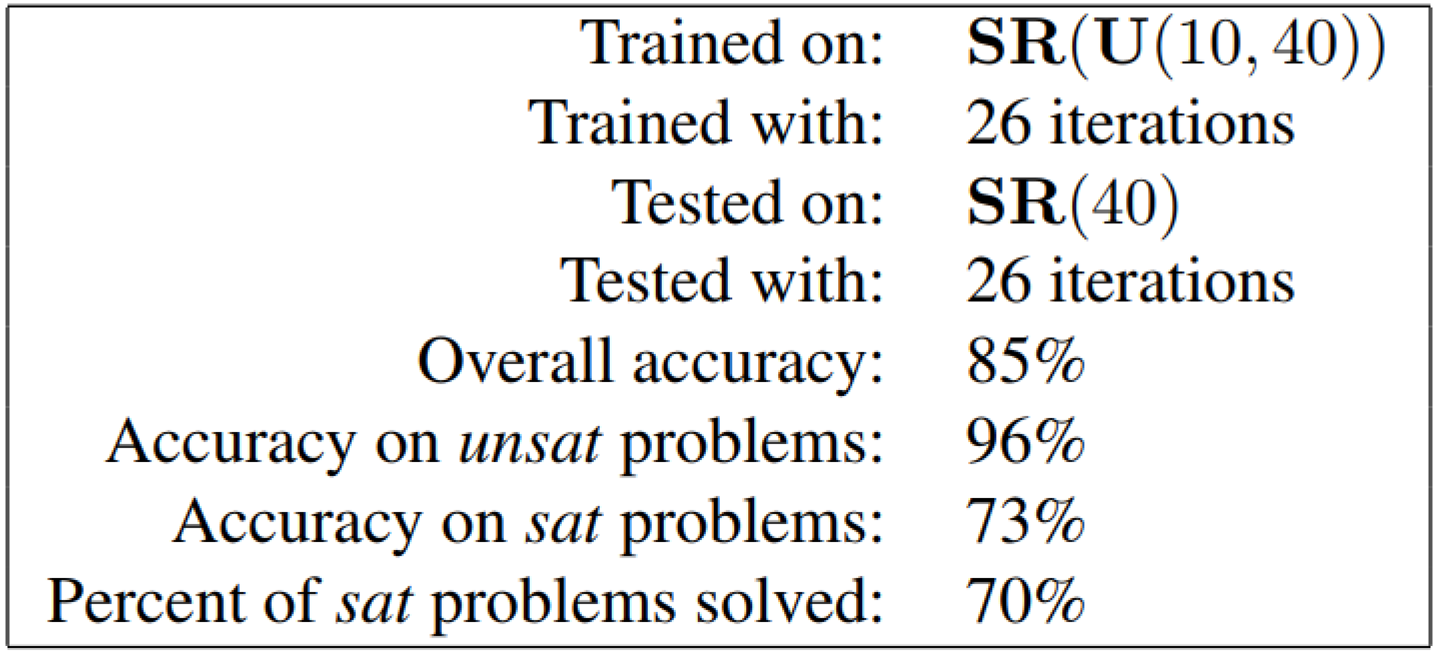
$$L_*^{(T)} \leftarrow \mathbf{L}_\mathrm{vote}(L^{(T)})$$
$$y^{(T)} \leftarrow \mathrm{mean}(L_*^{(T)})$$

- Each literals vote is contained in $L_*^{(T)} \in \mathbb{R}^{2n}$
- The average of vote is $y^{(T)} \in \mathbb{R}$
- We train the network using log likelihood

Training Data

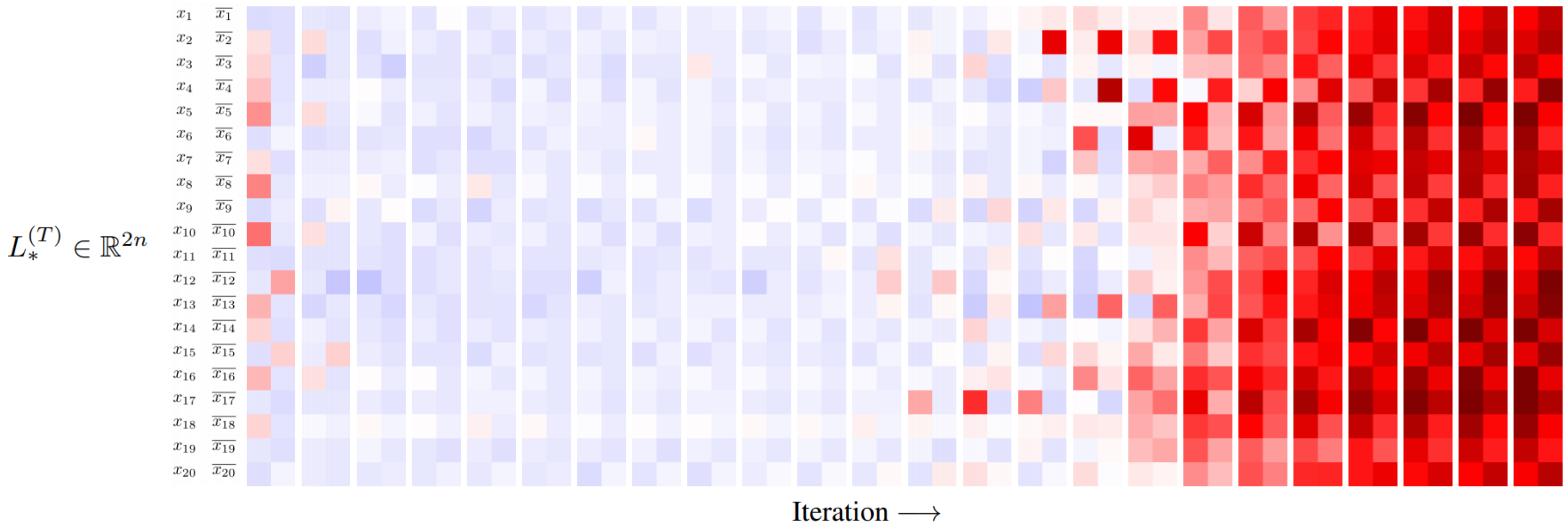
- Create a distribution **SR**(n) over pair of random SAT problems on n variables
- One element of the pair is satisfiable and other is unsatisfiable
- The two differ by negating only a single literal occurrence in a single clause
- To generate a random clause on n variable
  - **SR**(n) first samples a integer k so as to create clauses of variable size
  - Then k variables are sampled randomly without replacement
  - Finally, negate each one of them with 50% probability
  - Keeps generating clauses like this until the problem is unsatisfiable
  - So the pair are a sample from **SR**(n)

Result - Predicting Satisfiability



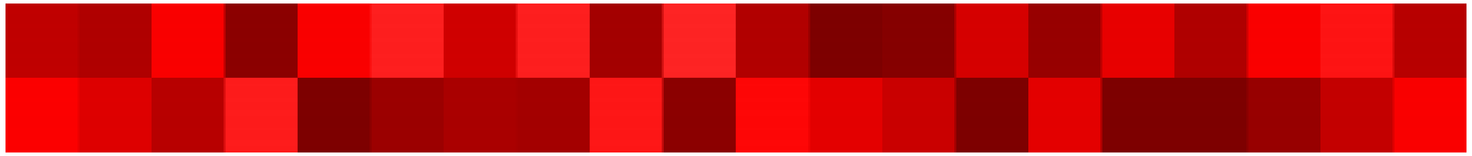
Trained on:	$\mathbf{SR}(\mathbf{U}(10, 40))$
Trained with:	26 iterations
Tested on:	$\mathbf{SR}(40)$
Tested with:	26 iterations
Overall accuracy:	85%
Accuracy on *unsat* problems:	96%
Accuracy on *sat* problems:	73%
Percent of *sat* problems solved:	70%

Result – Decoding Satisfying assignment



The sequence of literal votes $L_*^{(1)}$ to $L_*^{(24)}$ as NeuroSAT runs on a satisfiable problem from $\mathbf{SR}(20)$.

Result – Decoding Satisfying assignment



literal votes $L_*^{(24)}$

Result – Decoding Satisfying assignment

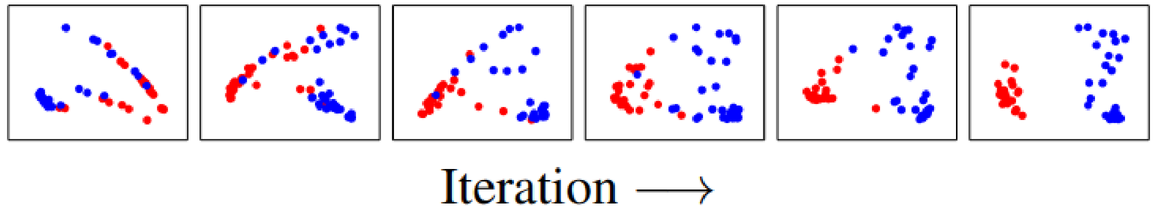


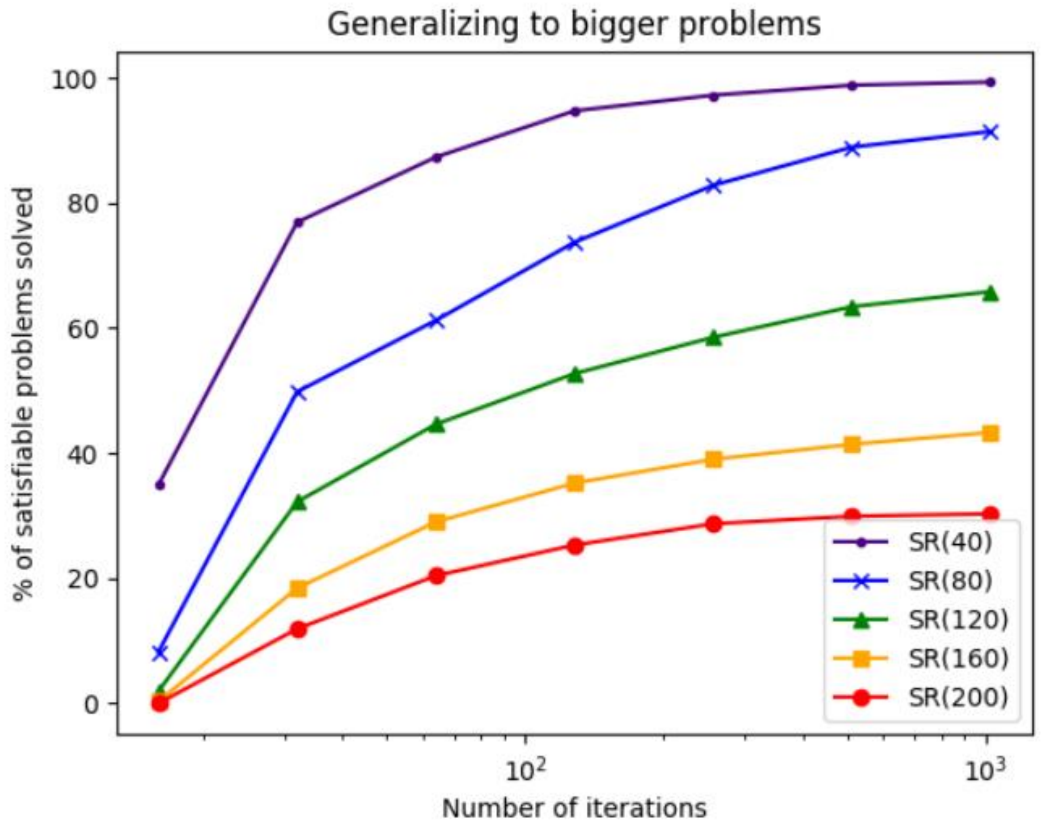
Figure 4. PCA projections for the high-dimensional literal embeddings $L^{(16)}$ to $L^{(26)}$ (skipping every other time step) as NeuroSAT runs on a satisfiable problem from $\mathbf{SR}(40)$. Blue and red dots indicate literals that are set to 0 and 1 in the satisfying assignment that it eventually finds, respectively. We see that the blue and red dots are mixed up and cannot be linearly separated until the phase transition at the end, at which point they form two distinct clusters according to the satisfying assignment.

- Decode the solution from NeuroSAT's internal activation:
  - 2-cluster $L^{(t)}$ to get cluster centers
  - Partition the variable according to the predicate –

$$\|x_i - \Delta_1\|^2 + \|\overline{x_i} - \Delta_2\|^2 < \|x_i - \Delta_2\|^2 + \|\overline{x_i} - \Delta_1\|^2$$

  - Then try both candidate assignment
- Decodes a satisfying assignment for over 70% of the satisfiable problem

Result – Generalizing to Bigger Problems



Result – Generalizing to Different Problems

Task	μ_{vars}	μ_{clauses}	#sat	%solved
3-color	30	89	350	64%
4-color	40	135	557	69%
5-color	50	191	590	54%
3-clique	30	389	586	88%
4-clique	40	686	241	96%
5-clique	50	1067	43	28%
2-domset	30	264	278	99%
3-domset	40	454	574	95%
4-domset	50	689	600	100%
4-cover	50	696	128	100%
5-cover	60	976	357	100%
6-cover	70	1301	584	96%
all	45	532	4888	85%

NeuroUNSAT

- NeuroSAT architecture keeps searching for a satisfying assignment non-systematically for a unsatisfiable problem
- Can train the same architecture with small subset of clauses that are already unsatisfiable (unsat cores)
- This makes the architecture learn to detect this unsat cores instead of searching for satisfying assignment
- Generate a new distribution SRC(n,u) and train it on that architecture

Conclusion

- Advantage:
  - NeuroSAT can solve substantially larger and more difficult than it ever saw during training by performing more iteration of message parsing
  - The learning process has yielded a procedure that can be run indefinitely to search for solutions to problems of varying difficulties
  - Generalized to completely new domain; i.e. SAT problems encoding any kind of search problem.
    - Examples: SAT problems encoding graph coloring, clique detection, dominating set, and vertex cover problems, all on a range of distributions over small random graphs
  - Same architecture can be used to help find proof for unsatisfiable problems
- Disadvantage: Does not beat state-of-the-art SAT solvers