



Directed Graph Networks for Logical Entailment

Michael Rawson and Giles Reger

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

December 17, 2019

Directed Graph Networks for Logical Entailment

Michael Rawson Giles Reger

December 17, 2019

Abstract

We introduce a neural model for approximating propositional entailment, a benchmark task for logical reasoning, based upon learned graph convolutions on directed graphs. The model dispenses with some of the inflexible inductive biases applied in previous work on this domain, while still producing competitive results on the dataset. In particular, model performance on larger problems surpasses all previous work.

1 Background and Introduction

Neural networks are ubiquitous in tasks in which structure is extracted from raw data — tasks such as computer vision, or natural language processing — and rightly so, as their performance is unmatched. However, approximate reasoning on data that are already highly-structured is under-studied, but sorely needed in fields such as program synthesis or automated reasoning. We ourselves approach this area from researching automatic theorem provers for first-order logic: an undecidable setting that nevertheless might benefit greatly from heuristic guidance, as strategies for a subset of “useful problems” can be learned this way.

1.1 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 $A \models B$ 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.

1.2 Previous Work

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

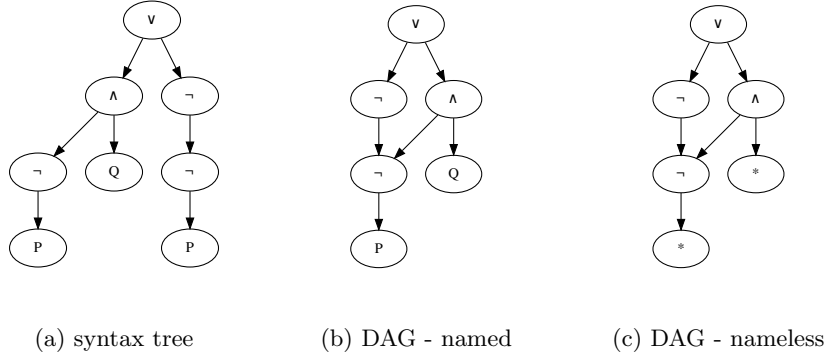


Figure 1: Producing an exemplar DAG representation of $(\neg P \wedge Q) \vee \neg\neg P$. A propositionally-equivalent formula can be retrieved from the final DAG.

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. 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 [13], while Paliwal et al [14] use undirected graph convolutions for a higher-order task. An interesting effort not directly related to this task is that of NeuroSAT [18], a neural network that learns to solve SAT problems presented in conjunctive normal form.

1.3 Graph Neural Networks

Graphs have historically proven difficult for learning algorithms of various varieties, mostly due to a very rich structure. However, recent advances [11] have produced a family of methods generally known as *Graph Neural Networks*, with *graph convolutions* as a central technique. These are typically simple, efficient networks that are practically useful for many tasks operating on graphical data.

1.4 Contributions

Our main contribution is a novel neural model based on graph neural network techniques that scores well on this dataset, in some cases beating PossibleWorldNet, yet does not suffer from the inductive bias applied by this approach. In particular, we introduce a directed graph convolution which appears crucial for this domain.

2 Input Representation

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 [22] or search guidance of automatic theorem provers [17]. 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.

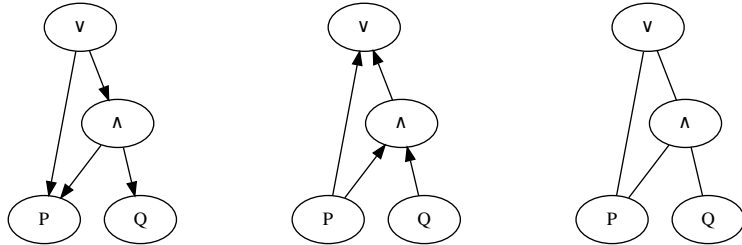
Bounded number of node labels. By use of nameless nodes, a finite number of different node labels are found in any DAG. This allows for one-hot node representations and does not require a separate textual embedding network.

Natural representation of bound variables. Representing bound variables such as those found in first-order logic can be difficult [16] — 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 solved in general by use of auxiliary nodes and/or edges such that an ordering can be retrieved. 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.



(a) top-down only (b) bottom-up only (c) both — lost structure

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

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 \mathbf{A} is the directed graph adjacency matrix. Various graph convolution operators [23] (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 [11].

However, this operator suffers from a shortcoming (illustrated in Figure 2) on DAGs such as those used here: information will only pass in one direction through the DAG, as messages propagate only along incoming edges. This is not necessarily a problem: bottom-up schemes such as TreeRNNs [21] exist, and Chvalovský shows [1] that a top-down approach can be useful here. However, to play to the strengths of the graphical approach the ideal would be to 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:

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

where the \parallel 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 [8]. DenseNets [9] 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:

$$\mathbf{X}_i = H_i(\mathbf{X}_0 \parallel \mathbf{X}_1 \parallel \dots \parallel \mathbf{X}_{i-1})$$

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 a “compression” fully-connected layer h , which reduces the input size before convolution:

$$\mathbf{X}_i = H_i(h(\mathbf{X}_0 \parallel \mathbf{X}_1 \parallel \dots \parallel \mathbf{X}_{i-1}))$$

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 [23] 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. 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) [10] is utilised before convolutional and fully-connected layers, and rectified linear units (ReLU) [12] are used as nonlinearities throughout, except for the embedding layer (no activation) and the output layer. See Table 1 for network hyperparameters.

Embedding. A single, fully-connected 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 linear layer produces the final output.

Table 1: Network and Training Hyper-Parameters

parameter	value
input features	4
biconvolution output features	16
convolutional layers	48
batch size	64
momentum	0.9
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

A large number of convolutional layers 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 [15] is available¹.

4.1 Training

Training set-up generally followed that suggested for DenseNets [9]: the network is trained using stochastic gradient descent with Nesterov momentum [20] and weight decay, with the suggested parameters. Parameter initialization followed PyTorch’s defaults: “Xavier” initialization [6] for convolutional weights and “He” initialization [7] for fully-connected weights in particular. A cyclic learning rate [19] 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.

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

¹<https://github.com/MichaelRawson/gnn-entailment>

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

stronger or B weaker — this would produce data augmentation without invalidating the value of y .

4.2 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 deterministic², 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 Results and Discussion

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 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 shown in these results: PossibleWorldNet remains unbeaten on the “hard” and “big” categories, but is surpassed on all others.

6 Future Work

Much future work is possible. No real effort has been made to tune network hyperparameters or overall architecture as yet. In particular, we suspect that multiple dense blocks might use fewer parameters and perform better than one very large block. Other convolution methods and the conspicuous absence of local pooling may also be investigated.

²An unfortunate consequence of GPU-accelerated “scatter” operations. See <https://pytorch.org/docs/stable/notes/randomness.html>

Clearly, small propositional logic problems are also not a very useful domain with the advent of efficient SAT solvers. However, the authors expect that this exercise on propositional problems will generalise well to many more-difficult domains, such as first-order logic. Producing benchmarks or useful applications is an interesting and unexplored direction.

7 Conclusions

We introduce EntailmNet, a new architecture for predicting propositional entailment and show that it has good performance characteristics, especially on larger entailment problems. 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 and can trade-off network accuracy with computational demands, suggesting that this style of network may be more applicable to these domains.

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] 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.
- [7] 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.
- [8] 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.
- [9] 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.

- [10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [12] 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.
- [13] Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *arXiv preprint arXiv:1911.12073*, 2019.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] Michael Rawson and Giles Reger. A neurally-guided, parallel theorem prover. In *International Symposium on Frontiers of Combining Systems*, pages 40–56. Springer, 2019.
- [18] 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.
- [19] 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.
- [20] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.
- [21] 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.
- [22] 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.
- [23] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.