NeuroSAT, A Graph Neural Network-Based Predictor for SAT Problem

by
Sourav Sarker

Supervisor: Dr. Antonina Kolokolova

A Report submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

December 2020

St. John's Newfoundland

Abstract

In theoretical research and practical applications, Boolean Satisfiability (SAT) is the most studied combinatorial optimization problem. Despite the fact that it is a canonical NP-complete problem which is not believed to be solvable faster than by brute-force search, in the last few decades enormous progress has been made on practical SAT solvers that are widely used for software and hardware verification, challenging algebraic problems and many more.

On the other hand, machine learning and deep neural networks are opening new doors to solving data validation problems, risk management, pattern recognition, and almost every problem in the business sector day by day. A natural question is how to harness the power of neural networks to help us solve SAT, and for which classes of instances would this approach give better results than the current heuristics.

This study will explore a Graph Neural Network (GNN) based SAT solver named "NeuroSAT." We looked at a variety of SAT problems to see which of them NeuroSAT can solve, and which instances remain difficult. Overall, we found that NeuroSAT can only handle very small and easily satisfiable instances, and, surprisingly, increasing the number of iterations of its message-passing stage beyond 100 does not necessarily help.

Acknowledgements

I would like to express my gratitude to Dr. Antonina Kolokolova, my research supervisor, for her patient guidance, enthusiastic encouragement and useful critique on this research work.

I also would like to thank Laura Graves and Vineel Nagisetty for their valuable time and support in this project.

Contents

List of Figures

List of Tables

1 Introduction

The Boolean Satisfiability (SAT) problem is determining whether a list of Boolean constraints can all be simultaneously satisfied by *0/1* assignments to the variables, or, equivalently, whether a given propositional formula has a satisfying assignment. Since many real-world problems can be reduced to SAT problem, the subject of practical SAT solver has been attracting a considerable amount of research attention, and numerous heuristics have been proposed and implemented in the past few decades. SAT is an NP-complete problem [Stephen A Cook et al., 1971], which means other problems in NP, in particular constraint satisfaction problems, can all be encoded as SAT instances. Though in the worst case SAT is believed to be exponentially hard, there is a plethora of heuristics attempting to solve the instances occurring in practice. Most modern SAT solvers are based on a variant of Davis-Putnam-Logemann–Loveland (DPLL) backtrack search [Moskewicz et al., 2001] called conflict-driven clause learning (CDCL). CDCL, introduced in the mid-1990s, performs surprisingly well on large industrial benchmarks in spite of NP-hardness of SAT [Mull, Nathan, Daniel, Sanjit 2016].

In contrast, today, machine learning and neural networks (NN) transform our lives, bringing us to the next level in artificial intelligence (AI). By emulating brain cells' function and interconnectivity, trained NN-enabled machines learn, recognize patterns, make predictions in a humanoid fashion, and solve problems in every business sector. However, there is still no precise characterization of which classes of functions various types of neural networks can learn to compute, beyond lower bounds for perceptrons. In particular, it is not clear whether a neural network can learn to determine Boolean satisfiability even for a limited class of instances. Subsequently, a joint venture of Microsoft Research and the computer science department of Stanford University developed a Graph Neural Network (GNN) based SAT solver, NeuroSAT [Selsam, Daniel, et al., 2019].

A Graph Neural Network is a recently developed neural network architecture which incorporates a message passing stage to account for inherent invariances in inputs representable as graphs, in particular invariance under permuting vertices. NeuroSAT architecture combines a message-passing stage with several LSTMs repeatedly re-representing the input, and a final binary classification stage with a deep neural network. See section 2.6 for details of NeuroSAT architecture.

The input to NeuroSAT is an instance of a formula (in DIMACS format) which it then converts into an undirected graph bipartite with literals and clauses as nodes (clause-literal incidence graph). It then creates a Message Passing Neural Network (MPNN), which is used to calculate a "vote" of each literal (whether a literal predicts that the formula is satisfiable or unsatisfiable based on its local information, and with what confidence). Eventually, these votes are aggregated to make a decision whether the formula is satisfiable or not. Initially, literals tend to predict that an instance is unsatisfiable (with low confidence), until they converge on some satisfying assignment, which can then be recovered (with extra work) from the votes.

For training (2.7), NeuroSAT begins by defining variables in the range of *10...40* in a dataset and iteratively adds clauses. Each clause contains *k* variables, where *k* is sampled through *1 + Bernoulli(0.7) + Geometric(0.4)*. This random selection minimizes clauses of length two, which makes the formula easy to solve. From there, *k* variables are sampled without replacement and added to the clause. Each literal is then negated with probability *0.5*, and the clause is added to the formula. After each added clause, the satisfiability is checked with MiniSAT [Sorensson & Een, 2005]. When the formula becomes unsatisfiable, then a satisfiable version must exist with a single literal negated from the final clause c_m. Formulas $c_1, c_2, \ldots, c_m$ (UNSAT) and $c_1, c_2, \ldots, c_m^{\prime}$ (SAT) are added to the training set.

1.1 Our results

We trained NeuroSAT on the distribution described above, and evaluated its performance on instances generated using CNFgen [Lauria, M., et. al., (2020)]. We considered three families of instances: PigeonHolePrinciple, Clique-Colour and random *k-SAT* near satisfiability threshold. Since NeuroSAT does not differentiate between an unsatisfiable instance and one for which it could not find an assignment, for the first two families we only considered satisfiable instances (eg, PigeonHolePrinciple with fewer pigeons than holes). For random *k-SAT*, we had both unsatisfiable and satisfiable instances, and checked whether they are satisfiable using CryptoMinisat5 [Soos, Mate, 2020] SAT solver.

Even for the smallest and easiest instances of PigeonHolePrinciple that we tried, such as *5* pigeons and *11* holes, NeuroSAT could not find a satisfying assignment. Similarly, for clique-colour instances NeuroSAT gave the answer of "None" for all instances in our experiments. Therefore, we focused on random *3-SAT* instances. The largest number of variables for which NeuroSAT finished execution was $n = 318$, with "None" for all instances in spite of all the instances being satisfiable.

Eventually, we limited our attention to random *3-SAT* instances with the number of variables starting at *5*, and *10* to *25* clauses. Even then, NeuroSAT could not reliably determine satisfiability for all instances, and increasing the number of iterations of message passing beyond *100* (to the maximum of *512*) did not seem to help. There, we looked at the number of solved instances for each setting of the parameters and the running times; see section 3.

Based on our results, NeuroSAT is not a viable alternative to state-of-the-art SAT solvers, though it is an interesting proof of concept. Potentially, a better application of neural nets would be classifying instances from some class of benchmarks such as verification instances into easy vs. hard for CDCL-based solvers, with the goal to understand what underlying structure of these instances results in such an impressive performance of CDCL.

2 Background

2.1 Propositional Satisfiability

Propositional satisfiability problems have been around for quite a long time. Nonetheless, the term first came in the 20th century [Justyna Petke et al., 2015]. A proposition or statement is a declarative statement, can only be true or false, but not both. For example:

i) 5 + 5 = 10

ii) Cow flies in the sky

Here the first proposition is true, and the second one is false, but it is impossible to be both for each statement. Any propositional formula is an instance for the boolean satisfiability problem. That is why any propositional satisfiability problem is also commonly known as the boolean satisfiability problem.

A boolean satisfiability problem determines a propositional calculus formula to be $true$ or $false$ if there is an assignment of truth or false values to its variable. The boolean formula is SAT if we assign truth values to the formula's variables so that the formula becomes $true$. In contrast, the Boolean unsatisfiability problem (UNSAT) means all possible assignments to a formula's variables, making the formula $false$. Consider, for example following formula:

$$(p \lor q) \land (p \land \neg q)$$

The formula will be satisfiable for the value of p and q to be $true$ and $false$. Furthermore, the formula can become unsatisfiable if $p\ and\ q$ both are $true$.

2.2 Terminology for Boolean Satisfiability Problem

In the form of the conjunctive normal form (CNF), a boolean logic formula consists of variables, literals, clauses and a combination of three basic logical operations such as conjunction (and), disjunction(or) and negation(not) [Bryant-rich & Barshaw-rich, 2012].

2.2.1 Conjunction

Conjunction is joining two propositions or statements by the English word $"and"$. Symbolically,

$$p \land q$$

The above expression can be read as "$p\ and\ q$"The following truth table shows the expression will only be satisfiable if both p and q are $true$.

p	q	$p \land q$
T	T	T
T	F	F
T	F	F
F	F	F

2.2.2 Disjunction

Again two propositions can also be combined with the word $"or"$ known as disjunction. Symbolically,

$$p \lor q$$

The following truth table indicates the above boolean expression is $false$ when both p and q are $false$; otherwise, the expression is $true$.

p	q	$p \vee q$
T	T	T
T	F	T
T	F	T
F	F	F

2.2.3 Negation

The negation of a proposition can be written as "The proposition is not true or $false$ that ...". Symbolically denoted as $\neg p$.
The truth table is shown below indicates negation p will only be true if p is false; otherwise, it is $false$.

p	$\neg p$
T	F
F	T

2.2.4 Variables, Literals, Clauses & Conjunctive normal form (CNF)

Let a set of n logical variables, $P = p_1, p_2, \ldots, p_n$ where the truth assignment for P would be a function $T : P \rightarrow \{true, false\}$. Each variable p has two literals, p and negation of p. Therefore, $T(p)$ would be $true$ or $false$, respectively, when p is $true$, and the $\neg p$ is $false$. Furthermore, a set of literals forms clause C when it is connected with logical $''or.''$ To construct a conjunctive normal form (CNF), a set or tuple of clauses get connected with logical AND. For example:

Let F is a propositional calculus formula consisting of variables P and clause C. Then we can interpret a proposition as conjunctive normal form (CNF) where for a truth assignment T to set of variables P makes C $true$ if and only if at least one literal $p \epsilon C$ is $true$ for that truth assignment and T satisfies F if and only if every clause becomes $true$. This truth assignment to a variable we call it a satisfying assignment for the solution. Formally,

$$F(u) = def\{C | C \epsilon F, \{p, \neg p\} \cup C = \emptyset\} \cap \{C\ \neg p | C \epsilon F, \neg p \epsilon C\}$$

[Stephen A. Cook and David G. Mitchell (1997)]

2.2.5 CNF Expression File Format and Dimacs

A textual representation of a boolean expression can be presented in the DIMACS-CNF file format. This format was created by the Center for Discrete Mathematics and Theoretical Computer Science in short DIMACS [DIMACS CNF (.CNF) FORMAT - MAPLE PROGRAMMING HELP, 2020].

As we know, DIMACS CNF is a textual format that may start with comments, denoted with the character c. Right after that, a DIMACS file begins with $p < format >$ $< nbvars >$ $< nbclauses >$ where "format" would be always $"cnf"$, $"nbvars"$ and $"nbclauses"$ means number of variables and number of clauses respectively. Let a formula,

$$(p \vee q \vee \neg r) \wedge (\neg q \vee r)$$

In the above formula, we can see there are three variables p, q and r and two clauses $(p \vee q \vee \neg r)$ and $(\neg q \vee r)$. Now, we can encode this into $DIMACS\ CNF$ as follows:

$$\begin{array}{l} c \\ c\ this\ is\ a\ comment \\ c \\ p\ cnf\ 3\ 2 \\ 1\ 2\ {-3}\ 0 \\ {-2}\ 3\ 0 \end{array}$$

2.3 Restriction of SAT & Complexity

The restriction of SAT to instances where all clauses have length *k* is denoted *k-SAT* [Stephen A. Cook and David G. Mitchell (1997)]. A variant of SAT where every clause has length *2*, known as $2 - SAT$, is solvable in polynomial time, whereas *3* is the smallest *k* that makes $k - SAT$ NP-complete [Stephen A Cook et al., 1971]. SAT is NP-complete, that is, all decision problems within complexity class NP can be solved in polynomial time if we make it possible to solve the SAT problem in polynomial time. However, unfortunately still, there is no such algorithm that can efficiently solve the SAT problem, but if once we can solve (most often takes exponential time) an SAT problem, that can be verifiable in polynomial time.

2.4 SAT Solvers

As we knew earlier, SAT is NP-Complete, and the set of problems under this complexity are so generic that we do not need to take care of each problem separately. As a result, the practical subject of SAT solvers has received a remarkable amount of attention from researchers. In 2007, we saw that heuristic SAT algorithms can solve an SAT problem containing tens of thousands of variables and formulas with millions of symbols [Ohrimenko, Olga; Stuckey, Peter J.; Codish, Michael et al., 2007]. That makes a significant improvement in constraint solving in artificial intelligence, software verification, circuit design, operations research and many more.

There are many publicly available SAT solvers (e.g. MiniSAT, CryptoMiniSat, Glucose, Sat4j etc.) because it is quite challenging to know what benefits we will get from which optimizations. In 2019, researchers from Stanford University and Microsoft Research combinedly developed a new neural network-based SAT solver named NeuroSAT.

2.5 NeuroSAT and Graph Neural Network (GNN)

NeuroSAT is a binary classifier for boolean satisfiability problems. It uses a deep and message passing neural network to predict those problems' satisfiability and employs random SAT problems to train the network. For each SAT problem, NeuroSAT with a single bit of supervision can designate whether that SAT problem is satisfiable or not and can automatically decode the solution from the network's activations.

2.6 Network Architecture

An SAT problem is like character strings, or in other words, it has a syntactic structure, and a standard sequence model like RNN could be applied. However, due to some invariances (e.g. permutation invariance and negation invariance) of boolean propositional logic that do not impact an SAT problem's satisfiability, there is a big chance of ignoring these invariances in the standard sequence model during learning about an SAT instance structure.

Therefore in NeuroSAT, the idea of Graph Neural Network (GNN) was exercised and constructed a Message Passing Neural Network (MPNN) so that the network architecture can enforce the SAT problem invariances [Scarselli, et al., 2009]. Enforcing those invariances in the network also helps not to learn from scratch about propositional logic. However, the GNNs are not a standard paradigm, it is quite recent.

2.6.1 Message Passing Neural Network (MPNN)

To develop the Message Passing Neural Network (MPNN), an SAT problem was plotted as an undirected graph where every single literal and every clause acts as a node for the graph. There would be two types of edges: one type of edge is for literals and clauses; another type of edge is between literals and its complements. In the constructed graph, each node sends messages to its connected nodes at every timestamp along the edges. Here, a message, also known as vector embedding, is some collection of floating-point numbers for a node representing something, hopefully, meaningful about that node. In every iteration of timestamp, there are two stages of message passing:

2.6.1.1 First Stage

In the first stage, literals pass messages to the corresponding clauses in every timestamp *t*. All the literals' vector embeddings of the SAT problem get stored in L^t to be a matrix R^{2n*d}. In other words, every row contains one of the literal embeddings.

A learned neural network, L_{msg} takes this matrix of embeddings and computes some other vector of floats representing a message. Then a clause will sum these incoming messages with a bipartite adjacency matrix M^T and also consider its previous state C_h^t to update its own embedding using RNN or LSTM named C_{update}.

$$C^{(t+1)}, C_h^{(t+1)} \leftarrow C_{update}([C_h^{(t)}, M^T L_{msg}(L^t)])$$

[Selsam, Daniel, et al., 2019]

Figure 1 shows an illustration of the first stage of message passing.

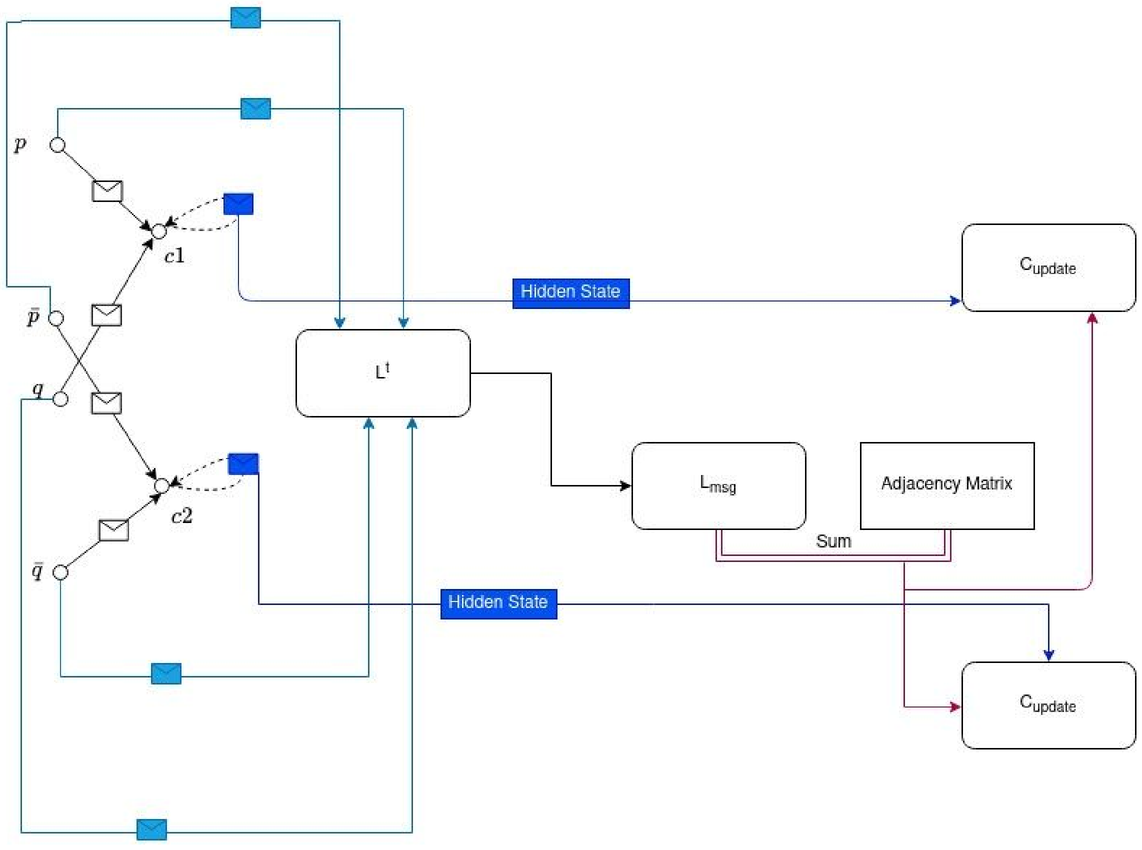

Figure 1: First step of message passing to update clauses' embeddings.

2.6.1.2 Second Stage

In this stage, now clauses pass messages to the corresponding literals. A multilayer neural network C_{msg} takes the matrix of all the clauses' vector embeddings, which outputs each clause's messages. Then a literal will sum the incoming messages with an adjacency matrix, flips each row of L^t with a row corresponding to its complement literal by a $''Flip''$ operator and also takes its hidden state L_h^t. Lastly, that literal updates own vector embedding by

2-layer-norm LSTM L_{update}. At every t iteration, the equation to update each clause's embedding:

$$L^{(t+1)}, L_h^{(t+1)} \leftarrow L_{update}([L_h^{(t)}, Flip(L^t), M^T C_{msg}(C^{(t+1)})])$$

[Selsam, Daniel, et al., 2019]

Figure 2 gives a representation of the second stage of MPNN.

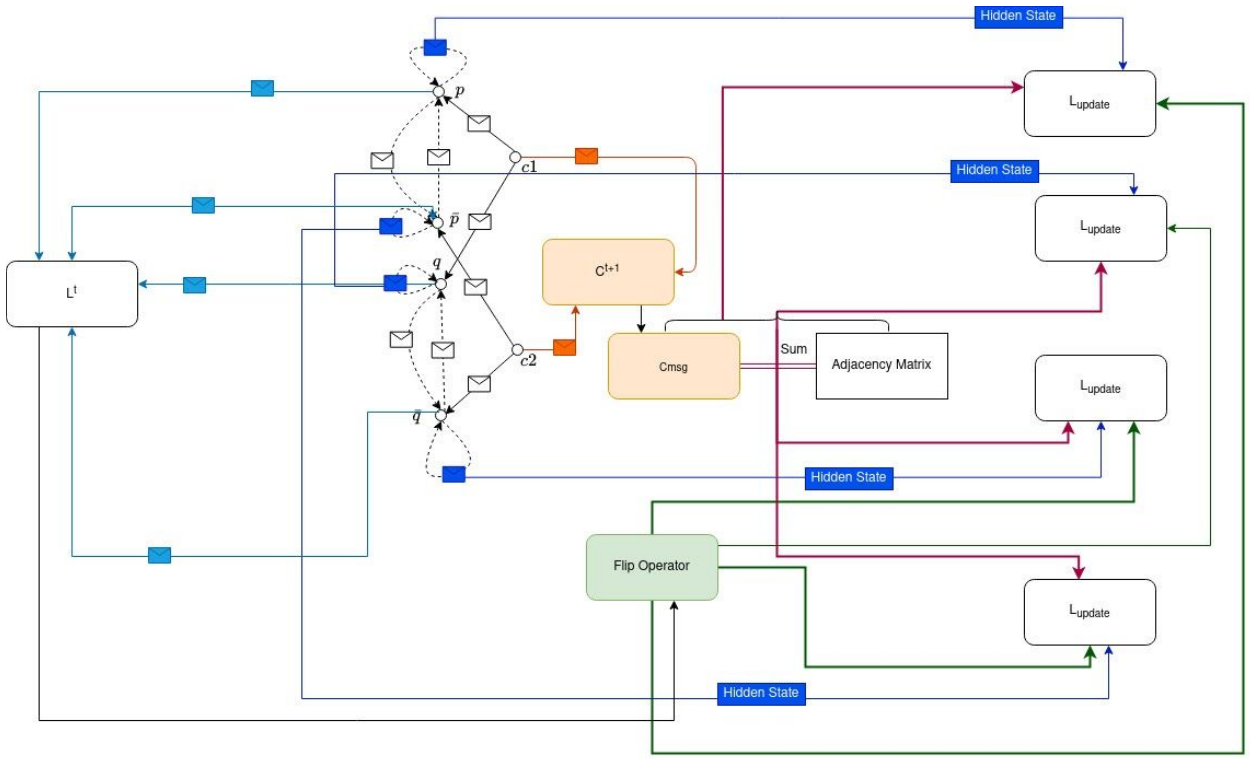

Figure 2: Second step of message passing to update literals' embeddings.

2.6.2 Voting for Result

After T iterations of message passing where $T = t_1 + t_2 + .. + t_n$, another multilayer perceptron L_{vote} takes the matrix of literals' updated embeddings L^T and maps each literal embedding into a single scalar that can call it as a literal vote. Then we average all the literals' votes and get a logit, which is just a single scalar for the entire network. Now predict the satisfiability using the calculated logit by applying a sigmoid function over the mean.

Figure 3 shows the logit vs probability scale where if logit is $-\infty$, then the probability is *0%*, whereas if logit $+\infty$, then probability *100%*.

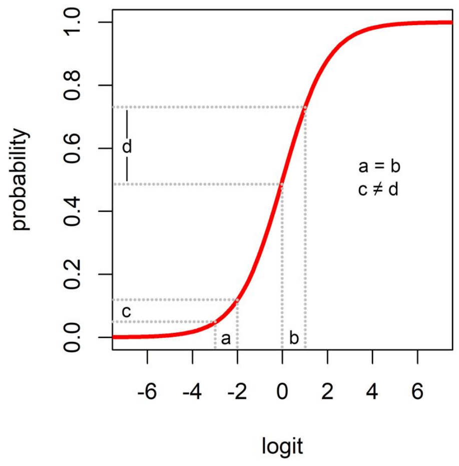

Figure 3: Logit vs Probability scale [Jones, Gavin & Peery, M., et al., 2019]

Figure 4 describes the architecture of calculating scalar vote for each literal to predict an SAT problem's satisfiability.

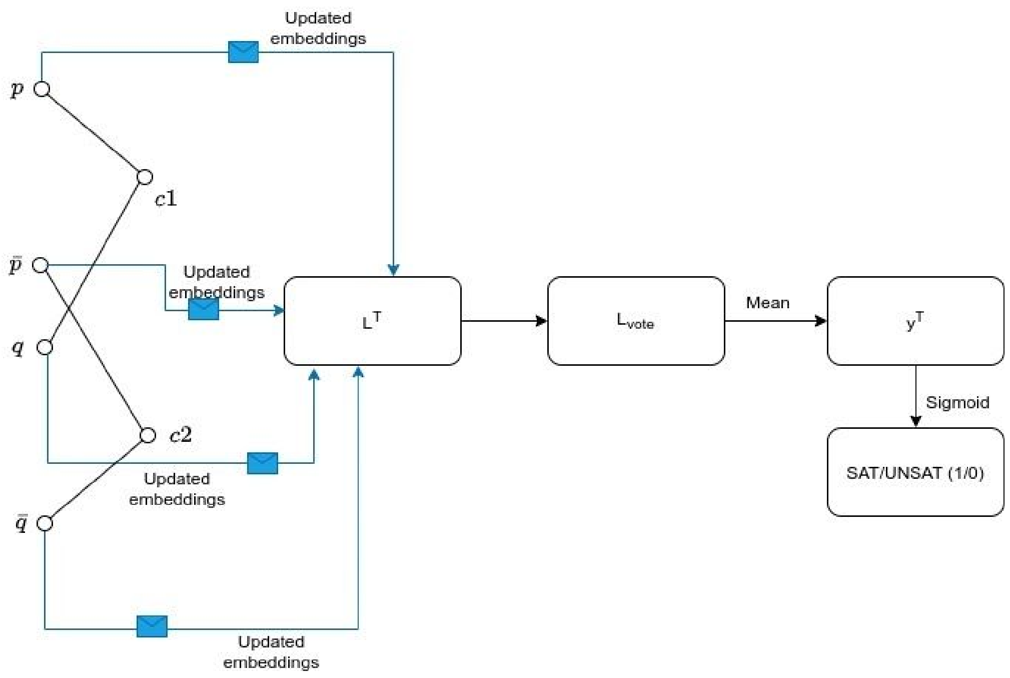

Figure 4: After the *T* iteration, applying a sigmoid function over calculated the mean of the literals' vote, NeuroSAT predicts the probability of SAT instances.

2.7 Distribution for Training Data

Some particular criteria have been considered to generate data for training the NeuroSAT network because some problem distributions are easy to classify based on their superficial properties (e.g. crude-statistics). As a result, there is a chance for the network model not to learn anything about other distributions. To create the data distribution for training, randomly generated SAT problems have been used where problems come in pairs: one SAT; one UNSAT and both differ by negating a single literal in a single clause.

Now for sampling one pair of problems, starting with n variables and no clause, then keep adding uniformly distributed clauses and verifying with MiniSat [Sorensson & Een, 2005] to check satisfiability. If it is UNSAT, then flip a single literal in the final clause to make it SAT. Lastly, return the pair - before and after flipping the literal. Therefore the number of clauses for an SAT instance is determined by the number of sampled clauses before the instance gets UNSAT.

For training, the whole distribution is split into multiple or several problems in different batches. Each batch contains *12,000* nodes (literal and clause). After *26* iterations of message passing on each problem from the distribution, the network model has been trained with cross-entropy loss using backpropagation to minimize the training dataset's classification error [Selsam, Daniel, et al., 2019].

2.8 Prediction and Satisfying Assignments

After training by sampling the number of variables from the uniform distribution between *10* and *40*, the NeuroSAT researcher claims that their model is *85%* accurate in classifying an SAT problem while testing on a dataset with *40* variables. However, initially training some LSTMs, NeuroSAT performs poorly (at most only *50%*), and to analyze this accuracy score, all the scalar literals' votes L_{vote} at each timestamp were plotted (as shown in Figure 5) of the test dataset. Figure 5 depicts the scalar literals' votes in pairs (literal and its negation) of *24* iterations of message passing, where the colours red, blue and white mean SAT, UNSAT and unsure, respectively. The initial red and blue effects are because of the initialization vectors, and after a few iterations of these art effects, every literal vote a UNSAT with low confidence (light blue). Then after a few iterations of message passing, suddenly, literals start to vote SAT extremely high confidence (dark red). For the problems which are truly SAT, NeuroSAT behaves similarly after the phase change, but for a true UNSAT problem, it always predicts UNSAT with low confidence for any number of iterations. This means NeuroSAT's prediction for the UNSAT problem is always right, and it searches for a certificate of satisfiability for the predicted SAT problem [Selsam, Daniel, et al., 2019].

Note that after the phase change, the reason NeuroSAT became confident in predicting an SAT problem as SAT was because it had detected a satisfying assignment and knew the problem was satisfiable. If we look more closely at Figure 5, when all votes of the literals converge at *24* message passing iterations, we notice if a literal votes high confidence, then its

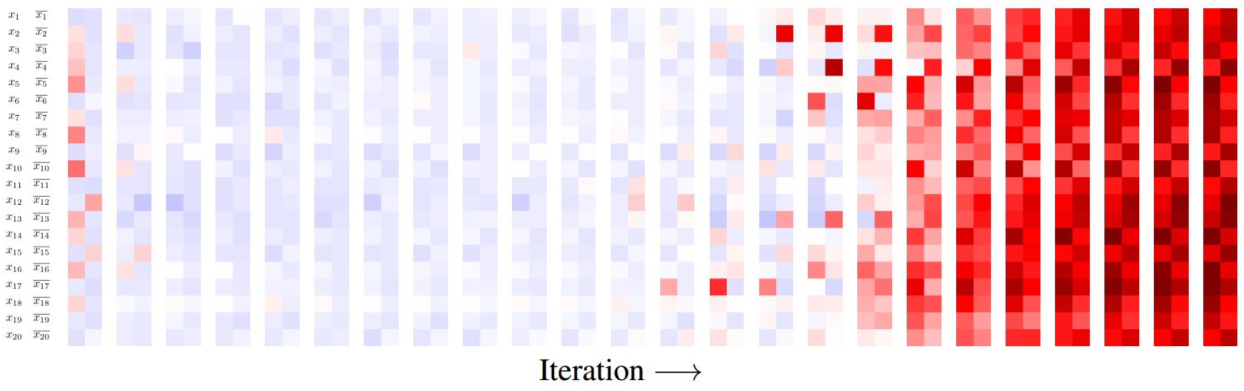

Figure 5: The sequence of literals' scalar votes from L^1 to L^{24} upto 24 iterations. In other words, the above figure illustrates the scalar literals' votes in pairs (literal and its negation) of 24 iterations of message passing, where the colours red, blue and white mean SAT, UNSAT and unsure, respectively. The initial red and blue effects are because of the initialization vectors, and after a few iterations of these art effects, every literal vote a UNSAT with low confidence (light blue) [Selsam, Daniel, et al., 2019].

complement votes with higher confidence and vice versa. That indicates a bit pattern for each variable, and the NeuroSAT researcher thought this bit might help to decode the satisfying assignment. Later it turns out that these bits usually encode satisfying assignments. Nevertheless, it is not a reliable way to decode the satisfying assignments because it has been seen that literals that vote SAT with lower confidence sometimes are the ones true in solution [Selsam, Daniel, et al., 2019].

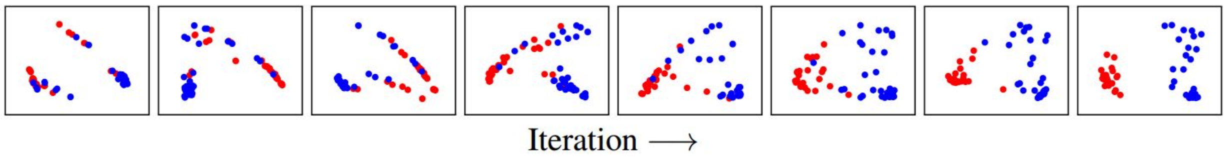

Figure 6: A 2-d PCA projection of high dimensional embeddings, we can notice after phase transition at *12* iterations of message passing the satisfying assignment starts to get seperated and form two distinct clusters after network convergence at *26* iterations. Here blue and red dots indicate literals that are $true$ or 1 and $false$ or 0 respectively [Selsam, Daniel, et al., 2019].

The most reliable way to decode a solution is using two clustering methods because only after the phase transition (after *12* iterations as shown in Figure 5) the satisfying assignments start to get separated and form two complete distinct clusters after network convergence at *26* iterations. Figure 6 shows an illustration of two dimensional PCA embeddings [Selsam, Daniel, et al., 2019].

3 Experiments

3.1 Methodology

NeuroSAT was trained on their distribution of random formulas differing in one literal, as described above, retraining for each experiment. We then tested it on formulas produced by CNFgen by [Lauria, M. (2020)]. The experiments were run in Google Colab using Python [Link] and [Link].

3.2 Satisfiable Versions of Hard Tautologies and medium-size random k-SAT

We generated three classes of formulas using CNFgen of [Lauria, M. (2020)]:

1. **Pigeonhole Principle:** It states that, if *m* number of holes contains *n* number of pigeons where *n*>*m,* there exists at least one hole containing more than one pigeon. For the experiment, some satisfiable pigeonhole principle formulas with a range of number of holes *(5 to 10)* and number of pigeons *(11 to 20)* were generated. But NeuroSAT could not find any solution for these SAT instances.
2. **Clique-colouring:** In the clique-coloring formula, a graph with *n* vertices contains a clique (complete subgraph) *k* as well as a coloring size *c*. The *k* = *c* and *k* = *c* + *1* formula makes a formula satisfiable and unsatisfiable, respectively. To inspect the performance of NeuroSAT on clique-colouring satisfiable instances, we generate some formulas with *5* to *10* vertices and the same size of clique and colour *(2 to 5)*. However, in our examination, NeuroSAT was not able to identify any solution to these instances.
3. **Random *k-SAT*:** Here, the generator creates a CNF with *m* clauses on *n* variables where every clause has width *k* by randomly sampling, for each clause, *k* out of *n* variables, then negating each variable in the clause with probability ½. First, we generated instances with *k = 5, n = 201-347, m = 1409-1635,* and *k*=*7, n*=*214-347, m = 1162-1755*. However, for instances with *n*>*318* NeuroSAT did not even terminate (after 15 hours), and for other instances in this set it did not find any satisfying assignments. By contrast, tCryptoMiniSAT took less than *0.5* seconds to solve these instances. Therefore, for the experiments with NeuroSAT we had to restrict our attention to very small instances: random *3-SAT* with *n = 5*. This is discussed in the next section.

3.3 Small Random 3-SAT

For this set of experiments, we produced small *3-CNF* random formulas with the Random *k-CNF* method of $CNFgen$ [Lauria, M. (2020)]. We selected *n*=*5* and used CNFgen to generate 10 instances for each number of clauses m from *10* to *25*; this gave us 150 instances.

There are n choose k ways to select variables for a k-clause, 2^k possible ways to assign polarities (positive or negative) to them. Hence, the number of possible clauses for *n*=*10*, *k*=*3* would be $2^3 * 10 = 80$, which gives 80 choose *10* possible formulas with *10* clauses.
As the clause-to-variable ratio increases, a random formula is more likely to be unsatisfiable.

[Table 1] represents an example of *4* different formulas for $(n = 5, m = 10)$ pair.

4 possible formulas for *k=3, n=5, m=10*			
p cnf 5 10	p cnf 5 10	p cnf 5 10	p cnf 5 10
-3 -4 5 0	1 -2 -5 0	2 -4 5 0	2 3 5 0
-2 -3 5 0	-1 -2 -3 0	2 -3 5 0	-3 4 -5 0
-1 2 -3 0	1 -2 4 0	-3 -4 5 0	3 4 -5 0
2 -4 -5 0	-2 3 -4 0	-1 3 5 0	-1 -3 4 0
1 2 -3 0	-3 4 -5 0	-2 4 -5 0	-1 4 5 0
-2 -3 -5 0	-1 -2 4 0	-1 -2 -5 0	2 -3 -4 0
-1 -2 -4 0	-1 2 -4 0	1 2 4 0	-1 3 -5 0
-1 3 -4 0	-2 -4 5 0	-1 2 5 0	-2 -3 -4 0
-1 4 5 0	-1 -2 -4 0	-2 -4 -5 0	2 -4 5 0
-1 -3 -4 0	-1 2 5 0	1 -2 4 0	1 2 5 0

Table 1: Four different random formulas in DIMACS format for $(n = 5, m = 10)$ pair created by $CNFgen$

3.4 NeuroSAT Accuracy

As we have learned earlier, NeuroSAT is a message passing based SAT solver as it uses the Message Passing Neural Network (MPNN), so we have run it with different number of iterations *[26, 50, 100, 200, 512]* of message passing to compare the performance. Figure 7 illustrates the number of detected SAT problems as SAT and “None”. Note that while there are more instances predicted as SAT when the number of iterations increases from *26 to 50* and then to *100*, there is no more improvement from *100 to 200 to 512*, even though there are many more satisfiable instances than what NeuroSAT detected.

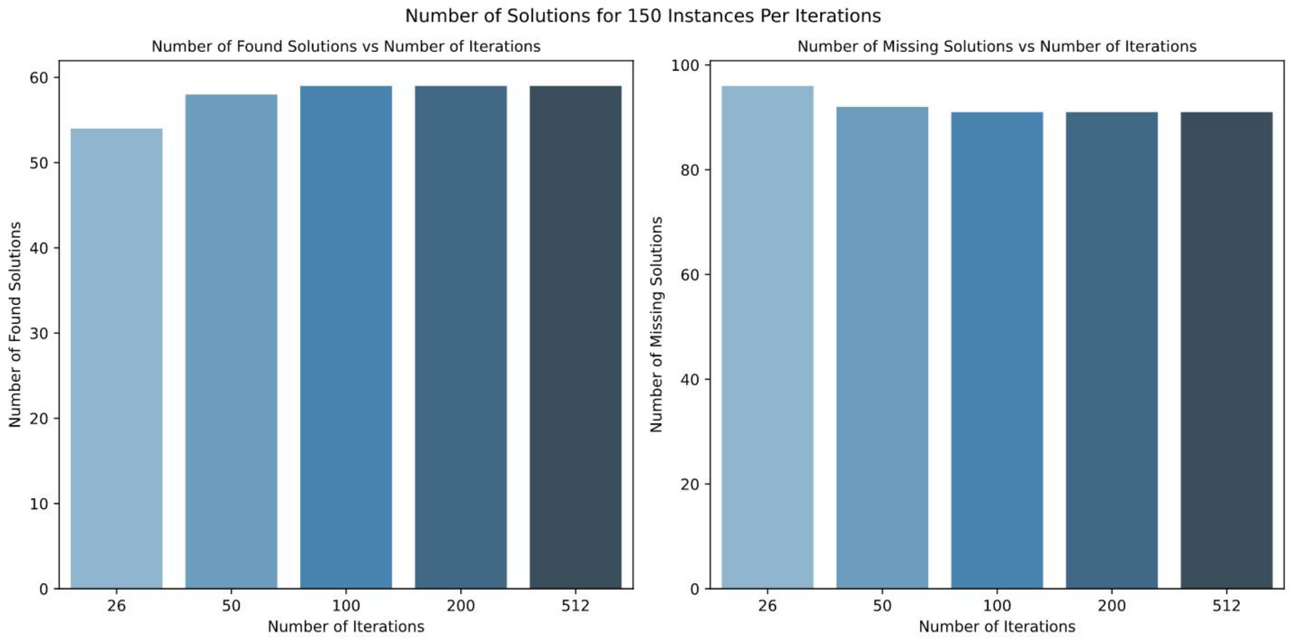

Figure 7: The number of detected SAT problems as SAT and "None" (No Solution) by NeuroSAT for *26, 50, 100, 200, 512* iterations.

More specifically, we verified satisfiability of these generated random instances with CryptoMinisat5 [Soos, Mate (2020)] which is a complete SAT solver (that is, unless it times out, it correctly returns "SAT" or "UNSAT"), and found that out of *150* SAT instances, *126* instances are satisfiable with the remaining *24* are unsatisfiable. Out of them, NeuroSAT executed for *26* and *50* iterations, it has predicted *54* and *58* SAT formulas as satisfiable respectively out of *126* satisfiable instances and for *100, 200, 512* iterations, NeuroSAT predicted *59* instances as SAT. Note that in this experiment NeuroSAT never incorrectly identified an unsatisfiable formula as satisfiable (see Figure 8).

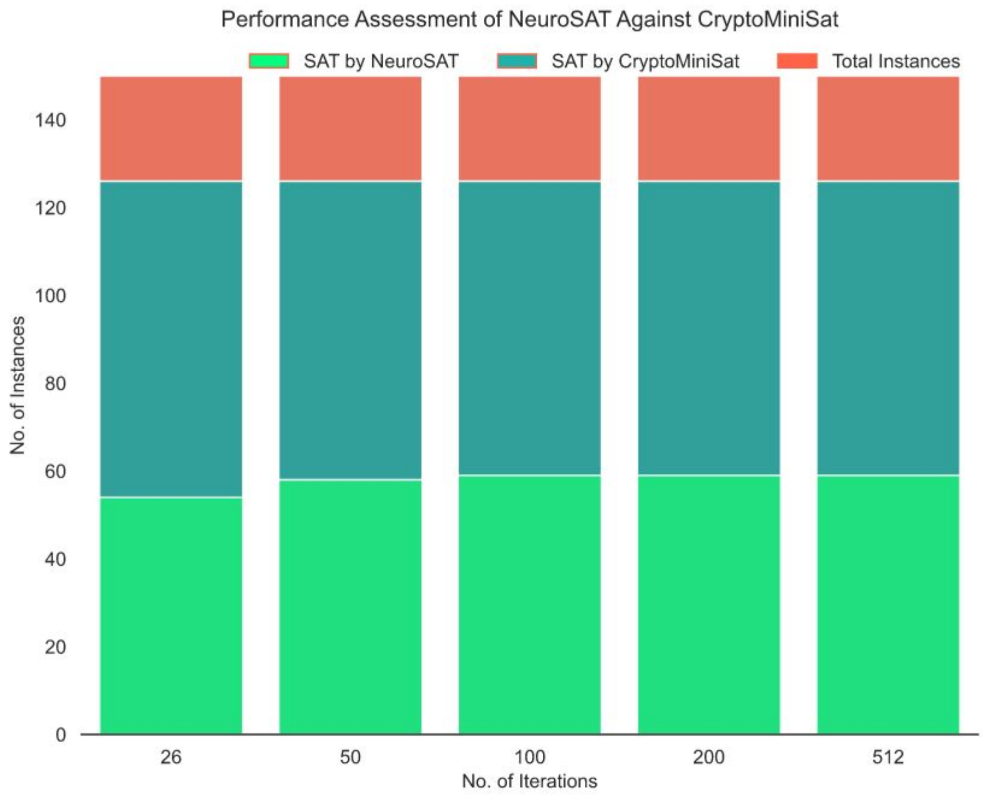

Figure 8: Out of *150* SAT instances, *126* instances are satisfiable the rest *24* are unsatisfiable (by Cryptominisat5)—then we compared that with NeuroSAT's accuracy of predictability. For *26* and *50* iterations, it has predicted *54* and *58* SAT formulas as satisfiable respectively out of *126* satisfiable instances and for *100, 200, 512* iterations, NeuroSAT predicted *59* instances as SAT. For the rest of the instances, it predicted UNSAT or "None".

3.5 NeuroSAT Running time

NeuroSAT's performance in solving an SAT problem is not competitive with a state-of-art SAT solver; thus, it is expected that it will take a longer time to find a solution. Please note again, NeuroSAT is built on message passing architecture. Hence now we will investigate iteration-wise accomplishment time in solving SAT instances. Figure 9 displays the total number of execution times per iteration, where for *200* messages passing iterations, it takes less time and an almost similar time for *50* and *512* iterations.

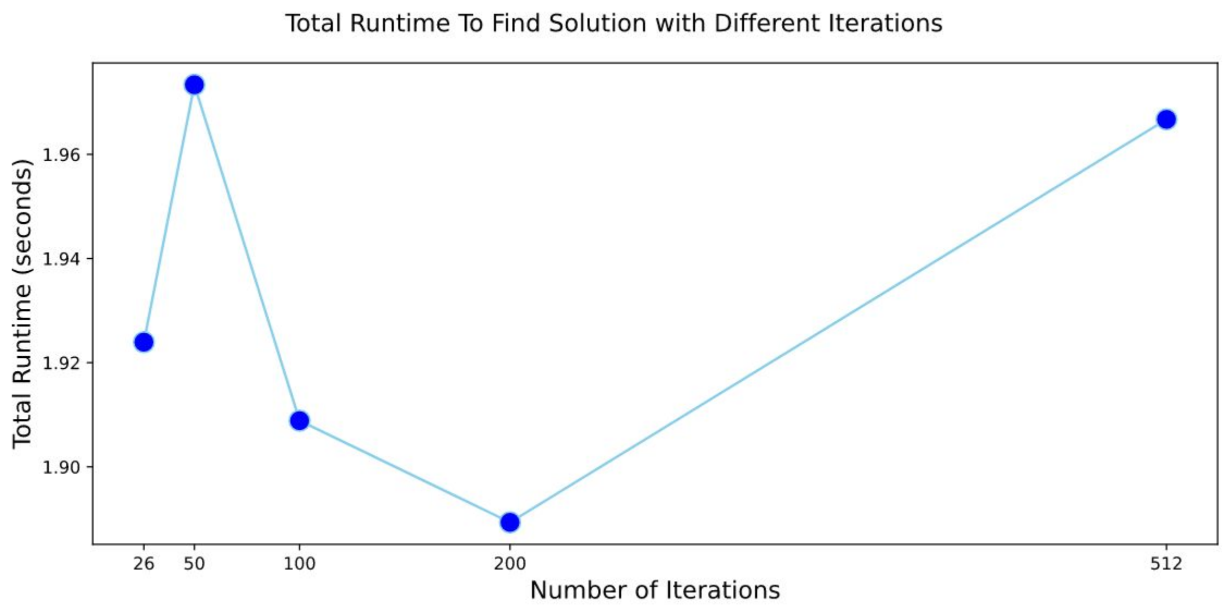

Figure 9: Aggregated execution times for the iterations *26, 50, 100, 200 & 512*. For *200* messages passing iterations, NeuroSAT takes less time and whereas consumes maximum but almost similar time for *50* and *512* iterations.

Now let us break down each iteration's total execution time and find out for every iteration where NeuroSAT spends most. The *150* uniformly random generated instances are verified by CryptoMinisat5 and among those instances *126* are satisfiable, *24* are unsatisfiable. It is interesting to observe that for *26, 50, 100, 200, 512* iterations, NeuroSAT spends longer time to recover a solution for the CryptoMinisat5 tested satisfiable instances but ultimately fails. Table 2 exhibits data regarding that.

Iterations	Total Execution Time For NeuroSAT(seconds)	NeuroSAT prediction	CryptoMinisat5 Prediction	No. of Instances
26	16.73	SAT	SAT	54
	44.28	No Solution	UNSAT	24
	131.43	**No Solution**	**SAT**	**72**
50	29.62	SAT	SAT	58
	43.47	No Solution	UNSAT	24
	124.34	**No Solution**	**SAT**	**68**
100	27.15	SAT	SAT	59

	43.29	No Solution	UNSAT	24
	120.56	**No Solution**	**SAT**	**67**
200	26.86	SAT	SAT	59
	42.44	No Solution	UNSAT	24
	119.7	**No Solution**	**SAT**	**67**
512	27.93	SAT	SAT	59
	44.71	No Solution	UNSAT	24
	124.05	**No Solution**	**SAT**	**67**

Table 2: For the truly satisfiable instances, NeuroSAT spends a longer time to predict those instances as SAT but fails (bold text).

Furthermore, it will also be appealing to observe at which pair of *n* and *m*, NeuroSAT consumes a higher amount of time for a solution. Figure 10 represents a trend of the NeuroSAT's iteration-wise average accomplishment time to each variable-clause *(n, m)* pair instance. Here we can notice that the running time only correlates with the instance size. For the instances *(n= 5, m = 24)*, it employs the highest amount of time and takes less time in the clauses' eventual deduction.

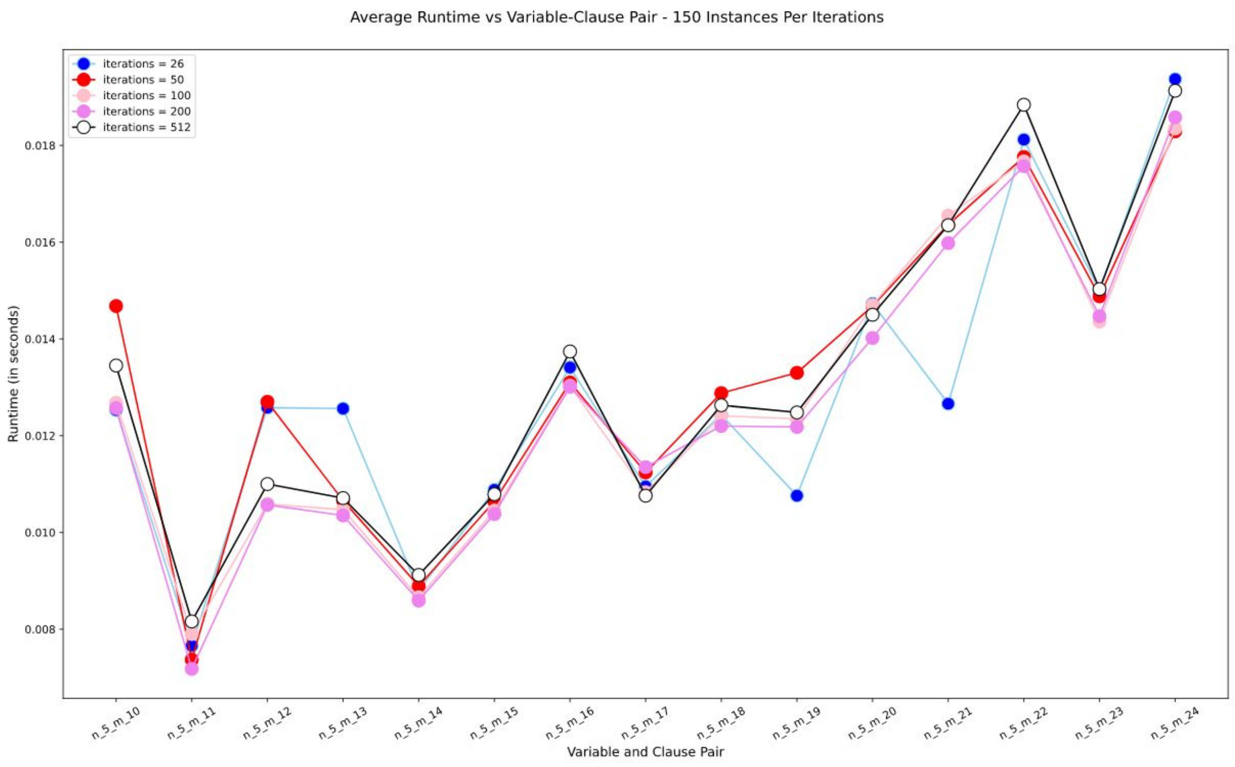

Figure 10: NeuroSAT's iteration-wise average runtime only correlates with instance size. For the instances *(n = 5, m = 24)*, it employs the highest amount of time and takes less time in the clauses' eventual deduction.

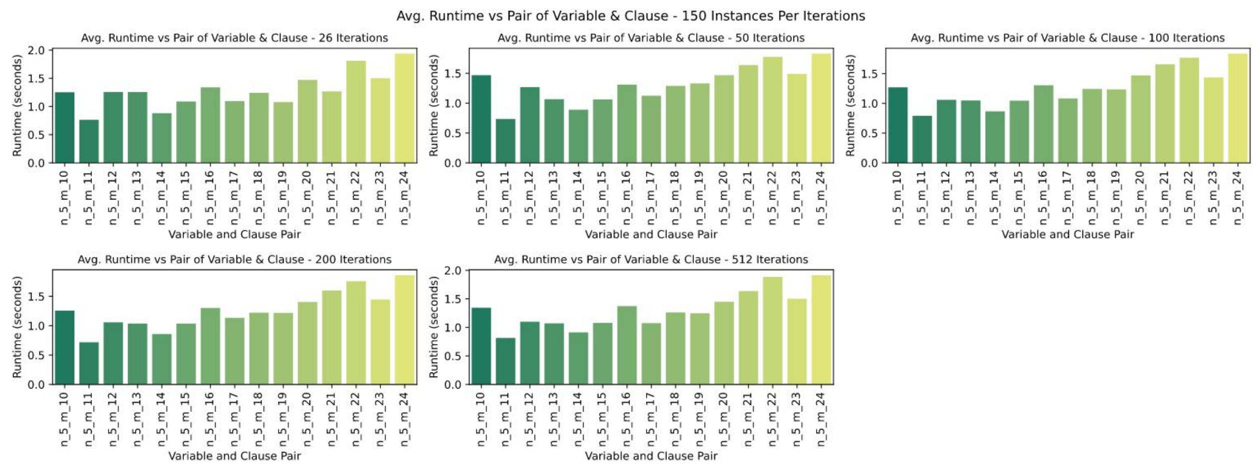

Figure 11: Execution time distribution for each pair of *n* and *m* for every iteration of message passing.

Figure 11 shows another angle of visualization of above explanation (Figure 10) but separately (per iteration) where it is surprising to note that for instances with *n=5, m=23* pairs, NeuroSAT takes less time to find out the solutions for those instances compared to its neighboring pairs. After our investigation, it has seen that there are *30* satisfiable instances out of all instances of this pair and from these satisfiable instances, NeuroSAT predicts *15* of them

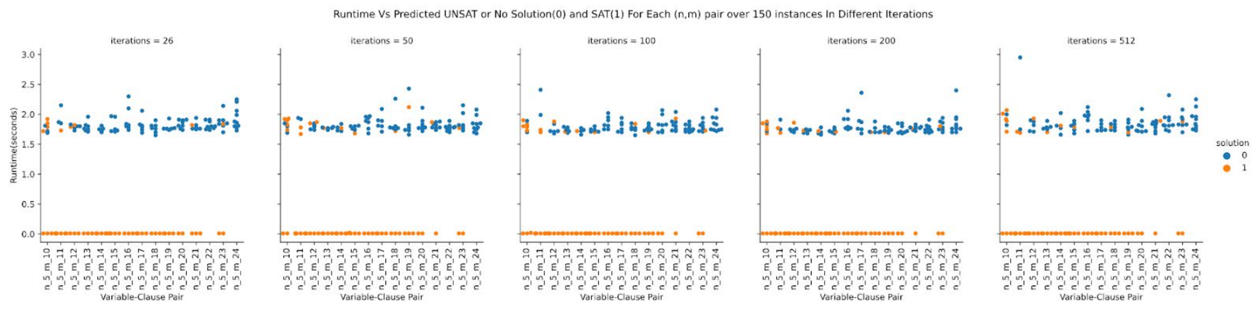

Figure 12: Iteration-wise the execution times break down of NeuroSAT predictions for the test every *n-m* pair of instances. Here orange points indicate the correct guess of satisfiable by NeuroSAT. Whereas the blue points (false negative or true negative) represent either it found no solution for those instances or these are unsatisfiable instances. Here we can notice the prediction time for orange points (true positive) is almost similar.

correctly. It is worthy of mentioning that NeuroSAT almost always takes minimum time (e.g. 0.1 seconds) to find satisfying assignments for satisfiable formulas (if it can predict correctly). Figure 12 depicts iteration-wise the execution times break down of NeuroSAT predictions for the test every *n-m* pair of instances. Here orange points indicate the correct guess of satisfiable by NeuroSAT. Whereas the blue points represent either it found no solution for those instances or these are unsatisfiable instances. Here we also can notice the prediction time for orange points is almost similar.

4 Conclusion

This project explores a Graph Neural Network(GNN) based SAT solver, NeuroSAT, and analyses its accuracy and performance in comparison to a state-of-the-art SAT solver. In our experiments, we have seen that it is capable of detecting satisfying assignments for SAT instances. However, out of 126 CryptoMinisat5 verified SAT instances, it only finds satisfying assignments for slightly less than *60* easy instances. It is quite impossible to distinguish between all unidentified satisfiable instances and instances identified as unsatisfiable from the output, which makes NeuroSAT an unreliable SAT solver for medium and harder instances.

Another downside of NeuroSAT is that its workings are very much a black box. For example, during message passing each node in the graph is associated with a vector of floating point numbers called embeddings, but there is no intuition as to what these embeddings represent and what sort of information is recorded as these numbers. It does not help that there is not much explanation for the choice of the two multilayer perceptrons that are used for estimating literal-clause messages as well as LSTMs used in updates. Lastly, it is also a bit confusing why and how all the literals start to vote SAT (red colour) as shown in Figure 5.

In 2008, Devlin and O'Sullivan attempted to use Random Forest for classifying SAT problems in their experiments. They got more than *90%* accuracy for difficult large industrial instances as well as the instances (random *k-SAT*) like we tried on for NeuroSAT [(Devlin and O'Sullivan 2008)]. That establishes a question regarding the significance of NeuroSAT.

However, NeuroSAT is a binary classifier, and from our experiments, it has been observed that it struggles to solve medium and hard instances, even for some easy instances. We assume NeuroSAT's prediction performance could be improved by extending its implementation with a multi-class classification problem for determining boolean satisfiability hardness where training dataset distribution would be labelled as *0, 1* and *2* for easy, medium and hard instances, respectively. We also can opt for a regression problem where instead of predicting classification, value will be predicted as value. For the regression problem, the training set can be normalized by labelling between *0* and *1* for easiest and hard instances, respectively, and any value in between this range would be a label for medium instances.

Overall, NeuroSAT is an exciting project that brings SAT solving and deep neural network architecture together, and it is an interesting example of an application of the Graph Neural Network architecture.

References

[Justyna Petke et al., 2015] Justyna Petke (2015). Bridging Constraint Satisfaction and Boolean Satisfiability

[Bryant-rich, Donald Ray (Haifa, IL), Barshaw-rich, Diana Eve (Haifa, IL) et al., 2012] Bryant-rich, Donald Ray (Haifa, IL), Barshaw-rich, Diana Eve (Haifa, IL) 2012 PERSONAL VOICE OPERATED REMINDER SYSTEM United States BRYANT-RICH DONALD RAY,BARSHAW-RICH DIANA EVE20120265535
https://www.freepatentsonline.com/y2012/0265535.html

[Stephen A. Cook and David G. Mitchell et al., 1997] Stephen A. Cook and David G. Mitchell (1997) Satisfiability Problem: Theory and Applications

[DIMACS CNF (.CNF) FORMAT - MAPLE PROGRAMMING HELP, et. el., 2020] De.maplesoft.com. 2020. DIMACS CNF (.Cnf) Format - Maple Programming Help. [online] Available at: <https://de.maplesoft.com/support/help/maple/view.aspx?path=Formats/CNF> [Accessed 24 November 2020].

[Stephen A Cook et al., 1971] Stephen A Cook. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing, pp. 151–158. ACM, 1971.

[Ohrimenko, Olga; Stuckey, Peter J.; Codish, Michael et al., 2007] Ohrimenko, Olga; Stuckey, Peter J.; Codish, Michael (2007), "Propagation = Lazy Clause Generation", Principles and Practice of Constraint Programming – CP 2007, Lecture Notes in Computer Science, 4741, pp. 544–558, CiteSeerX 10.1.1.70.5471, doi:10.1007/978-3-540-74970-7_39, "modern SAT solvers can often handle problems with millions of constraints and hundreds of thousands of variables".

[Selsam, Daniel, et al., 2019] “Learning a SAT Solver from Single-Bit Supervision.” ArXiv.org, ICLR 2019, 12 Mar. 2019, arxiv.org/abs/1802.03685.

[Scarselli, et al., 2009] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, 2009.

[Li, et al., 2015] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493, 2015.

[Gilmer, et al., 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212, 2017.

[Sorensson & Een, 2005] Niklas Sorensson and Niklas Een. Minisat v1. 13-a sat solver with conflict-clause minimization. SAT, 2005(53):1–2, 2005.

[Lauria, M., et. al., (2020)] CNFgen formula generator and tools. Retrieved November 12, 2020, from https://massimolauria.net/cnfgen/

[Soos, Mate, et. al., 2020] Soos, Mate. "Wonderings of a SAT Geek | A Blog about SAT Solving and Cryptography." Wonderings of a SAT Geek, 26 July 2020, www.msoos.org/. Accessed 12 Nov. 2020.

[M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, et al., 2001] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: engineering an efficient SAT solver," Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), Las Vegas, NV, USA, 2001, pp. 530-535, doi: 10.1145/378239.379017.

[Jones, Gavin & Peery, M., et. al., 2019] Phantom interactions: Use odds ratios or risk misinterpreting occupancy models. The Condor. 121. 1-7. 10.1093/condor/duy007.

[(Devlin and O'Sullivan 2008)] Devlin, David, and Barry O'Sullivan. 2008. "Satisfiability as a Classification Problem." In *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*.

[Mull, Nathan, Daniel, Sanjit 2016] Mull, Nathan, Daniel J. Fremont, and Sanjit A. Seshia. "On the Hardness of SAT with Community Structure." *Theory and Applications of Satisfiability Testing – SAT 2016 Lecture Notes in Computer Science*, 2016, 141-59. doi:10.1007/978-3-319-40970-2_10.