A Data generation

We use a methodology similar to Xu, Hoos, and Leyton-Brown (2012) to evaluate the performance of our architecture. First, we used the cnfgen Python library (Lauria et al. 2017) to generate uniform random 3-SAT instances at the phase transition. Following Crawford and Auton (1996), we used $m = 4.258n + 58.26n^{-2/3}$ to estimate the location of the phase transition. We generated $5,000$ SAT instances and $5,000$ UNSAT instances between 100 and 600 variables in steps of 50 variables. In total, we obtained $11,000$ instances in 11 sets.

To obtain the prediction targets (satisfiability and satisfying assignments) for each instance, we ran used the lookahead solver march_hi (Heule and van Maaren 2009). Although our generation script was limited to a runtime of 24 hours for all instances in a particular set, we repeatedly executed the script until all instances were obtained.

For our experiments, we used 61 features from Nudelman et al. (2004). These include 7 problem size features, 29 graph-based features, 13 features related to balance of positive and negative literals, 6 features related to distance to horn formula, and 6 features related to LP relaxation.

All instances were generated on a shared compute cluster, on one of 24 nodes equipped with 1 cores, two 2.1 Ghz Intel E5-2683 v4 Broadwell processors per core, and 16 GB of RAM per core. A memory budget of 15 GB was set for each execution of the generation script.

B Baseline model details

B.1 Random forests

We followed Xu, Hoos, and Leyton-Brown (2012) in choosing hyperparameters for our decision forests. Each forest was composed of 99 trees. Given a training set of n instances, we drew a bootstrap sample of n instances with replacement for each tree, and sampled a random subset of $\lfloor \log_2(91) \rfloor + 1 = 7$ features at each internal node for splitting. We used majority voting, weighted by probability estimates, across all trees to make predictions. Xu, Hoos, and Leyton-Brown (2012) did not specify either the splitting criteria or min-split parameter (i.e, number of samples required to split a node) that they used. We used the Gini impurity for evaluating the quality of splits, and required a minimum of five samples to split a node.

B.2 Multilayer perceptron

Our feed-forward network architecture maps the input to a width of 64, followed by 8 hidden layers with 128 units each, and then maps to a 1×1 output. Leaky ReLU activation and dropout with a ratio of 0.5 are applied to each layer. The same training procedure and computing nodes used to train the exchangeable network were used to train the feed-forward network.

B.3 Convolutional network

Our convolutional network applies three convolutional layers to the input, each with 18 output channels and kernel size 3. Each convolutional layer is followed by a 2×2 max pooling layer and RELU activations. We then attach two fully connected layers of 64 and then 10 nodes. Our convolutional network can only be used to make predictions for fixed-size instances due to the fixed size of the input layer.

B.4 Nearest neighbour

Our nearest neighbour method uses a graph-edit distance metric, which is permutation-invariant. To do so, we represent a CNF SAT instance as a clause-variable graph. In the graph, each variable and its negation are connected vertices, and a variable vertex shares an edge with a clause vertex if the variable participates in that clause. We use both the greedy and Hausdorff graph-edit distance metrics from the Python library GMatch4py. The Hausdorff edit distance matches nodes based on the similarity of local substructure, and computes the number of edits required for the graphs to be equivalent. The greedy graph edit distance is a greedy approximation of the Hausdorff edit distancem, which runs in quadratic time.

C Training details

All training took place on a shared compute cluster, with 114 nodes equipped with 24 cores, two 2.2 Ghz Intel E5-2683 v4 Broadwell processors per core, 5.3 GB of RAM per core, and four NVIDIA P100 Pascal GPUs with 12 GB of HBM2 memory per core. Each execution of the training script was allocated one GPU and had a memory budget of 2 GB for the exchangeable models and 20 GB for the NeuroSAT models. The same computing nodes used for instance generation were used to evaluate the random forests.

C.1 Training plots

Similar trends in validation error (e.g. Figure 5) were observed between different training experiments on each instance size. Training losses for models that predicted satisfying assignments were higher due to cross-entropy losses for assignments. The inclusion of assignment prediction and size information led to more rapid convergence, but also earlier overfitting in the training process. As expected, overfitting also occurred earlier for smaller instances, for which less structural information can be learnt by the network.

D Additional experiments

D.1 Size information

When pooling across representations, mean pooling ensures size invariance, meaning that information is lost if absolute counts of features are important. We tested whether this was true by appending the size of any pooled representation as an additional feature, which allows subsequent layers to learn to "undo" the pooling if necessary. If $\phi(x)$ is a representation over which we pool, the pooling operation becomes $y = \frac{1}{N} \sum_{i=1}^{N} [\phi(x_i); N]$, where $[\mathbf{x}; N]$ denotes the concatenation of the vector x and N. We used a base-2 encoding of N to ensure that, if the input varies significantly in size, the weights associated with N would only cover a small range.

We found that, for an exchangeable network with eight exchangeable layers, appending size information caused networks to overfit more quickly, and consistently resulted in poorer performance at the training set sizes we explored. However, they still generally achieved improvements of $1 - 8\%$ over random forests, and $1 - 6\%$ over feed-forward networks. These results are shown in Table 3. However, when experimenting with different architectures, we found that size information improved prediction accuracy on models with 3–5 exchangeable layers.

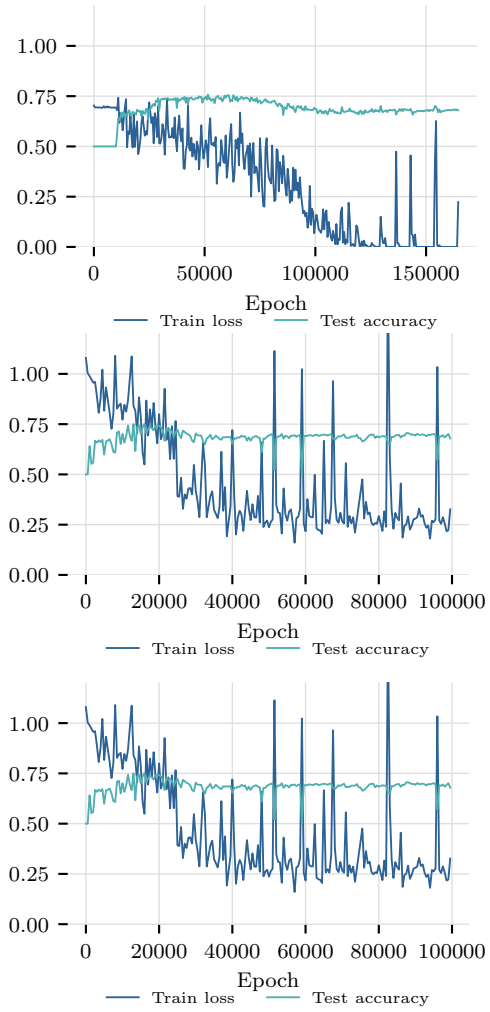

Figure 5: Training loss and SAT prediction accuracy for exchangeable networks trained to predict satisfiability *(above)*, both satisfiability and satisfying assignments *(middle)*, and both with size information appended to the input *(below)*.

D.2 Predicting satisfying assignments

For the variants of our exchangeable network trained to jointly predict satisfiability and satisfying assignments, we also evaluated their prediction accuracy on satisfying assignments. As shown in Table 4, both networks acheived prediction accuracies between 70% and 73%. Note that we were only evaluating assignment accuracy based on single satisfying assignment found by a SAT solver. Each satisfiable instance may have several satisfying assignments. Earlier overfitting by the network with size information appended to the inputs again decreased prediction accuracy.

D.3 Cross-training with multiple sizes

We further tested the ability of our network to capture general structural information by determining whether networks trained on instances of multiple sizes achieved better prediction accuracy. To do so, we considered the 100-variable, 200-variable, 300-variable, 400-variable, 500-variable, and 600-variable instance sets. Networks were trained on five of the sets, and evaluated on the sixth set, and the resulting prediction accuracy was compared against networks trained on instances of the same sizes as the test set.

Variable	+Sizes
100	0.711
150	0.741
200	0.759
250	0.776
300	0.780
350	0.791
400	0.781
450	0.778
500	0.768
550	0.806
600	0.812

Table 3: Comparison of prediction accuracy for satisfiability achieved during the epoch with the lowest validation error, using networks trained to predict satisfiability and satisfying assignments with size information appended to the input. Refer also to Table 1.

Table 5 shows the satisfiability prediction accuracies for all three variants. Their performance was comparable to networks trained on single problem sizes; on smaller instances (< 400 variables), they had better prediction accuracy. Unlike networks trained on single problem sizes, size information generally improved prediction accuracy, which suggests that size information is useful for capturing structural information from instances of multiple sizes.

Variable	+Assigns	+Sizes
100	**0.725**	0.721
150	**0.722**	0.715
200	**0.722**	0.715
250	**0.717**	0.712
300	**0.719**	0.709
350	**0.713**	0.707
400	**0.715**	0.708
450	**0.714**	0.704
500	**0.712**	0.709
550	**0.711**	0.706
600	**0.712**	**0.712**

Table 4: Comparison of prediction accuracy for satisfying assignments achieved during the epoch with the lowest validation error. +Assigns denotes the network trained to predict satisfiability and satisfying assignments, and +Sizes denotes the network trained to predict satisfiability and satisfying assignments with size information appended to the input.

Test Variable	SAT (vs orig.)	+Assigns (vs orig.)	+Sizes (vs orig.)
100	0.760 (0.712)	**0.775** (0.726)	0.749 (0.711)
200	0.784 (0.760)	**0.786** (0.772)	**0.787** (0.759)
300	0.773 (0.789)	0.775 (0.800)	**0.791** (0.780)
400	0.739 (0.765)	0.768 (0.790)	**0.770** (0.781)
500	0.774 (0.800)	0.769 (0.809)	**0.770** (0.768)
600	0.762 (0.814)	0.765 (0.837)	**0.766** (0.812)

Table 5: Comparison of prediction accuracy for satisfiability using networks evaluated using one of the 100-variable, 200-variable, 300-variable, 400-variable, 500-variable, or 600-variable instance sets, and trained on the five other sets. Prediction accuracies for networks trained on the testing variable sets are replicated from Table 1 in parentheses. SAT denotes the permutation-equivariant network trained to predict satisfiability; +Assigns, the network trained to predict satisfiability and satisfying assignments; and +Sizes, the network trained to predict satisfiability and satisfying assignments with size information appended to the input. Boldface indicates the best-performing approach.