EasyChair Preprint

№ 2185

Directed Graph Networks for Logical Entailment

Michael Rawson and Giles Reger

February 22, 2020

Directed Graph Networks for Logical Entailment

Michael Rawson and Giles Reger

University of Manchester, UK
michael@rawsons.uk, giles.reger@manchester.ac.uk

Abstract

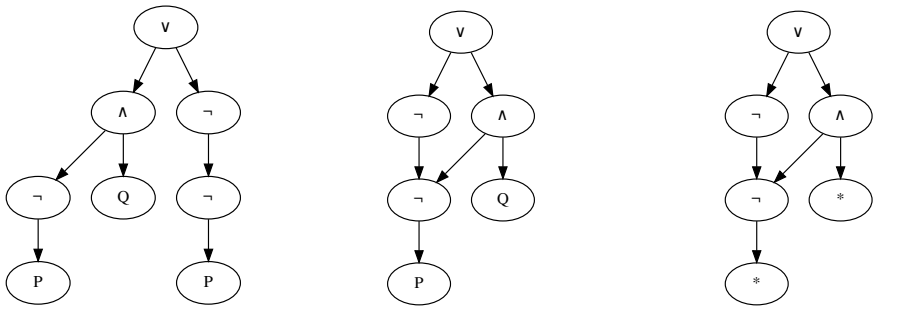
We introduce a novel neural model for detecting propositional entailment, a benchmark task for learning on logical structures, based upon learned graph convolutions on directed syntax graphs. The model removes some inflexible inductive bias found in previous work on this domain, while producing competitive results on the benchmark datasets. Model performance on larger problems surpasses all previous work. We also introduce a similar first-order learning problem and show good performance of the same model on this task. Such models have many applications for learned guidance of first-order theorem provers.

1 Introduction

Neural networks are ubiquitous in tasks in which features must be extracted from unstructured data — tasks such as computer vision, or natural language processing. However, learning from data that are already highly-structured is under-studied, but sorely needed in fields such as program synthesis or automated reasoning. We approach this area from guidance of automatic theorem provers for first-order logic: an undecidable setting that nevertheless might benefit from heuristic guidance, as strategies for a subset of "useful problems" can be learned this way. It should be noted that we do not aim to *solve* known computationally-hard or undecidable problems with a neural approach, merely approximate these functions for practical purposes. In this work we explore the use of neural models for heuristic tasks on logical data using detection of logical entailment in both propositional and first-order settings as a benchmark task.

Propositional Task and Dataset Evans et al. [3] introduce a dataset for studying the ability of neural networks to perform tasks which are "primarily or purely *about* sequence structure". The dataset consists of tuples of the form (A, B, y) where A and B are propositional formulae and y is the binary output variable. The task is to predict logical entailment: whether or not $A \models B$ holds in classical propositional logic. A and B use only propositional variables and the connectives $\{\neg, \wedge, \vee, \Rightarrow\}$ with the usual semantics. The dataset provides training, validation and test sets, with the test set split into several categories: "easy", "hard", "big", "massive" and "exam". The "massive" set is of particular interest to us as it contains larger entailment problems, more similar in size to those found in real-world problems where redundant axioms and voluminous structures are commonplace.

Previous Approaches *PossibleWorldNet* is introduced alongside this dataset as a possible solution to the task: an unusual neural network architecture making use of algorithmic assistance in generating repeated random "worlds" to test the truth of the entailment in that world, in a similar way to model-based heuristic SAT solving. This approach performs exceptionally well, but does suffer from inflexibility: it is unclear how this model would perform on harder tasks without a finite number of possible worlds, or tasks where model-based heuristics don't perform as well. Tending instead toward a purely-neural approach, Chvalovský introduces TopDownNet [1], a recursively-evaluated neural network with impressive results on this dataset.



(a) syntax tree (b) DAG - named (c) DAG - nameless

Figure 1: Producing an exemplar DAG representation of $(\neg P \wedge Q) \vee \neg\neg P$. A propositionally-equivalent formula can be retrieved from the final DAG.

These two neural models are the most accurate learned estimators for logical entailment to date. Graphical representations have been used with some success for other logical tasks: Olšák et al. introduce a model based on message-passing networks working on hypergraphs [14], while Paliwal et al [15] use undirected graph convolutions for a higher-order task. An interesting effort not directly related to this task is that of NeuroSAT [19], a neural network that learns to solve SAT problems presented in conjunctive normal form.

Graph Neural Networks Graphs have historically proven difficult for learning algorithms of various varieties, mostly due to a very rich structure. However, recent advances [12] have produced a family of methods generally known as *Graph Neural Networks*, with *graph convolutions* as a central technique. These are simple, efficient networks practically useful for many tasks operating on graph data.

Contributions Our main contribution is a neural model that scores well on this propositional dataset, surpassing PossibleWorldNet in several test categories, yet not suffering from the inductive bias applied by this approach. To achieve this performance we introduce a directed graph convolution which appears crucial for this domain. In order to demonstrate the generality of our method, we also design a first-order classification problem and show good learning performance of the same model.

2 Input Encoding

Directed acyclic graphs (DAGs) are a natural, lossless representation for most types of logical formulae the authors are aware of; including modal, first-order and higher-order logics, as well as other structural data such as type systems or parsed natural language. A formula-graph is formed by taking a syntax tree (such as that produced by a parsing routine) and merging common sub-trees, followed by mapping distinct named nodes to nameless nodes that nonetheless remain distinct: an example is shown in Figure 1. This translation takes only polynomial time, and clearly does not trivialise the problem. Such graphs have previously

been used for problems such as premise selection [23] or search guidance of automatic theorem provers [18]. It should be noted that the acyclic property of these graphs does not seem to be particularly important — it just so happens that convenient representations happen to be acyclic. This representation has several desirable properties:

Compact size. Sufficiently de-duplicated syntax DAGs have little to no redundancy, and in pathological cases syntax trees are made exponentially smaller.

Shared processing of redundant terms. Common sub-trees are mapped to the same DAG node, so models that work on the DAG can identify common sub-terms trivially.

Bounded number of node labels. By use of nameless nodes, a finite number of different node labels are found in any DAG. This allows for simple node representations and does not require a separate textual embedding network.

Natural representation of bound variables. Representing bound variables such as those found in first-order logic can be difficult [17] — this representation side-steps most, if not all, of these issues and naturally encodes α-equivalence.

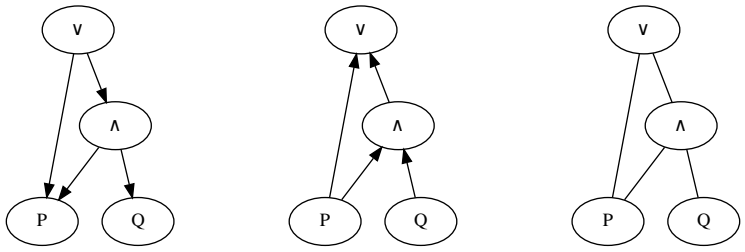
One drawback of such DAGs as a representation for logical formulae is that they lack ordering among node children: with a naïve encoding, the representation for $A \Rightarrow B$ is the same as $B \Rightarrow A$, but the two are clearly not equivalent in general. Similar problems might also arise with first-order terms: $f(c, x)$ is indistinguishable from $f(x, c)$. However, this problem can be mitigated by use of auxiliary nodes and/or edges such that an ordering can be retrieved, as shown in Section 6. For this particular dataset, the classical equivalence $A \Rightarrow B \equiv \neg A \vee B$ is used to rewrite formulae without implication, thus avoiding ordering issues as $\wedge$ and $\vee$ are commutative operators. We also recast the entailment problem $A \models B$ as a satisfiability problem: is $A \wedge \neg B$ unsatisfiable? These methods reduce the total number of node labels used (4 in total — one for propositional variables, and one for each of $\{\neg, \wedge, \vee\}$), and allow the network to re-use learned embeddings and filters for the existing operators.

3 Model

We introduce and motivate a novel neural architecture — EntailmNet — for learning based on DAG representations of logical formulae. Certain unusual neural structures were found to be useful, and are described first. These blocks are then combined into the model architecture used for the entailment task.

3.1 Bi-directional Graph Convolutions

We assume the input DAG is a graph $(\mathbf{X}, \mathbf{A})$ where $\mathbf{X}$ is the node feature matrix and A is the directed graph adjacency matrix. Various graph convolution operators [24] (denoted $\text{conv}(\mathbf{X}, \mathbf{A})$ here as an arbitrary operator) have enjoyed recent success. These generalise the trainable convolution operators found in image-processing networks to work on graphs, by allowing each layer of the network to produce an output node per input node based on the input node's existing data and that of neighbouring nodes connected with *incoming* edges. This can be seen as passing messages around the graph: with k convolution layers, a conceptual "message" may propagate k hops across the graph. Here, we use the standard convolutional layer found in Graph Convolutional Networks [12]. This operator suffers from a shortcoming (illustrated in Figure 2) on DAGs such as those used here: information will only pass in one



(a) top-down only (b) bottom-up only (c) undirected

Figure 2: Information flow in a formula DAG representing $P \wedge Q \vee P$.

direction through the DAG, as messages propagate only along incoming edges. Unidirectional messages are not necessarily a problem: bottom-up schemes such as TreeRNNs [22] exist, and Chvalovský uses [1] a top-down approach. Cyclic edges are another possible solution. However, to play to the strengths of the graphical approach the ideal would have messages passed in both directions, with messages from incoming and outgoing edges dealt with separately. It is possible to simply make the input graph undirected, but this approach discards much of the crucial encoded structure and was not found to perform much better than chance on this task. Instead, a bi-directional convolution is one possible solution:

$$\text{biconv}(\mathbf{X}, \mathbf{A}) = \text{conv}(\mathbf{X}, \mathbf{A}) \| \text{conv}(\mathbf{X}, \mathbf{A}^\mathsf{T})$$

where the $\|$ operator denotes feature concatenation. By convolving in both edge directions and concatenating the node-level features produced, information may flow through the graph in either direction while retaining edge direction information. A concern with the use of bi-directional convolution in deep networks is that each unidirectional convolution must decrease the size of output features by a factor of at least 2 in order to avoid exponential blowup in the size of feature vectors as the graph propagates through the network. Due to the use of a *DenseNet*-style block with feature reduction built-in, this was not an issue here.

3.2 *DenseNet*-style blocks

Recent trends in deep learning for image processing suggest that including shorter "skip" connections between earlier stages and later stages in a deep convolutional network can be beneficial [9]. DenseNets [10] take this to a logical extreme, introducing direct connections from any layer in a block to all subsequent layers. We found a graphical analogue of this style of architecture very useful for this task. Suppose that $\mathbf{X}_{i-1}$ is the input of some convolutional layer H_i. Then, by analogy with DenseNets, H_i should also be given the outputs of previous layers as input:

$$\mathbf{X}_i = H_i\left(\mathbf{X}_0 \| \mathbf{X}_1 \| \ldots \| \mathbf{X}_{i-1}, \mathbf{A}\right)$$

However, in later layers this node-level input vector becomes very large for a computationally-expensive convolutional layer such as H_i. DenseNets also include measures designed to reduce the size of inputs to convolutional layers, such as 1×1 convolutions. We include an analogous "compression" fully-connected layer h, which reduces the input size before convolution by

allowing the network to project relevant node features from previous layers:

$$\mathbf{X}_i = H_i\left(h\left(\mathbf{X}_0 \| \mathbf{X}_1 \| \ldots \| \mathbf{X}_{i-1}\right), \mathbf{A}\right)$$

3.3 Graph Isomorphism Networks and Pooling

It has been shown that the standard graph convolution layer is incapable of distinguishing some types of graph. Since this task is almost entirely about graph structure and is known to be computationally hard, it was expected that the more-powerful Graph Isomorphism Networks [24] would produce better results, but this was not found to be the case. Similarly, localised pooling is well-known to be useful in image processing tasks, and its graphical analogues such as top-k pooling [5] and edge contraction pooling [2] also perform well on some benchmark tasks. These also appear useful for this task, perhaps corresponding to the human approach of simplifying propositional sub-formulae. However, these were also not found to be useful, possibly due to the lack of redundancy in formula graphs. Further investigations into integrating these powerful methods is left as future work.

3.4 Architecture

A simplistic neural architecture is described. Batch normalisation (BN) [11] is utilised before convolutional and fully-connected layers, and rectified linear units (ReLU) [13] are used as nonlinearities throughout, except for the embedding layer (no activation) and the output layer.

Embedding. An embedding layer maps one-hot input node features into node features of the size used in convolutional layers.

Dense Block. DenseNet-style convolutional layers follow, including the fully-connected network so that each layer consists ReLU-BN-FC-ReLU-BN-BiConv. Only one block is used, with each layer using all previous layers' outputs.

Global Average Pooling. At this point the graph is collapsed via whole-graph average pooling into a single vector. Passing forward outputs from all layers in the dense block to be pooled was found to stabilise and accelerate training significantly.

Output Layer. A fully-connected layer produces the final classification output.

A relatively large number of convolutional layers — 48 — are included in the dense block, for both theoretical and practical reasons. Theoretically, if information from one part of the graph must be passed to another some distance away in order to determine entailment or otherwise, then a greater number of layers can prevent the network running out of "hops" to transmit this information. Practically, more layers were found to perform better, particularly on the larger test categories, confirming the theoretical intuition. In principle there is no limit to the number of layers that might be gainfully included: the more layers, the larger the problems that may (theoretically) be tackled.

4 Experimental Setup

Source code for an implementation using the PyTorch Geometric [4] extension library for PyTorch [16] is available[1].

[1] https://github.com/MichaelRawson/gnn-entailment

Table 1: Network and Training Hyper-Parameters

network		**training**	
input features	4	batch size	64
convolutional features	16	momentum	0.9
convolutional layers	48	weight decay	0.0001
		initial min. learning rate	0.01
		initial max. learning rate	0.1
		learning rate decay factor	0.99995
		learning rate cycle length	8000

Training Training setup generally follows that suggested for DenseNets [10]: the network is trained using stochastic gradient descent with Nesterov momentum [21] and weight decay, with the suggested parameters. Parameter initialisation uses PyTorch's defaults: "Xavier" initialisation [7] for convolutional weights and "He" initialisation [8] for fully-connected weights. A cyclic learning rate [20] was found to be useful for this model — we applied a learning rate schedule ("`exp_range`" in PyTorch) in which the learning rate cycles between minimum and maximum learning rates over a certain number of minibatches, while these extremes themselves decay over time. Training continued until validation loss ceased to improve. See Table 1 for training parameter details.

Augmentation No data augmentation is used as the dataset is relatively large already, and further it is unclear what augmentation would be applied: the "symbolic vocabulary permutation" approach [3] is not applicable here due to the nameless representation, but randomly altering the structure of the graph does not seem useful as it could well change the value of y unintentionally. One could imagine a *semantic* augmentation in which A is made stronger or B weaker — this would produce data augmentation without invalidating the value of y.

Reproducibility Results are reproducible, but with caveats. Training runs performed on a CPU are fully deterministic, but tediously slow. Conversely, training runs performed on a GPU are not fully deterministic[2], but are significantly accelerated. The results reported here are obtained with a GPU, but produce very similar results on repeated runs in practice. This is a significant limitation of this work that we hope to address if and when a suitable deterministic implementation becomes available.

5 Propositional Results

Experimental results are shown in Table 2. Results reported from the best-performing models to date, PossibleWorldNet and TopDownNet ($d = 1024$) are also included verbatim, without reproduction, for comparison. Test scores of the best-performing model on each data split are highlighted. Results show that our model is competitive on all categories, both with algorithmically-assisted approaches (PossibleWorldNet), and with the best-known pure neural approach (TopDownNet). The model significantly outperforms all known approaches on the "massive" test category. We conjecture that our model generalises to some degree the approach

[2] An unfortunate consequence of GPU-accelerated "scatter" operations. See https://pytorch.org/docs/stable/notes/randomness.html

Table 2: Propositional Entailment Accuracy

model	valid	easy	hard	big	massive	exam
PossibleWorldNet	98.7	98.6	**96.7**	**93.9**	73.4	96.0
TopDownNet	95.5	95.9	83.2	81.6	83.6	96.0
EntailmNet	**99.4**	**99.3**	91.2	88.3	**89.2**	**97.0**

taken with TopDownNet. In EntailmNet arbitrary message-passing schemes within the entire DAG are permitted, rather than TopDownNet's strict top-down/recurrent approach, which may go some way to explaining the difference in performance. However, the relationship with PossibleWorldNet is less clear-cut, and this is reflected in results: PossibleWorldNet remains unbeaten on the "hard" and "big" categories, but is surpassed on all others.

6 First-Order Logic

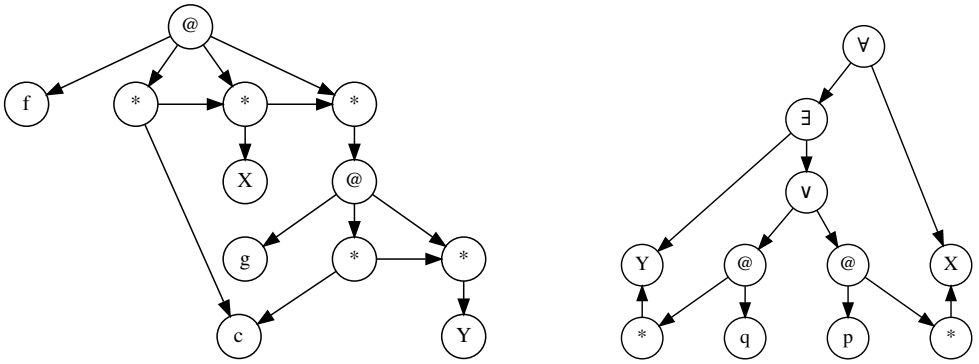
In order to demonstrate EntailmNet's generality, we design a new dataset for first-order logic. The dataset consists of a balanced mixture of valid and invalid sequents $F_1, \ldots, F_n \vdash G_1, \ldots, G_m$ expressed in full classical first-order logic. The task is to differentiate valid from invalid sequents.

Generation Producing a balanced set of randomised propositional entailment examples is computationally expensive, particularly if the formulae are generated completely randomly and checked for entailment afterwards [3]: this problem is only worsened in the first-order case. Instead, we employ a generative approach[3] in the style of natural deduction: first symbols and variables are randomly generated, then compound terms, and finally valid first-order sequents via randomised applications of rules in System LK [6]. Invalid sequents are produced by perverting the "axiom" rule such that any formula may entail any other formula. 100,000 sequents are used in this work, with 1,000 reserved for the test set.

Constraints Naïve generation of such sequents produces trivial, duplicative training examples easily discriminated by simple heuristics. To improve matters, we constrain generation. Generating α-equivalent samples is avoided by discarding sequents with duplicate graph representations. Trivial sequent sets can be made more complex by weighting the generated space toward more "interesting" rule applications: the rules of cut or quantifier introduction are used more often than weakening, for example. Limiting the number of available symbols makes invalid sequents closer in structure to valid ones and also generates more complex sequents.

Representation A similar representation to that in the propositional case is used here. However, argument order in function and predicate application must be preserved in order to maintain a lossless representation. This is achieved by use of an auxiliary "argument node" for each argument in an application, connected by edges indicating the order of arguments, shown in Figure 3. Quantifier nodes have two children: the variable which they bind, and the sub-formula in which the variable is bound. More space-efficient or performant graph representations are a possibility left as future work. 17 node types are used in total.

[3]Implementation and data available at https://github.com/MichaelRawson/fol-entailment-dataset.



(a) $f\,(c, X, g\,(c, Y))$ (b) $\forall x.\exists y.P(x) \vee Q(y)$

Figure 3: First-order graph encodings, showing (a) argument ordering and (b) variable binding.

Training and Results A very similar model and training regime to that used for propositional entailment is used for this classification problem, only differing in the number of convolutional layers: 32 rather than 48. More layers did not appear to harm performance but were not necessary in this case. The model achieved a classification accuracy of 98.3% on unseen problems. Training converged in a comparable length of time to propositional entailment.

Limitations We do not claim that this synthetic task is representative of real-world reasoning problems, or that the example generation procedure is completely free of systematic bias. Problems are also relatively small compared to those found in competitive benchmarks for first-order theorem provers. The experiment does show the flexibility of this approach, and the ability of the network architecture to perform well on more complex tasks than propositional entailment.

7 Conclusions and Future Work

We introduce EntailmNet, a new architecture for predicting propositional entailment and show that it has good performance characteristics, especially on larger entailment problems. The approach appears to work well with other logics, and performs well on a synthetic first-order benchmark. The network does not utilise any algorithmic assistance as PossibleWorldNet does, yet achieves competitive performance — this allows the network to process similar tasks which do not have a useful concept of "possible worlds". Additionally, the network is not based on a TreeNet, instead utilising graph neural network techniques. In some applications, such as guiding automatic theorem provers, network prediction throughput is crucial. Graph neural networks parallelise [4] more naturally than previous approaches, suggesting that this style of network may be more applicable to these domains.

Much future work is possible. No systematic effort has been made to tune network hyperparameters or overall architecture yet. In particular, we suspect that multiple dense blocks might use fewer parameters or perform better than one large block. Other convolution methods and the conspicuous absence of local pooling may also be investigated. In order to produce a comparative result, sensible baselines and other competitive models should be applied to the first-order benchmark. Additionally, the performance of this model suggests that further testing may require harder benchmark tasks. Real-world settings are yet to be investigated.

References

[1] Karel Chvalovský. Top-down neural model for formulae. In *International Conference on Learning Representations*, 2019.

[2] Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

[3] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? *arXiv preprint arXiv:1802.08535*, 2018.

[4] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[5] Hongyang Gao and Shuiwang Ji. Graph U-Nets. *arXiv preprint arXiv:1905.05178*, 2019.

[6] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[14] Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *arXiv preprint arXiv:1911.12073*, 2019.

[15] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *arXiv preprint arXiv:1905.10006*, 2019.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[17] Andrew M Pitts. Nominal logic: A first order theory of names and binding. In *International Symposium on Theoretical Aspects of Computer Software*, pages 219–242. Springer, 2001.

[18] Michael Rawson and Giles Reger. A neurally-guided, parallel theorem prover. In *International Symposium on Frontiers of Combining Systems*, pages 40–56. Springer, 2019.

[19] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[20] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[21] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initial-

ization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[22] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[23] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems*, pages 2786–2796, 2017.

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.