Predicting Propositional Satisfiability via End-to-End Learning

Chris Cameron, Rex Chen, Jason Hartford, Kevin Leyton-Brown
Department of Computer Science
University of British Columbia
{cchris13, rexctran, jasonhar, kevinlb}@cs.ubc.ca

Abstract

Strangely enough, it is possible to use machine learning models to predict the satisfiability status of hard SAT problems with accuracy considerably higher than random guessing. Existing methods have relied on extensive, manual feature engineering and computationally complex features (e.g., based on linear programming relaxations). We show for the first time that even better performance can be achieved by end-to-end learning methods — i.e., models that map directly from raw problem inputs to predictions and take only linear time to evaluate. Our work leverages deep network models which capture a key invariance exhibited by SAT problems: satisfiability status is unaffected by reordering variables and clauses. We showed that end-to-end learning with deep networks can outperform previous work on random 3-SAT problems at the solubility phase transition, where: (1) exactly 50% of problems are satisfiable; and (2) empirical runtimes of known solution methods scale exponentially with problem size (e.g., we achieved 84% prediction accuracy on 600-variable problems, which take hours to solve with state-of-the-art methods). We also showed that deep networks can generalize across problem sizes (e.g., a network trained only on 100-variable problems, which typically take about 10 ms to solve, achieved 81% accuracy on 600-variable problems).

1 Introduction

NP-complete combinatorial problems are pervasive in many domains, such as planning, scheduling, and networking. The propositional satisfiability (SAT) problem is among the most widely studied of these; indeed, it was the first to be proven NP-complete. It is also used in many applications (e.g., model checking (Clarke et al. 2001) and radio spectrum reallocation (Fréchette, Newman, and Leyton-Brown 2016)) due both to its encoding flexibility and the availability of many high-performance solvers. The SAT problem asks whether a given Boolean expression evaluates to true. SAT is typically represented in conjunctive normal form (CNF), as a conjunction (AND) over clauses, each of which is an disjunction (OR) over Boolean variables, which may optionally be negated. The conjunction and disjunction operators are commutative, so permuting their arguments does not change a problem instance. The two-layer logical structure of CNF is simple to reason about and is used by most SAT solvers.

Over the past two decades, machine learning has been shown to be very useful for making instance-specific predictions about properties of SAT problems (e.g., algorithm runtime prediction (Hutter et al. 2014), algorithm selection (Xu et al. 2008), and satisfiability prediction (Xu, Hoos, and Leyton-Brown 2012)). Perhaps the key drawback of this work is its reliance on hand-engineered features. The computation of most of these features requires between linear and cubic time in the size of the input. It is difficult to assess whether less computationally expensive features would yield similar results, to determine whether important features are missing, and to translate a modeling approach to a new domain. Learning representations from raw problem descriptions via neural networks is a promising approach for addressing these obstacles. There has been recent work in this direction (Evans et al. 2018; Selsam et al. 2019); this work is very interesting from a machine learning perspective, but has tended to focus on problems that are trivial from a combinatorial optimization perspective (e.g., < 1 second to solve by modern SAT solvers).

One of the most widely studied distributions of SAT instances is uniform random 3-SAT at the solubility phase transition (Cheeseman, Kanefsky, and Taylor 1991; Mitchell, Selman, and Levesque 1992). These problems are easy to generate, but are very challenging to solve in practice. Indeed, empirical runtimes for high-performance complete solvers have been shown to scale exponentially with problem size on these instances (Mu and Hoos 2015). Holding the number of variables fixed, the probability that a randomly-generated formula will be satisfiable approaches 100% as the number of clauses shrinks (most problems are underconstrained) and approaches 0% as the number of clauses grows (most problems are overconstrained). For intermediate numbers of variables, this probability does not vary gradually, but instead undergoes a sharp phase transition at a critical point (a clauses-to-variables ratio of about 4.26) where the probability that a formula will be satisfiable is exactly 50%. Using hand-engineered features, past work (Sandholm 1996; Xu, Hoos, and Leyton-Brown 2012) showed that an instance's satisfiability status can be predicted with accuracy higher than that of random guessing. In particular, Xu, Hoos, and Leyton-Brown built the models which we believe represent the current state of the art, and they also investigated whether models could generalize across problem sizes so as to claim to identify "asymptotic" behavior. They thus limited their models to features that preserve their meanings across problem scales (e.g., graph-theoretic properties of the constraint structure and the proximity to integrality of the solution to the linear programming relaxation of SAT, rather than e.g., the solution quality attained by a simple local search procedure in a fixed amount of time). They demonstrated that random forest models achieve classification accuracies of at least 70%, even when trained on tiny (100 variable) problems and tested on the largest problems they could solve (600 variables).

In our work, we investigate the use of end-to-end deep networks for this problem. Combinatorial problems are highly structured; changing a single variable can easily flip a formula from satisfiable to unsatisfiable. We thus believe (and will later show experimentally) that success in this domain requires identifying model architectures that capture the correct invariances. Specifically, we encode raw CNF SAT problems as permutation-invariant sparse matrices, where rows represent clauses, columns represent variables, and matrix entries represent whether or not a variable is negated. The deep network architectures we explore are invariant to column-wise and row-wise permutations of the input matrix, which produce logically equivalent problems. The architectures support arbitrary-sized problems, and both memory demands and forward pass time complexity scale linearly with the number of non-zeroes in the input matrix.

Specifically, we evaluate two different architectural approaches. The first is to compose a constant number of network layers based on trainable parameters; we use the *exchangeable architecture* of Hartford et al. (2018), since it was shown to be maximally expressive and thus generalizes all other such approaches. The second alternative is to use *message passing* networks, which repeatedly apply the same layer for any desired number of steps; we use the neural message passing implementation of Selsam et al. (2019), as it captures the correct invariances. Selsam et al.'s approach was already applied to SAT, but the focus of their work was on the much harder task of decoding satisfiable solutions. For that reason, their work only applied the architecture to small problems, which are trivial for modern SAT solvers.

We evaluated both of these neural network approaches on uniform-random 3-SAT instances at the phase transition, primarily to facilitate comparison with past work. Despite the fact that our models did not have access to hand-engineered features and that they were only able to learn linear-time computable features, we achieved substantially better performance than that of Xu, Hoos, and Leyton-Brown (2012). Specifically, we respectively achieved 78.9% and 79.1% accuracy on average across problems ranging from 100–600 variables for the exchangeable and message passing architectures, compared to 73% accuracy on average for random forests with features. On 600-variable problems (which typically take hours to solve), we achieved $> 80\%$ accuracy with both deep learning architectures. Like Xu, Hoos, and Leyton-Brown (2012), we were able to build models that generalized to much larger problems than those upon which they were trained, even though we were unable to explicitly force learned features to preserve their meanings across problem scales. (However, in this case, the exchangeable architecture showed a clear advantage over the message passing architecture.) For example, we achieved 81% prediction accuracy on 600-variable problems using the exchangeable architecture trained only on (very easy) 100-variable problems.

Overall, our work introduces the first example of state-of-the-art performance for the end-to-end modelling of the relationship between the solution to a combinatorial decision problem and its raw problem representation on a distribution that is challenging for modern solvers. We expect our work to be useful for the solving and modelling of SAT and other constraint satisfaction problems.

The rest of the paper begins with a survey of important related work (Section 2); then, we describe the permutation-invariant neural network architectures that we use to represent SAT instances (Section 3). We apply these architectures to predicting satisfiability (Section 4), and validate them by comparing our results to past work and evaluating their generalization across different instance sizes (Section 5). Finally, we summarize our contributions (Section 6).

2 Related work

Using learning to reason about NP-complete problems Over the past two decades, the combinatorial optimization community has become increasingly interested in using machine learning to make instance-specific predictions about properties of problems. Much work has focused on the problem of predicting how long a solver will take to run (Finkler and Mehlhorn 1997; Smith-Miles and Lopes 2012; Hutter et al. 2014). These methods have shown surprisingly good performance across a wide range of problems, solvers, and instance distributions. Indeed, many still find it counter-intuitive that it is even possible to predict the behavior of algorithms that run in exponential time in the worst case.

Most work for learning to reason about SAT problems builds on the set of features first proposed by Nudelman et al. (2004). They generated 84 features that they considered predictive of solver performance, which they derived from known heuristics (e.g., ratio of positive to negative occurrences of clauses and per variables), tractable subclasses (e.g., proximity to Horn formulae), and other proxies for problem complexity (e.g., statistics of the solution to LP relaxations of the SAT problem, and statistics about the progress of SAT solvers over time-bounded runs). The features vary in computational complexity from trivial (e.g., the size of the problem) to expensive (e.g., the LP relaxation, which is roughly cubic).

These features have subsequently been combined with a variety of machine learning models (Xu, Hoos, and Leyton-Brown 2007), and they form the basis of the random forest models studied by Xu, Hoos, and Leyton-Brown (2012). Hutter et al. (2014) used hand-engineered features to predict per-instance runtimes; features have also been used to build algorithm portfolios, where performance can be improved by selecting different solvers on a per-instance basis (Xu et al. 2008; Lindauer et al. 2015).

Neural network representations for combinatorial problems (Selsam et al. 2019)'s work is the closest in spirit to our own. They learned a neural SAT solver that performs "search" via neural message passing. They focus on the problem of *solving* SAT formulas, but they do so by supervising a message passing architecture with only the satisifiability status of an instance. They therefore cast the problem of determining which variables to instantiate as one of predicting the formula's satisfiability status, and refine this prediction with every recurrent iteration.

Allamanis et al. (2016) and Sekiyama and Suenaga (2018) used TreeNNs to learn end-to-end models for instance-specific predictions of propositional formulae. They considered arbitrary propositional formulae, which lack the same logical invariances of formulae in CNF, and they considered problems with at most 50 variables.

Loreggia et al. (2016) created fixed-size representations for SAT instances by converting CNF representations to 128×128 images and applying a convolutional neural network. Although this representation is not invariant to variable or clause permutations, the resulting algorithm selector outperformed the best individual solver (but still fell far short of methods based on hand-engineered features).

Both Li, Chen, and Koltun (2018) and Selsam and Bjørner (2019) used graphical neural networks to learn heuristics for guiding SAT solvers.

Deep networks have also been used to attack a variety of combinatorial problems beyond SAT; see Bengio, Lodi, and Prouvost (2018) for a recent survey. Most notably, Evans et al. (2018) use a recurrent network for predicting logical entailment; this RNN architecture is not permutation-invariant, and it does not scale to the size of instances we considered in our own work. Prates et al. (2019) also studied predicting optimal tours in Euclidean traveling salesman problems.

Deep networks for exchangeable data A large body of recent work has studied deep network architectures for exchangeable arrays, such as sets (Zaheer et al. 2017), matrices and tensors (Hartford et al. 2018; Bloem-Reddy and Teh 2019), and graph structured data (Battaglia et al. 2018; Hamilton, Ying, and Leskovec 2017, and references therein). All of these approaches build deep network layers respecting the in- or equivariances[1] implied by the exchangeable array (e.g., sets are permutation-invariant, while matrices are equivariant under permutations of rows or columns), but they differ in design decisions about which representations to aggregate over, how multiple layers are composed, and which permutation-invariant functions are used to perform the aggregation.

In this paper, we explore the first two of these design dimensions. First, we use the exchangeable matrix architecture presented in Hartford et al. (2018) because it supports a natural CNF-style matrix encoding of SAT problems without requiring an intermediate graph representation. It was

[1]A function is *equivariant* if permutations of its input only result in a corresponding permutation of its output, and is *invariant* if permutations of its input leave the output unchanged. For instance, given a permutation matrix Π and an input matrix $X \in \mathbb{R}^{n \times m}$, a function $g : \mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$ is equivariant iff $g(\Pi X) = \Pi g(X)$ for all Π and X, and $g : \mathbb{R}^{n \times m} \to \mathbb{R}$ is invariant iff $g(\Pi X) = g(X)$ for all Π and X.

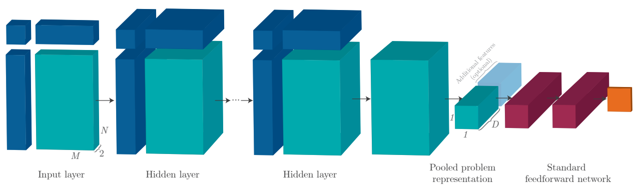
N
M
2
Input layer
Hidden layer
Hidden layer
Additional features (optional)
1
1
D
Pooled problem representation
Standard feedforward network

Figure 1: Illustration of the exchangeable architecture.

also shown to be maximally expressive among parameter sharing-based architectures for exchangeable matrices, and thus generalizes all approaches based on parameter sharing. Second, in terms of deciding how multiple layers are composed, exchangeable layers can either be treated like standard feedforward layers and stacked (with each layer having its own trainable parameters), or the same layer can be repeatedly applied for some number of steps as with a recurrent network. The latter approach is often referred to as neural message passing[2] (Gilmer et al. 2017), and was used by Selsam et al. (2019) for learning to solve SAT problems; in this paper, we evaluate their NeuroSAT model on the SAT prediction problem. More expressive attention-based aggregation functions (Vaswani et al. 2017) offer the hope of further performance improvements, but their quadratic complexity makes them infeasible for problems of the size that we study.

3 Model architecture

Data representation A SAT instance with n clauses and m variables in CNF can be represented as an $n \times m \times 2$ clause-variable tensor, where entry (i, j) is the one-hot vector $[1, 0]$ if the *true* literal for variable i appears in clause j, $[0, 1]$ if the *false* literal for variable i appears in clause j, and $[0, 0]$ otherwise. This tensor is a sparse exchangeable tensor: each clause will typically only reference a few variables (exactly three in the case of random 3-SAT), so most entries in the tensor will be the zero vector, and permuting the first two dimensions (rows or columns) will not change the SAT problem's satisfiability status.

We denote this input tensor $\mathbf{X}$ and let $\mathbb{I} = \{(i, j) :$ clause i references variable $j\}$ denote an index set of the non-zero entries of $\mathbf{X}$. The number of non-zero entries of $\mathbf{X}$ is given by $|\mathbb{I}| = \bar{m}n$, where $\bar{m}$ is the average number of variables that appear in each clause. For the random 3-SAT problems that we consider in the experiments $\bar{m} = 3$, so $|\mathbb{I}| = 3n$.

Exchangeable architecture In order to predict the satisfiability of a problem, we need to map from the raw representation of the problem, $\mathbf{X}$, to a scalar output, $y \in [0, 1]$, that indicates the probability of satisfiability. We achieve this in two steps that are trained jointly. First, we map each element of the raw input $\mathbf{X}$ to a D-dimensional embedding, using

[2]The term "message passing" refers to an aggregation step in a graph-based model, so it is sometimes also used for deep networks with separate feedforward-style layers. In this paper we limit it to the case where the same layer is applied repeatedly, analogous to the repeated application of local updates in message passing for graphical models.

a permutation-equivariant function $\phi : \mathbb{R}^{|\mathbb{I}|\times 2} \rightarrow \mathbb{R}^{|\mathbb{I}|\times D}$. We then pool the output of $\phi(\mathbf{X})$ to produce a single D-dimensional representation of the SAT problem. This is fed into a second function, $\rho : \mathbb{R}^D \rightarrow \mathbb{R}$, which is used to predict the probability that the problem is satisfiable.

We represent ϕ using a sequence of exchangeable matrix layers (Hartford et al. 2018). These layers apply to tensors $\in \mathbb{R}^{n\times m\times k}$; a tensor consists of k channels of exchangeable matrices, each of which is equivariant with respect to permutations of their n rows and m columns. For any given layer, the oth output channel's (i,j)th entry can be computed as follows (with bias terms omitted for clarity):

$$\mathbf{Y}_{o,i,j} = \sigma\left(\sum_{k=1}^{K}\left(\theta_1^{(k,o)}\mathbf{X}_{i,j,k} + \frac{\theta_2^{(k,o)}}{|\mathbb{C}(j)|}\sum_{i'\in\mathbb{C}(j)}\mathbf{X}_{i',j,k} + \frac{\theta_3^{(k,o)}}{|\mathbb{V}(i)|}\sum_{j'\in\mathbb{V}(i)}\mathbf{X}_{i,j',k} + \frac{\theta_4^{(k,o)}}{|\mathbb{I}|}\sum_{i',j'\in\mathbb{I}}\mathbf{X}_{i',j',k}\right)\right),$$

where $\sigma(\cdot)$ is a nonlinear activation function applied element-wise, $\mathbf{X}$ denotes an $n\times m\times K$ input tensor, $\theta_i^{(k,o)}$ are trainable weights, $\mathbb{C}(j)$ denotes the indices of all variables referenced in clause j, and $\mathbb{V}(i)$ denotes the indices of all clauses which contain variable i.

While the notation is complex, an exchangeable layer has a simple interpretation as a feed-forward network applied to each layer's literal representation (the terms associated with θ_1), as well as the respective mean-pooled representations of the associated variables (θ_2 terms), clauses (θ_3 terms), and the entire problem (θ_4 terms). The latter three terms provide the mechanism through which the network is able to share information about the assignment of literals between associated clauses and variables. By stacking multiple layers, longer chains of information propagation become possible.

The second function, ρ, acts on the pooled the output of $\phi(X)$. We use a standard multi-layer perceptron as follows:

$$\hat{y} = \rho\left(\frac{1}{|\mathbb{I}|}\sum_{i,j\in\mathbb{I}}\phi(\mathbf{X}_{i,j,:});\ \beta\right),$$

where $\phi(\mathbf{X}_{i,j,:})$ is a D-dimensional vector associated with clause i and variable j, and ρ is a multi-layer perceptron with weights β.

Given a dataset $\mathcal{D} = \{(\mathbf{X}_i, y_i)\}_{i\in[1,\ldots,n]}$ of SAT problems, $\mathbf{X}_i$, and targets $y_i \in \{0,1\}$ indicating whether or not a given problem is satisfiable, we train both networks jointly by optimizing β and θ to minimize the binary cross-entropy loss

$$\mathcal{L}_S = \sum_{(\mathbf{X}_i,y_i)\in\mathcal{D}} -y_i\log(\sigma(\hat{y}_i)) - (1-y_i)\log(1-\sigma(\hat{y}_i)).$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

Assignments In addition to a formula's overall satisfiability, this architecture can easily be extended to predict satisfying variable assignments. Given the representation $\phi(X)$, we can pool across clauses to produce variable-specific representations. Then, we can apply a third function, $\mu : \mathbb{R}^{m\times D} \rightarrow \mathbb{R}^m$, which yields the probability of each variable taking the value *true*. Again, we use a multi-layer perceptron to represent μ, $\hat{v}_j = \mu\Big(\frac{1}{|\mathbb{C}(j)|}\sum_{i'\in\mathbb{C}(j)}\phi(\mathbf{X}_{i',j,:})\Big)$.

As before, μ is trained in conjunction with the rest of the architecture by optimizing binary cross-entropy loss for each variable. The resulting combined loss is

$$\mathcal{L} = \sum_{(\mathbf{X}_i,y_i,\mathbf{v}_i)\in\mathcal{D}}\Big(\mathcal{L}_S(y_i,\hat{y}_i) + \frac{1}{\|\mathbf{v}(i)\|_0}\sum_j\mathcal{L}_A(v_{i,j},\hat{v}_{i,j})\Big),$$

where $\mathcal{L}_A(\cdot,\cdot)$ is the binary cross-entropy loss function, $\mathbf{v}_i$ is the true vector of assignments for problem i, and $\|\mathbf{v}(i)\|_0$ is the length of $\mathbf{v}_i$, i.e. the number of variables in the problem.

Message passing In the exchangeable model described above, each layer has its own set of parameters. Another approach is to have all layers share the same set of parameters in a manner similar to an unrolled recurrent neural network (RNN), where each "layer" corresponds to a single recurrent step. This approach is taken by the NeuroSAT architecture.

NeuroSAT represents SAT problems as a bipartite graph with one set of vertices containing true and false literals, and the other clauses; edges denote a literal appearing in a clause. The model has two separate RNNs – one for clauses and one for literals. Each iteration involves first updating the clauses by aggregating over the neighbouring literals and applying the clause RNN, and then updating the literals by aggregating over the updated neighbouring clauses and applying the literal RNN. Each iteration can be expressed formally as

$$(C^{(t+1)}, C_h^{(t+1)}) \leftarrow g_c([C_h^{(t)}, \sum_{i\in\mathcal{N}(C)} f_c(L_i^{(t)})])$$
$$(L^{(t+1)}, L_h^{(t+1)}) \leftarrow g_l([L_h^{(t)}, \sum_{i\in\mathcal{N}(L)} f_l(C_i^{(t+1)})])$$

where $g_c(\cdot)$ and $g_l(\cdot)$ denote the clause and literal RNNs, C^t and C_h^t are the clause representation and recurrent hidden state (similarly for the literals), $f_c(\cdot)$ and $f_l(\cdot)$ are multilayer perceptrons (MLP), and $\mathcal{N}(\cdot)$ returns the indices of the neighbours of a clause or literal.

By applying the clause and literal RNNs sequentially and using MLPs in the aggregation operation, the NeuroSAT layer introduces multiple intermediate non-linearities that make exact comparisons with the exchangeable layer impossible. Loosely, the recurrent hidden state plays the same role as the θ_1 terms, and aggregation across clauses and literals corresponds to a nonlinear version of the θ_2 and θ_3 terms.

4 Experimental setup

Data generation To evaluate our approach, we generated uniform-random 3-SAT instances at the solubility phase transition. Following Crawford and Auton (1996), we used a clause (n) to variable (m) ratio of $n = 4.258m + 58.26m^{-2/3}$ to approximate the location of the phase transition. We created 11 datasets, each with a fixed number of variables ranging from 100 to 600 variables at intervals of 50; each dataset contained $10,000$ instances, including exactly

5000 satisfiable instances and 5000 unsatisfiable instances.[3] 100-variable instances can trivially be solved by modern SAT solvers in milliseconds; 300-variable instances require several seconds; and 600-variable instances take several hours to solve.

We believe that this dataset constitutes a useful benchmark for deep models on exchangeable data. Our data generation process creates hard-to-predict problems, but with a noise-free target that offers the asymptotic potential for 100% accuracy. Indeed, training sets of arbitrary size can be generated (albeit at significant computational cost), and testing whether models generalize to unseen instance sets is easily possible because of the natural relationship between sizes.[4]

For our experiments, we randomly split both satisfiable and unsatisfiable problems of each instance size into training, validation, and test sets according to an 80:10:10 ratio. We report the test performance of the models which performed best on the validation set.

Network training procedure We evaluated four variants of deep network architectures: the standard exchangeable architecture and message passing NeuroSAT architecture predicting only satisfiability, and an extension to both architectures where we jointly predicted satisfying variable assignments. These variants are described in Section 3.

For the exchangeable architecture, we adapted the public implementation of exchangeable matrix layers from Hartford et al. (2018). We instantiated the permutation-equivariant portion of the exchangeable architecture as eight exchangeable matrix layers with 128 output channels, with leaky RELU as the activation function. We mapped the final layer to an output width of $D = 64$, which was pooled to a vector before being mapped to the output. We also experimented with inserting a hidden layer between the pooled vector and the output, but observed no significant impact on performance.

For training the exchangeable architecture, we used the Adam optimizer with a learning rate of 0.0001, and mini-batches of 32 examples. Since instances can vary in size, with some being very large, we accumulated gradients for the mini-batches sequentially, back-propagating losses individually for each instance. This slowed training, but was necessary because entire mini-batches could not always be accommodated in memory.

For the the message passing network, we used Selsam et al. (2019)'s implementation in the Tensorflow framework. We used the hyperparameters of Selsam et al. (2019): a dimension of 128 for clause and variable embeddings, 3 hidden layers and a linear output layer for each of the MLPs, a scaling factor of 10^{-10} for the ℓ_2 norm to regularize the parameters, 32 iterations of message passing for each problem, a learning rate of 0.00002 for Adam, and a clipping ratio of 0.65 for clipping gradients by global norm. To jointly predict assignments and satisfiability, we added an additional MLP to the aggregation operation that maps literal representations to assignments for each variable, and optimize the same combined loss function described in Section 3. Like Selsam et al., we created batches of problems containing up to $12,000$ clause and literal nodes that we fed through the network at once. We selected the best-performing model with a validation set, and report test set accuracy obtained by running the model with 32 message passing iterations per problem.

Baselines To compare our results with the state of the art, we evaluated the performance of decision forests trained on the hand-engineered features used by Xu, Hoos, and Leyton-Brown (2012). We also implemented a feed-forward neural network that took as input the same hand-engineered features, to ensure that any performance differences were not driven by Xu, Hoos, and Leyton-Brown's choice of model family.

We also compared our results to two simple baselines that could be trained end-to-end. First, we considered convolutional neural networks (CNNs) to evaluate the impact of capturing permutation invariance. The input to the CNN is a dense representation of the sparse tensor described in 3. Additionally, to determine whether a simple version of permutation invariance was sufficient to achieve good performance, we also investigated a permutation-invariant nearest-neighbour approach. We used the graph edit distance between variable-clause graphs to determine nearest neighbours, and predicted the satisfiability status of a new point as the satisfiability status of its nearest neighbour.

5 Experimental results

Prediction accuracy We evaluated prediction accuracy for the four variants of deep neural network architectures and the four baselines. For the nearest-neighbour baseline, we only considered the 100- and 200-variable datasets because of the high computational cost of computing graph edit distances; for all other approaches, we considered all 11 fixed-size datasets, as described in Section 4.

Both nearest neighbour and CNNs performed poorly. Nearest neighbour never achieved prediction accuracies above 53%, even when we used the expensive Hausdorff graph edit distance. The performance of CNNs consistently approached that of random guessing, achieving no more than 50.5% on any of the 11 datasets after 48 hours of training. We concluded that permutation-invariant architectures made a significant impact on predictive performance.

Table 1 presents the performance results for the permutation-invariant methods. Like Xu, Hoos, and Leyton-Brown (2012), we observed that prediction accuracy *increased* with instance size for the exchangeable variants of the permutation-invariant architecture. With the exchangeable model variant predicting only satisfiability, we achieved prediction accuracies between 71% and 82% — 1–7% higher than random forests and 1–8% higher than the fully-connected neural network using hand-engineered features.

Using the exchangeable model variant where satisfiability and assignments were jointly predicted, we achieved prediction accuracies between 72% and 84%, with an improvement

[3]To confirm that we sampled instances at the phase transition, we examined the fraction of satisfiable instances that we generated; we found no evidence that it diverged significantly from 50%.

[4]For the raw data and the full details of our data generation process, please see http://www.cs.ubc.ca/labs/beta/Projects/End2EndSAT/.

	Hand-Engineered		Exchangeable		MP	
#Vars	RF	NN	SAT	+Assign	SAT	+Assign
100	0.702	0.704	0.712	0.726	0.675	**0.751**
150	0.712	0.714	0.731	0.745	**0.778**	0.771
200	0.734	0.718	0.760	0.772	0.767	**0.781**
250	0.703	0.723	0.776	**0.800**	0.758	0.788
300	0.744	0.725	0.789	**0.800**	0.758	0.788
350	0.734	0.730	0.787	0.809	**0.834**	0.812
400	0.711	0.710	0.765	**0.790**	0.777	0.781
450	0.700	0.710	0.788	0.789	0.774	**0.803**
500	0.773	0.778	0.800	**0.809**	0.791	0.795
550	0.756	0.761	0.804	0.810	0.789	**0.813**
600	0.813	0.810	0.811	**0.837**	0.816	0.818

Table 1: Comparison of prediction accuracy for satisfiability in the epoch with the lowest validation error. RF denotes random forests; NN, the standard feed-forward network; Exchangeable, the standard exchangeable network; and MP, the message passing model of Selsam et al. (2019). SAT denotes the permutation-invariant model variants trained to predict satisfiability; +Assigns, the permutation-invariant model variants trained to predict satisfiability and satisfying assignments. Boldface indicates the best-performance.

in the model by an additional 1–2% across all datasets. We achieved similar levels of performance with message passing architectures, achieving prediction accuracies usually between 75% and 83%, however accuracies were roughly uncorrelated with instance size. Using the message passing variant where satisfiability and assignments were jointly predicted, we achieved an additional 1.6% accuracy on average. Averaged across all datasets, the exchangeable and message passing architectures with assignments respectively achieved 78.9% and 79.1% prediction accuracy. Neither architecture was better than the other on more than 6/11 of the instance sets. In terms of prediction accuracy when testing on the same instance sizes used for training, we believe there is no reason to hold a preference between the exchangeable and message passing architectures.

Running time Permutation-invariant neural networks are far more expensive to train than the random forest baseline, requiring at least 40 hours of training time and 500 MB of GPU memory (for exchangeable networks; message passing is more expensive, requiring 0.6 to 3.3 GB per instance). However, evaluating this class of models requires only time linear in the size of the input, whereas the hand-engineered features upon which the random forest models depend have roughly cubic time complexity. This asymptotic difference is not overwhelmed by constants: at the input sizes we investigated, exchangeable models are faster to evaluate by several orders of magnitude, as shown in Figure 2.We note that this difference might make such models particularly attractive for constructing algorithm portfolios, where time saved on feature computation can be reallocated to solving problems.

Generalizing across sizes To verify that permutation-invariant models are able to capture general structural properties of the given SAT instances rather than simply memorizing instances at particular sizes, we evaluated the prediction accuracy of networks trained on 100-variable instances using all of the other datasets. Our results, along with analogous results for random forests, are shown in Table 2.

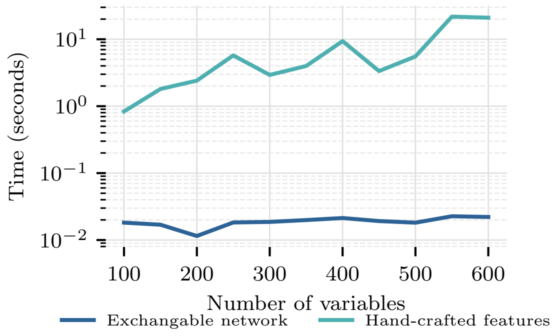

Figure 2: Average running times per instance as size varies for the exchangeable architecture and hand-engineered features. Note the log scale on the y-axis.

#Vars	RF-100	Exch-100	MP-100
150	0.695	**0.758**	0.734
200	0.695	**0.759**	0.704
250	0.654	**0.776**	0.722
300	0.711	**0.780**	0.729
350	0.705	**0.791**	0.725
400	0.681	**0.756**	0.711
450	0.692	**0.778**	0.699
500	0.716	**0.777**	0.686
550	0.722	**0.768**	0.669
600	0.739	**0.809**	0.683

Table 2: Comparison of satisfiability prediction accuracy achieved by testing a model trained on 100-variable instances on other datasets. RF-100 denotes the random forests trained on 100 variables and tested on other sizes; Exchangeable-100, the exchangeable network trained to predict satisfiability and assignments on 100 variables, and tested on other sizes; and MP-100, the message passing model of Selsam et al. (2019) trained on 100 variables and tested on other sizes. Boldface indicates the best-performing approach for each dataset.

With the exchangeable architecture trained on 100-variable instances, we achieved nearly undiminished performance and definitively outperformed random forests on all instance sizes. Notably, the model trained on trivial 100-variable instances achieved 81% accuracy on hard 600-variable instances. We again observed the increase in prediction accuracy with instance size reported by Xu, Hoos, and Leyton-Brown.

Prediction accuracy for message passing was usually between 68–74%. We note that this range is similar to the prediction accuracy achieved when testing message passing on 100-variable instances. By comparison, considering models trained on 100-variable instances, the performance of the message passing model did not generalize as well as the exchangeable architecture, which achieved better performance on all instance sizes. In fact, we observed that the accuracy

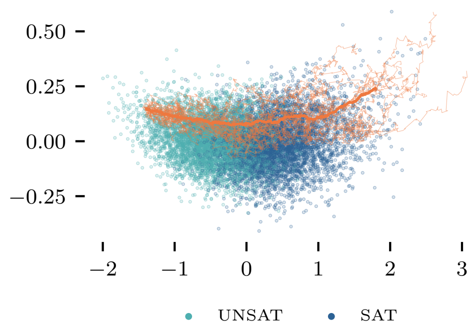

Figure 3: In the space of problem embeddings in the exchangeable architecture, orange lines show paths through towards the SAT portion of the space as clauses are removed from an UNSAT instance, with the thicker line showing the average path. The original 64-dimensional space is projected down to the first principal component of a model trained on the 300-variable dataset.

of message passing decreased with instance size, and that the exchangeable architecture achieved 12% better accuracy for the biggest instance sizes (550 and 600 variables).

Overall, the exchangeable architecture and message passing achieved comparable performance when trained on the same distribution. However, based on their superior generalization performance and lower memory requirements, we recommend exchangeable models over message passing for predicting satisfiability.

SAT invariances Visualizing the embedding space provides a way to verify that the exchangeable architecture performed as desired. We used the following observation to examine its behaviour as instances were modified: the more clauses are removed from any unsatisfiable instance, the greater the chance that it will become satisfiable, since removing a clause can never reduce the solution space. Taking a random instance predicted to be unsatisfiable by the network, we iteratively removed randomly-selected clauses. At every step, we recorded the network's pooled representation of the instance. A projection of the paths from 20 independently sampled trajectories is shown in Figure 3. As clauses were removed, the representation of the instance shifted from the portion of the space associated with unsatisfiable instances to the portion associated with satisfiable instances. The predicted probability of satisfiability also increased, as expected.

Variance in latent space We explored the variance in the exchangeable architecture's latent space as problem size changed. Figure 4 shows the distributions from the final exchangeable layer for the exchangeable architecture projected down to the first principal component for both satisfiable and unsatisfiable instances in four different-sized datasets. We observe a clear trend of variance decreasing with problem size. Certain functions on random graphs converge as $n \to \infty$ (e.g., the size of the maximum clique in Erdos-Renyi graphs). We conjecture that the neural networks are learning a function that concentrates with increasing problem size.

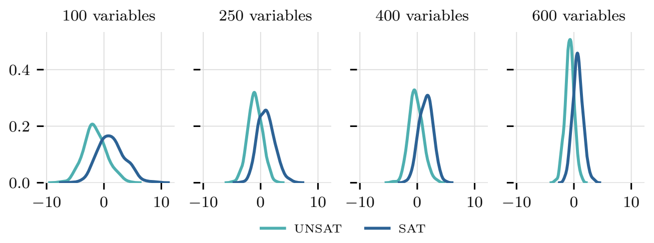

Figure 4: Comparison of variance in problem embeddings of the exchangeable architecture as instance size increases. The original 64-dimensional space is projected down to the first principal component based on models trained with 300 variables. For each size, a kernel density function is fit for UNSAT and SAT problems.

6 Conclusions and future work

This paper is the first to study end-to-end learning of the satisfiability status of empirically challenging SAT problems based on their CNF representations, with direct applicability to other combinatorial problems with permutation-invariant structures. We showed that both deep exchangeable and neural message passing models achieved state-of-the-art prediction performance on random 3-SAT problems at the phase transition, consistently outperforming models based on sophisticated hand-engineered features that have been central to machine learning in SAT for over a decade. These models also have a clear computational advantage over hand-engineered feature-based models: for 600-variable problems, a forward pass of the exchangeable architecture was more than two orders of magnitude faster than computing hand-engineered features. We also showed out-of-sample generalization to much larger instance sizes at nearly undiminished levels of accuracy. Indeed, the exchangeable network architecture trained on 100-variable instances (milliseconds to solve) achieved performance on 600-variable instances (hours to solve) which was on par with that of hand-engineered feature-based models trained on 600-variable instances! We observed no clear difference between exchangeable networks and message passing in terms of generalization to new SAT instances of the same size used for training, but observed that exchangeable networks were considerably better at generalizing to new SAT instances of larger size.

We are currently investigating the extent to which the predictability of SAT from raw instances depends on the choice of SAT distribution. Early results indicate significant variance in prediction performance across distributions. Further study is needed to understand which properties of empirically hard SAT distributions relate to predictability.

Acknowledgements

This work leveraged compute resources provided by Compute Canada, CFI, BCKDF, and NVIDIA; it was funded by an NSERC Discovery Grant, a DND/NSERC Discovery Grant Supplement, a CIFAR Canada AI Research Chair (Alberta Machine Intelligence Institute), and DARPA award FA8750-19-2-0222, CFDA #12.910 (Air Force Research Laboratory).

References

Allamanis, M.; Chanthirasegaran, P.; Kohli, P.; and Sutton, C. A. 2016. Learning continuous semantic representations of symbolic expressions. *arXiv preprint*.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2018. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint*.

Bloem-Reddy, B., and Teh, Y. W. 2019. Probabilistic symmetry and invariant neural networks. *arXiv preprint*.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, IJCAI '91, 331–337. Morgan Kaufmann.

Clarke, E.; Biere, A.; Raimi, R.; and Zhu, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 19(1):7–34.

Crawford, J. M., and Auton, L. D. 1996. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence* 81:31–57.

Evans, R.; Saxton, D.; Amos, D.; Kohli, P.; and Grefenstette, E. 2018. Can neural networks understand logical entailment? *arXiv preprint*.

Finkler, U., and Mehlhorn, K. 1997. Runtime prediction of real programs on real machines. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, 380–389. Society for Industrial and Applied Mathematics.

Fréchette, A.; Newman, N.; and Leyton-Brown, K. 2016. Solving the station repacking problem. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, AAAI '16, 702–709. AAAI Press.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *ICML '17*, 1263–1272. JMLR.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint*.

Hartford, J. S.; Graham, D. R.; Leyton-Brown, K.; and Ravanbakhsh, S. 2018. Deep models of interactions across sets. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 1914–1923. PMLR.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: methods & evaluation. *Artificial Intelligence* 206:79–111.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems 31*, NIPS '18, 539–548.

Lindauer, M.; Hoos, H. H.; Hutter, F.; and Schaub, T. 2015. Autofolio: Algorithm configuration for algorithm selection. In *Workshops at the 29th AAAI Conference on Artificial Intelligence*, AAAI '15. AAAI Press.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. 2016. Deep learning for algorithm portfolios. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, AAAI '16, 1280–1286. AAAI Press.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of SAT problems. In *Proceedings of the 10th AAAI Conference on Artificial Intelligence*, AAAI '92, 459–465.

Mu, Z., and Hoos, H. H. 2015. On the empirical time complexity of random 3-SAT at the phase transition. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, IJCAI '15, 367–373. Morgan Kaufmann.

Nudelman, E.; Leyton-Brown, K.; Hoos, H. H.; Devkar, A.; and Shoham, Y. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Tenth International Conference on Principles and Practice of Constraint Programming*, CP '04, 438–452. Springer Berlin Heidelberg.

Prates, M.; Avelar, P. H.; Lemos, H.; Lamb, L. C.; and Vardi, M. Y. 2019. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4731–4738.

Sandholm, T. W. 1996. A second order parameter for 3SAT. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, AAAI '96, 259–265. AAAI Press.

Sekiyama, T., and Suenaga, K. 2018. Automated proof synthesis for propositional logic with deep neural networks. *arXiv preprint*.

Selsam, D., and Bjørner, N. 2019. Guiding high-performance SAT solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing - SAT '19*, Lecture Notes in Computer Science. Springer.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill., D. L. 2019. Learning a SAT solver from single-bit supervision. In *Proceedings of the 7th International Conference on Learning Representations*, ICLR '19.

Smith-Miles, K., and Lopes, L. 2012. Measuring instance difficulty for combinatorial optimization problems. *Computers and Operations Research* 39(5):875–889.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2007. Hierarchical hardness models for SAT. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP '07, 696–711. Springer-Verlag.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2012. Predicting satisfiability at the phase transition. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI '12, 584–590. AAAI Press.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R. R.; and Smola, A. J. 2017. Deep sets. In *Advances in Neural Information Processing Systems*, NIPS '17, 3391–3401.