Neural heuristics for SAT solving

Sebastian Jaszczur
University of Warsaw

Michał Łuszczyk
University of Warsaw

Henryk Michalewski
University of Warsaw, deepsense.ai

Abstract

We use neural graph networks with a message-passing architecture and an attention mechanism to enhance the branching heuristic in two SAT-solving algorithms. We report improvements of learned neural heuristics compared with two standard human-designed heuristics.

1 Introduction

The Boolean satisfiability problem (SAT) is the problem of determining the existence of a solution for a given propositional logic formula. It is a NP-complete problem, meaning that any NP problem can be reduced to SAT problem in polynomial time (Kar72).

We explore the possibility of using neural networks in SAT solving as branching heuristics in search algorithms[1]. We focus on two SAT-solving algorithms: DPLL (Algorithm 1) and more advanced CDCL. Both of those are complete backtracking-based search algorithms. Both depend on the branching heuristic `choose-literal`, which chooses branching variable and its Boolean value. The expected running time is heavily dependent on the quality

---

1: **function** DPLL(Φ)
2:     $\Phi \leftarrow$ `simplify`(Φ)
3:     **if** Φ is trivially satisfiable **then return** True
4:     **if** Φ is trivially unsatisfiable **then return** False
5:     $literal \leftarrow$ `choose-literal`(Φ)
6:     **if** DPLL($\Phi \wedge literal$) **then return** True
7:     **if** DPLL($\Phi \wedge \neg literal$) **then return** True
8:     **return** False

Algorithm 1: High level overview of DPLL. In this work, we embed neural network as `choose-literal`*. In DPLL,* `simplify` *contains unit propagation and clause elimination. Trivially satisfiable and trivially unsatisfiable for CNF means respectively an empty formula, and a formula containing an empty clause.*

of this heuristic (MS99). In this work we use neural networks as `choose-literal` heuristic and compare its performance with DLIS and Jeroslow-Wang One-Sided (JW-OS) heuristics, which are presented in (MS99; MMZ+01) as one of the best strategies in most circumstances. We compare the performance in terms of number of branching decisions and show the possibility of enhancing the performance of SAT solvers with the help of learned heuristics.

2 Related work

(SLB+18) proposed the **NeuroSAT** message-passing network architecture for SAT-solving that generates the assignment of variables directly in the graph. This is an important inspiration for our work, although in contrast to (SLB+18) we use a message-passing network for guidance of a backtracking-based algorithm instead. A similar graph representation, but more general in order to accommodate for higher-order logic is used in **FormulaNet** presented in (WTWD17). To the best of our knowledge the FormulaNet architecture was never used for neural guidance. In (SLB+18; WTWD17) formulas are represented as graphs and a general approach to neural networks and graphs, including the attention mechanism, can be found in (BHB+18; VCC+17). The **PossibleWorldNet** architecture described in (ESA+18) is based on the **TreeNN** architecture, with an additional idea of checking multiple possible worlds. We consider the exploration of possible worlds as an alternative to structured backtracking-based search algorithms like DPLL and CDCL. It is worth noting that PossibleWorldNet could be modified to use a message-passing architecture while keeping the exploration of possible worlds. Another application of TreeNN for proof synthesis in propositional logic was proposed in (SS18). **EqNet** (ACKS16) solves a more

---

[1] See `https://bit.ly/neurheur` for a TPU-bound implementation of all algorithms in this paper.

general problem of determining equivalence of Boolean (alternatively: arithmetic) expressions (satisfiability can be seen as equivalence to any unsatisfiable formula e.g. $a \wedge \neg a$). However, the formulas solved by EqNet have up to 10 variables and 13 symbols, while we tackle formulas beyond one hundred variables and thousands of symbols. **Learned Restart Policy** (LOM[+]18) presents a different approach to improve a SAT solver with machine learning, where the network decides at each step whether the algorithm should be restarted to follow another random path in the search tree.

3 ARCHITECTURE

We use a message-passing graph-based neural network architecture similar to NeuroSAT introduced in (SLB[+]18). The general idea is to represent a formula as a graph with two node types (literal and clause) and two edge types (literal-literal edges represent the negation relation, and clause-literal edges represent relation between each clause and literals it contains). Example formula represented as a graph is shown in Figure 2 Left. Each node has its own state, represented by an embedding vector. Thanks to this representation, we have the following properties: **1.** Invariance to variable renaming. **2.** Invariance to negation of all occurrences of a variable. **3.** Invariance to permutation of literals in a clause. **4.** Invariance to permutation of clauses in a formula.
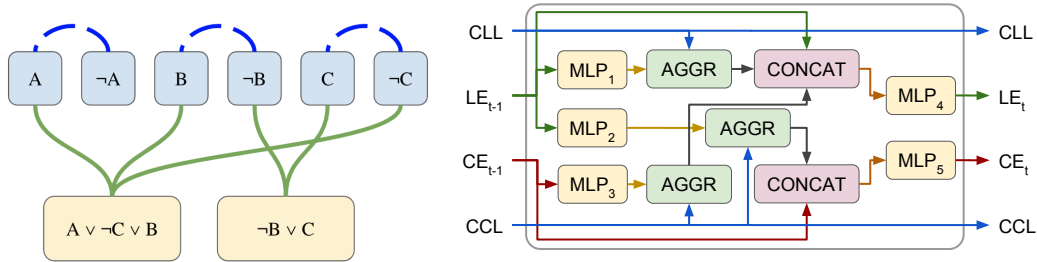

Figure 2: Left: A graph representation of formula $(A \vee \neg C \vee B) \wedge (\neg B \vee C)$ used in our work. In the model nodes are unlabeled (labels are included only for the reader's convenience). Different colors mark two distinct types of nodes (clause and literal) and two distinct types of edges (literal-literal and clause-literal). Right: Overview of message-passing architecture. In each iteration we take as the input: connection matrix between clauses and literals (CCL), connection between literals and their negations (CLL), literal embeddings from previous iteration (LE_{t-1}), and clause embeddings from previous iteration (CE_{t-1}). We use 5 separate MLPs, which share parameters across iterations. Aggregation method depends on a model, see the description below.

We initialize all embedding vectors with a trainable initial embedding, different for each type of node. Then we run a number of iterations (from 20 to 40 in our experiments), visualized in Figure 2 Right. Each iteration consists of three stages: **Stage 1.** Message: Each node generates a message vector V (and a vector K if needed) based on its embedding, to every connected node. V and K are generated with a three-layer MLP with LeakyReLU (MHN13) activation after each hidden layer and linear activation after the last layer. **Stage 2.** Aggregate: all messages are delivered according to the connection matrix, then aggregated for each receiver with one of the aggregation functions (described in the next paragraph). **Stage 3.** Update: Each node updates its embedding based on its previous embedding and aggregated received messages. New embedding is computed by a three-layer MLP with LeakyReLU activation after each hidden layer and sigmoid activation after the last layer.

We explore two different aggregation methods. The first is the average of received V vectors. The second method is a modified attention mechanism. As a message, instead of just a single vector V, we send two vectors, V and K. Receiving node generates one vector Q based on its embedding, and the result of aggregation is $\sum_i V_i \, sigmoid(K_i \cdot Q)$. Thanks to this, each message may be selectively rejected or accepted by the receiver, depending on relation between K and Q. The intuitive difference between this mechanism and the standard attention is as follows: the standard attention as in (VCC[+]17) chooses one message to look at, while our mechanism rejects or accepts messages independently and looks at their sum.

Like NeuroSAT, our architecture learns to predict satisfiability of the whole formula (which we name *sat prediction*). However, it also predicts, for each literal separately, the existence of a solution

with this literal (which we name *policy prediction*). To get *policy prediction* we add on top of each literal's embedding in each iteration a logistic regression (with parameters shared across all literals and iterations). To get *sat prediction* we add a linear regression on top of each literal's embedding in each iteration, and then apply a sigmoid on sum of their outputs. We define *sat loss* as cross-entropy loss between the *sat prediction* and the ground truth. We define *policy loss* as zero if formula is unsatisfiable and as the average of cross-entropy losses between *policy predictions* and ground truths if whole formula is satisfiable. To get a loss of the whole model we sum together both losses for every iteration.

4 EXPERIMENTAL RESULTS

Dataset and training details.   To train and evaluate the models we use a class of SAT problems $SR(n)$ introduced and described in detail in (SLB$^+$18). It is parametrized only by n – the number of variables used in a formula. Both the size and the number of clauses vary. The dataset is balanced in terms of number of satisfiable and unsatisfiable examples. Each of the $SR(n)$ samples has two labels (see Section 3): *sat* indicating whether the formula Φ is satisfiable and *policy* indicating for each literal l whether $\Phi \wedge l$ is satisfiable. We generate each of those numbers by running MiniSat 2.2 (ES03). Sample random $SR(30)$ formulas are solved by MiniSAT 2.2 in 0.007 seconds, while $SR(110)$ takes 0.137 second and $SR(150)$ takes 3.406 seconds (for a Xeon E5-2680v3@2,5 GHz computer). We have trained separate models on SR(30), SR(50), SR(70) and SR(100). Table 1 shows the details of the training procedure. Metrics *sat* error and *policy* error are defined as mean absolute error of *sat* or *policy* prediction versus labels. The presented models are message-passing neural networks with our modified attention mechanism.

Problem	Loss	*sat* error	*policy* error	Batch size	Train. steps	Train. time
SR(30)	28.178±0.672	0.084±0.004	0.050±0.002	128	1200K	20h
SR(50)	32.024±0.555	0.233±0.017	0.105±0.006	64	600K	12h
SR(70)	33.010±0.482	0.266±0.033	0.110±0.007	64	600K	22h
SR(100)	34.227±0.127	0.319±0.007	0.123±0.002	32	1200K	28h

Table 1: *Each of the models was trained on SAT samples drawn from the distribution marked in the first column. The metrics: loss, sat error and policy error are evaluated on an independently generated evaluation set. The values indicate mean and standard deviation over 3-5 trained models. Models were trained using single TPU v2.*

Experiment 1: comparison of all models with DLIS and JW-OS heuristics.   We evaluated the DPLL algorithm guided by our 4 kinds of models described above and compared to DPLL guided by JW-OS and DLIS. As a performance consideration we decided to stop DPLL after 1000 steps (see Experiment 2 below for a comparison without this restriction) and count the number of solved formulas out of 100 in each class. We present the results in Figure 3. For this and subsequent experiments we only consider satisfiable $SR(n)$ samples. JW-OS proved to be the best on average classes of problems: $SR(50)$ and $SR(70)$, whereas neural guidance-based algorithms proved to be the best on large problems: $SR(90)$ and $SR(110)$.
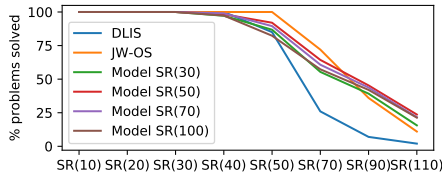

Figure 3: *Performance of DPLL with different guidance heuristics on specific problem sizes. The x axis indicates the class of the evaluation set: evaluation is performed on fresh randomly chosen one satisfiable hundred $SR(x)$ formulas. The y axis indicates the percent of instances (out of 100) solved by DPLL within 1000 steps.*

Experiment 2: detailed comparison with the JW-OS heuristic.   We have selected the SR(50) model for a detailed comparison of the learned heuristics versus JW-OS and for the sake of this comparison designed hybrid guidance algorithm that uses a model trained on $SR(50)$ (a fixed one of the three similar replicas) and switches to JW-OS when the network predicts *sat* probability below a threshold of 0.3^2. We then compared the new hybrid guidance with the heuristic JW-OS without the 1000 step restriction. JW-OS was selected on the basis of Experiment 1. The experiment shows that the hybrid approach is faster in terms of number of steps in a significant majority of cases, both when used with DPLL (Figure 4 Left) and with CDCL (Figure 4 Right).

---

[2]We leave further parameter and model searches as a topic which should be considered in the full version of this paper.
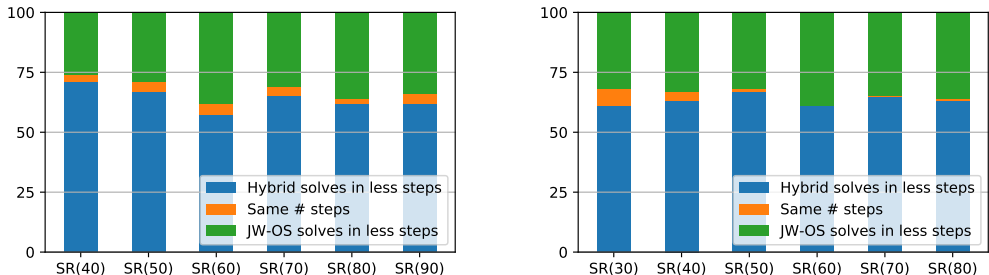
Figure 4: *Left: Comparison of Hybrid (ours) and JW-OS as heuristics in DPLL. We measure the performance of each method according to the number of steps required to find a solution for a given SAT instance. A method wins if it solves a given instance in a smaller number of steps. The blue bar reflects the percentage of formulas where the Hybrid (ours) method won, the green bar means that JW-OS won, and the orange bar means that there was a draw. Right: the same for CDCL.*

Experiment 3: an ablation for the attention mechanism. From experiments presented in Figure 5 follows that in most cases attention improved evaluation metrics by a significant margin. Only in the case of $SR(30)$, level 20 attention degraded the model performance. For $SR(50)$, level 40 the metrics with and without attention stayed within the standard deviation of each other.

Reproducibility. For each set of hyperparameters (e.g. $SR(30)$, level 40), we trained five models. We considered a model not correctly trained if adding it to the set of models raised standard deviation of the losses above 1, see Table 1. We excluded such models (up to 2 models out of 5 for a set of hyperparameters) from further comparisons and left the question of
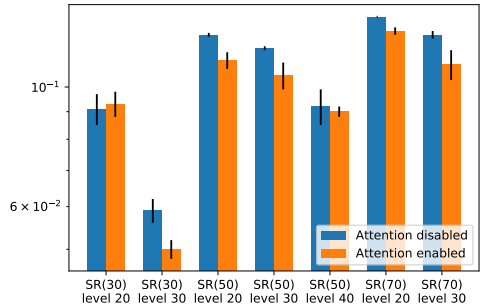

Figure 5: *Comparison of policy error with and without attention. The presented values are mean and standard deviation over 3-5 trained models calculated on the evaluation set.*

stability of training as a topic of further investigations. The code including hyperparameters is published at `https://bit.ly/neurheur`. Our code is based on TensorFlow (AAB+15). It uses a CDCL implementation by (Zho18). We access MiniSat through PySAT interface (IMM18).

5 CONCLUSIONS AND FUTURE WORK

In this work we have shown three experiments confirming that SAT-solving can be augmented by neural networks. The message-passing architecture augmented by attention performs competitively comparing with standard heuristics when evaluated on relatively large propositional problems, including problems with more than a hundred variables (see Section 4). From the ablation presented in Experiment 3 follows that the message-passing architecture that uses the attention mechanism overall performs better then the same architecture without attention and we attribute it to a selective acceptance of incoming messages made possible by the attention mechanism. We believe that using an appropriately large computing infrastructure the learning process can be extended to more complex examples and that in the near future parallelization combined with a variant of the message-passing architecture can be used to train models which will tackle larger SR problems, and possibly SAT problem classes currently beyond the reach of SAT-solvers. As a future step we consider extending our improved heuristics so that a neural network would be able to control other aspects of the SAT solver behavior, like restarting and backtracking. Eventually, other prediction targets, including expected number of steps, may be beneficial. Once we exhaust the pool of available supervised data it would be interesting to apply reinforcement learning methods, including methods recently presented in (KUMO18). In this work we focus on the number of steps of the algorithm rather than execution time. Moving the main loop of DPLL or CDCL to a tensor computation graph would be a step towards making the algorithms more competitive in terms of the execution time.

REFERENCES

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ACKS16] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles A. Sutton. Learning continuous semantic representations of symbolic expressions. *CoRR*, abs/1611.01423, 2016.

[BHB+18] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[ESA+18] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? *CoRR*, abs/1802.08535, 2018.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KUMO18] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olsák. Reinforcement learning of theorem proving. *CoRR*, abs/1805.07563, 2018.

[LOM+18] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh. Machine learning-based restart policy for CDCL SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, pages 94–110, 2018.

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.

[MS99] Joao Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *EPIA*, 1999.

[SLB+18] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018.

[SS18] Taro Sekiyama and Kohei Suenaga. Automated proof synthesis for propositional logic with deep neural networks. *CoRR*, abs/1805.11799, 2018.

[VCC+17] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

[WTWD17] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. *CoRR*, abs/1709.09994, 2017.

[Zho18] Zhang Zhongwei. Simple SAT solver with CDCL implemented in Python. `https://github.com/z11i/pysat/`, 2018.