Towards Determining How Deep Neural Models Learn to Reason

Anthony Marchiafava, Atriya Sen

Oklahoma State University
anthonymarchiafava@gmail.com, atriya@atriyasen.com

Abstract

Large Language Models (LLMs) are well known to perform poorly on tasks involving reasoning, including deductive, inductive, abductive, and spatial reasoning. This is evidenced by many existing benchmarks. While techniques such as chain-of-thought prompting and inference computation attempt to improve reasoning performance, it is necessary to assess whether large models are ultimately "memorizing" answers to the questions being used to assess their purportedly learnt reasoning capabilities. In this short paper, we describe a work in progress that aims to investigate the innate reasoning processes of large models by generating new questions that require deductive reasoning to answer. Initially, we train from scratch and then assess the ability of a recurrent deep neural model to make such a binary decision. We then discuss how our preliminary results bear on the hypothesis that some deep neural models can indeed learn to reason in the absence of memorization and semantic shortcuts, and conclude by discussing future work.

Code — https://github.com/rAIson-Lab/cnf-classification-with-rnn

Motivation

Numerous existing datasets illustrate the performance of *Large Language Models* (LLMs) on tasks in the *Natural Language Processing* (NLP) domain that require the model to correctly *reason*, implicitly or explicitly. This ability, and especially the ability of LLMs and other deep neural models to *explain* their reasoning in a human–understandable fashion, is an essential Explainable AI goal for achieving human trust in conclusions drawn by AI systems. Existing evaluatory datasets include those based on *first–order logic formulae* (Han et al. 2024b) and Natural Language Inference (NLI) datasets (Bowman et al. 2015; Williams, Nangia, and Bowman 2018; Nie et al. 2020).

There are also large deep learning models that are trained on mathematical problems, such as Llemma (Azerbayev et al. 2024), or focused on generating mathematical proofs, such as Baldur (First et al. 2023). These two models are especially of note, as Llemma is a language model for mathematical tasks in general as it was trained on a mathematically oriented dataset. Llemma is capable of chain-of-thought math problem solving and formal and informal mathematical proofs. Baldur is a model for generating first order logic proofs in specific. The goal of these models is generative in nature while our intention is to make something which is interpretable.

Transformer–based deep language models have been used for reasoning tasks involving generating formal proofs of correctness (Gontier et al. 2020; Polu and Sutskever 2020). These language models can improve performance using techniques such as *chain–of–thought prompting* (Wei et al. 2024). This use of intermediate reasoning steps can help models reason correctly on more difficult challenges. However, the performance on the metrics evaluated does not provide direct evidence that a model is in fact learning correct & generalizable reasoning strategies. This is shown by the GSM-Symbolic benchmark (Mirzadeh et al. 2024) which is itself inspired by a natural language math problem dataset (Cobbe et al. 2021). This work showed "that LLMs exhibit more robustness to changes in superficial elements such as proper names but are very sensitive to changes in numerical values" when it came to solving these numerical problems (Mirzadeh et al. 2024).

In justifying their choices specifically in the case of *in-context learning*, LLMs can exhibit untrustworthy reasoning and use *shortcuts*, which may lead to false conclusions. Examples of arithmetic reasoning tasks, when provided examples that were biased to always have the correct answer be the same letter from a set of multiple choices (for example if the correct choice is B from the options A, B, C), showed models would select the B option on subsequent questions instead of the correct one (Han et al. 2024a).

Further examination is necessary to provide evidence that a model is using reasoning in a way that can be shown. The approach we take in this short paper is as follows. We automatically generate a series of propositional formulae (as described in the following section), and use an *automated theorem prover* to determine whether each formula is a theorem or a non–theorem. While any propositional theorem–prover may be chosen for this task, we use the theorem prover *E*, as described in the following section. Propositional logic was chosen for simplicity; we intend to generalize to *first–order logic*, as will be described in the concluding section. Initially, we train "from scratch" on our dataset, and assess the

ability of our recurrent deep neural model to predict whether a propositional formula is a theorem or a non–theorem: a binary classification task.

Since this model is trained from scratch on a randomly generated dataset consisting of logical symbols in the form of propositional variable and therefore no semantic content, our hypothesis is that performance on this binary classification task indicates that our deep neural model indeed learns reasoning strategies in the absence of memorization and semantic shortcuts, providing evidence of the innate promise of such models on reasoning tasks, despite the limitations of the current state–of–the–art.

Methodology

Formula Generation

We chose to use propositional logic because of the ability of that type of logic to be clearly proven or shown that no such proof could be found. The form of the argument is the focus here instead of the truth or falsity of propositions. Disconnecting the meaning of the statements from the form of the statements allows the form of logic to be the topic of interest. This allows our work to focus on whether the neural network can learn the form of the argument. We do this to contrast with learning the semantics of the argument.

A literal in boolean logic is a statement that is either true or false or the negation of that sort of statement, and a clause is either a literal or a disjunction between two literals (Rich 2008). Here our literals are treated as propositions, but the form the logic takes in boolean or propositional logic is the same. To create a logical formula in conjunctive normal form (CNF) one needs to either have a single clause or have the formula made up of only the conjunction of two or more clauses. The CNF was chosen to generate the synthetic data created here because of the simplicity of the rules to generate formulas of arbitrary length. Axioms are logical formula we assume as true, while conjectures are logical formula we are checking follow the axioms. Multiple formulas were combined together to form axioms then another formula was chosen to be a conjecture in our logical theorems.

To illustrate the following is an example of an unfound input: $(a|\ b)\&(\ a|\ a).(\ a|b)\&(\ b).\ >\ (\ b|a)\&(a)$. We can imagine this as saying (a or not b) and (not a or not a) as one axiom, another axiom would be (not a or b) and (not b) imply (not b or a) and a. While $(b)\&(\ a|a).(\ b)\&(a|\ a).\ >\ (b)\&(b|b).$ is an example of an found input.

These arbitrarily created theorems could either be proven valid or invalid. To discover which were proven valid we used the automated theorem prover E. This theorem prover is a sound and complete prover for clausal first order logic with equality that also works with propositional logic (Schulz 2002). This theorem prover was used on the synthetic statements that were generated in CNF. These synthetic statements were provided as input to E with those statements we chose as axioms or conjectures marked as such. E would then provide a proof or show that the theorem was not provable. This data was collected and used as the input to a recurrent neural network which was trained to correctly classify if the input was something that could be proven or not.

RNN Classifier Training

The model was trained on 229,216 samples in conjunctive normal form. There were an equal number of theorems and non-theorems since the samples were generated synthetically.

A recurrent neural network (RNN) is a neural network which uses the output of hidden states as additional input to that network in subsequent input. The RNN was relatively shallow with only one recurrent layer. We tested five variations of this shallow network by changing the number of features in the hidden state (8, 16, 32, 64, and 128 features respectively). For this work we used PyTorch's default RNN implementation modified to incorporate input padding using packing padded sequences for batches of inputs whose lengths vary within the batch.

The data was generated by repeatedly applying these rules to a set of input literals, "a, b". For each literal we generated a literal using itself or its negation. Then we exhaustively combined each literal with every other literal using the rules specified for CNF. Since this process is automated we can generate any number of combinations/permutations of these literals, negations, conjunctions, and disjunctions. Following these rules we generated 420 unique logic statements by applying those rules to 20 disjunctions. Using these 420 statements, we then treated each statement as an axiom and for each statement treated it as a conjecture. This produces 176,400 unique combinations.

Then we added another axiom to each of the previously generated combinations, which produced another 73,911,600 combinations. Combining them together we created 74,088,000 combinations. However most of these combinations are non-theorems: conjectures that cannot be proven from the axioms. To help create a dataset that is more suitable for training we chose the group of samples in each category (theorem and non–theorem), shuffled the data, and selected a number of samples from each category equal to the number of the smaller set (here the set of proven theorems). Combined together this produced a final balanced dataset size of 229,216 (which is 114,608 samples from each category).

We chose to balance the dataset because our simple RNN classifier may not learn effectively from our initial highly imbalanced dataset. In general, it should not be necessary to produce such a balanced dataset for learning reasoning processes, i.e., it is conceivable that a different learning strategy could learn correct reasoning processes purely from examples of provable theorems, implicitly learning what constitutes a non–theorem. However this would constitute a more difficult learning task, so we avoid it here. In the Discussions section, we discuss a generative learning approach that always learns *constructively*, in the sense that even when a proof search fails, the data comprises of constructive steps towards that failed proof, not simply a negative label. In this sense this system will learn to construct a proof from exclusively positive examples of proof construction.

A recurrent neural network was used for this classification task. The recurrent neural network was trained for 20

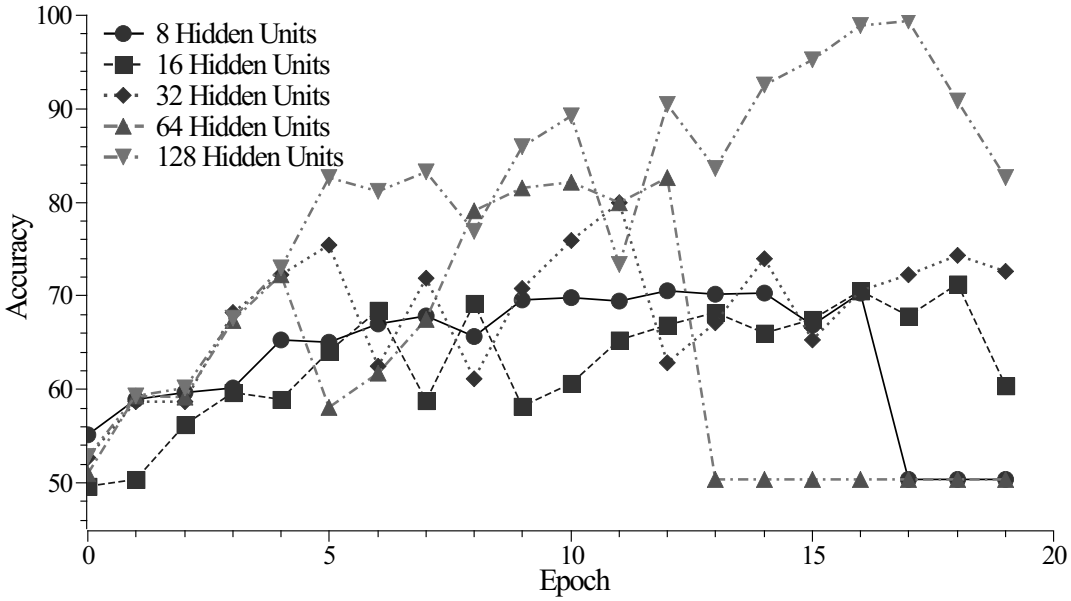

Figure 1: Testing accuracy grouped by number of hidden neural units

epochs, with a training set of eighty percent of the data and a testing set of twenty percent. Different iterations of the network were tried, choosing the model with the highest performance. We tested RNN's with 16, 32, 64, and 128 hidden units. To convert the dataset strings into RNN input we used our limited vocabulary to create an embedding (which would include a padding character) and batched our inputs into groups of 16, where each group is padded to be of equal length to the longest sample in that batch. Due to the input not being perfectly divisible by 16, we dropped the last batch.

Results

The models we trained were able to accurately predict the theorem–hood of a propositional formula, as shown in figure 1. The highest accuracy value observed was for the network with 128 hidden units (99.4% accuracy at epoch 17).

Discussion & Future Work

Classification is an achievable task With the limited dataset we created. With just these results, however, we cannot assert that the model has actually learned logical reasoning rules. It does, however, perform well on accuracy metrics.

Our results on the binary classification task reported here indicate that our deep neural model indeed learns reasoning strategies in the absence of memorization and semantic shortcuts, providing evidence of the innate promise of such models on reasoning tasks, despite the limitations of the current state–of–the–art.

The complete code and generated dataset for our experiments is currently redacted in the interests of anonymity, and will be made available upon publication under an open–source license.

Our binary classification model is an exploratory first step in the direction of a *generative* deep transformer-based AI model that is trained not only on the evaluation from a theorem prover of a formula being a theorem or a non–theorem, but also on the generated proof of theorem–hood (or failed proof of it). The generative model will then attempt to generate correct proofs on test propositional formulae. While much more challenging than the work reported here, this is natural next step, given the almost unreasonable efficacy of state–of–the–art generative models.

These generative models are much more complicated and difficult to interpret than the much simpler recurrent neural network used in this work. However, since we know the proof steps used to train the model further work in mechanistic interpretability can use facts we know about our entirely synthetically generated dataset to guide our search for interpreting if models we train are learning logical reasoning and where that reasoning is happening.

Neural Networks are often described as a black box. The internal workings of a model which is trained from random weights using backpropagation are challenging to interpret and understand. One approach to understanding this is through using mechanistic interpretability techniques. These techniques are used to discover the mechanisms which transform inputs into outputs (Bereska and Gavves 2024). Future work will focus on using one mechanistic interpretability technique, feature circuits, to search for model behavior in an automated fashion (Marks et al. 2024; Olah et al. 2020). These and other similar techniques can produce subgraphs or circuits of neural networks that indicate repeated behavior.

We will apply these techniques to models trained with the synthetic data generation technique shown here. This will

provide an opportunity to evaluate the behavior of models where we know step by step proofs, due to the generation of these proofs with a theorem prover. We believe that we will detect feature circuits related to the proof steps and that these feature circuits should be common to related proof steps. Prior work has focused on finding circuits for large language models like Claude (Templeton et al. 2024) but has not aligned the circuits to a formal logic task.

Acknowledgements

We would like to thank our highly insightful reviewers for highlighting improvements that we have incorporated into this publication. We also thank Justin Moua for engaging with us in discussion.

References

Azerbayev, Z.; Schoelkopf, H.; Paster, K.; Santos, M. D.; McAleer, S. M.; Jiang, A. Q.; Deng, J.; Biderman, S.; and Welleck, S. 2024. Llemma: An Open Language Model for Mathematics. In *The Twelfth International Conference on Learning Representations*.

Bereska, L.; and Gavves, S. 2024. Mechanistic Interpretability for AI Safety - A Review. *Transactions on Machine Learning Research*.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In Màrquez, L.; Callison-Burch, C.; and Su, J., eds., *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 632–642. Lisbon, Portugal: Association for Computational Linguistics.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.

First, E.; Rabe, M. N.; Ringer, T.; and Brun, Y. 2023. Baldur: Whole-Proof Generation and Repair with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, 1229–1241. New York, NY, USA: Association for Computing Machinery. ISBN 9798400703270.

Gontier, N.; Sinha, K.; Reddy, S.; and Pal, C. 2020. Measuring systematic generalization in neural proof generation with transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713829546.

Han, P.; Song, P.; Yu, H.; and You, J. 2024a. In-Context Learning May Not Elicit Trustworthy Reasoning: A-Not-B Errors in Pretrained Language Models. In *ICML 2024 Workshop on LLMs and Cognition*.

Han, S.; Schoelkopf, H.; Zhao, Y.; Qi, Z.; Riddell, M.; Zhou, W.; Coady, J.; Peng, D.; Qiao, Y.; Benson, L.; Sun, L.; Wardle-Solano, A.; Szabó, H.; Zubova, E.; Burtell, M.; Fan, J.; Liu, Y.; Wong, B.; Sailor, M.; Ni, A.; Nan, L.; Kasai, J.; Yu, T.; Zhang, R.; Fabbri, A.; Kryscinski, W. M.; Yavuz, S.; Liu, Y.; Lin, X. V.; Joty, S.; Zhou, Y.; Xiong, C.; Ying, R.; Cohan, A.; and Radev, D. 2024b. FOLIO: Natural Language Reasoning with First-Order Logic. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 22017–22031. Miami, Florida, USA: Association for Computational Linguistics.

Marks, S.; Rager, C.; Michaud, E. J.; Belinkov, Y.; Bau, D.; and Mueller, A. 2024. Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models. arXiv:2403.19647.

Mirzadeh, I.; Alizadeh, K.; Shahrokhi, H.; Tuzel, O.; Bengio, S.; and Farajtabar, M. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. arXiv:2410.05229.

Nie, Y.; Williams, A.; Dinan, E.; Bansal, M.; Weston, J.; and Kiela, D. 2020. Adversarial NLI: A New Benchmark for Natural Language Understanding. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4885–4901. Online: Association for Computational Linguistics.

Olah, C.; Cammarata, N.; Schubert, L.; Goh, G.; Petrov, M.; and Carter, S. 2020. Zoom In: An Introduction to Circuits. *Distill*. Https://distill.pub/2020/circuits/zoom-in.

Polu, S.; and Sutskever, I. 2020. Generative Language Modeling for Automated Theorem Proving. arXiv:2009.03393.

Rich, E. 2008. *Automata, Computability and Complexity: Theory and Applications*. Pearson Prentice Hall.

Schulz, S. 2002. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3): 111–126.

Templeton, A.; Conerly, T.; Marcus, J.; Lindsey, J.; Bricken, T.; Chen, B.; Pearce, A.; Citro, C.; Ameisen, E.; Jones, A.; Cunningham, H.; Turner, N. L.; McDougall, C.; MacDiarmid, M.; Freeman, C. D.; Sumers, T. R.; Rees, E.; Batson, J.; Jermyn, A.; Carter, S.; Olah, C.; and Henighan, T. 2024. Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet. *Transformer Circuits Thread*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713871088.

Williams, A.; Nangia, N.; and Bowman, S. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In Walker, M.; Ji, H.; and Stent, A., eds., *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 1112–1122. New Orleans, Louisiana: Association for Computational Linguistics.