Qeios

Limitations of and Lessons from the Learning of Large Language Models.

Reinhard Oldenburg, Math department, Augsburg University, Reinhard.oldenburg@math.uni-augsburg

ABSTRACT

It is argued that the Curry-Howard correspondence for classical logic implies limitations for logical reasoning that can be learned and performed by large language models. The correspondence establishes an isomorphism between proofs in logic and programs in functional typed lambda calculus. While intuitionistic logic maps to a version of lambda calculus that can be carried out in a local way, i.e., considering local parts of the code in isolation, the version of lambda calculus that corresponds to classical logic requires non-local relations – and this non-locality cannot be learned by large language models due to their restriction to investigate a relative short sequence of tokens. A possible way to go beyond this limitation is sketched as well. Implications for other areas are investigated as well.

KEYWORDS

Large language model, Curry-Howard-Isomorphism, Logic, Proof Theory

INTRODUCTION

It is pretty safe to say that nobody really understands how large language models produce their results. Their learning process is quite different from the learning of language and argumentation by humans. Yet, a better understanding of their limits could also shed light on the particular strengths of human thought.

At least since the introduction of ChatGPT public interest in large language models sparked to new heights. A large strand of scientific analysis of such systems has concentrated on the abilities for logical reasoning (see e.g., Evans et al., 2018; Friedmann, 2023) and this is still investigated in recent research (e.g., Liu, 2023). Findings are that logical reasoning is limited, but also successful in many cases. However, it seems that there is a lack in theoretical understanding of the principal limitations of such systems. The present paper is indented as a small brick to fill this hole. It will argue that the structure of classical logic implies that it cannot be completely learned by large language models. The relevant property of logic that allows this conclusion is, however, only visible when applying the appropriate form of the Curry-Howard isomorphism that maps logic to functional computer programs.

The results of this paper do, of course, not imply that large language models cannot give answers derived by classical logic if these results were in some form in the training data. The restriction that will be discussed only applies to original logical reasoning.

One idea to address the restrictions of LLMs in the domain of logical reasoning is to combine a LLM with a classical symbolic inference engine into a hybrid system (e.g., Pan et al., 2023). However, if the limit described here is, in fact, the essential bottleneck, then another approach might be feasible and should avoid the drawbacks that are associated with hybrid systems.

In the conclusion, some possible implications for philosophy of language and education are briefly sketched.

https://doi.org/10.32388/9FH6AD

LARGE LANGUAGE MODELS

Large language models like Google Bard or OpenAI's ChatGPT (e.g., Vaswani et al., 2017; Radford et al., 2019) take a sequence of input tokens or words and produce tokens or words that, given a large base of text, are likely to follow the input string. The meaning (relative to natural language) of the tokens is fixed by the translation of training text to token sequences and vice versa. The crucial point is that input sequences are of a certain fixed length. In current models, this fixed length is about several 1000s – which is impressive from a technical point of view, but this is still a finite number, and this is the only thing of information needed in the current argumentation. This length limits the model's ability to refer non-locally to distant parts of the input stream.

CURRY-HOWARD CORRESPONDENCE

This correspondence connects logical proofs and functional programs, see Girard, Lafont, and Taylor (1989), Sørensen and Urzyczyn (2006) and Mimram (2020). The theory is far too complicated to be explained here, but as a starter for readers who don't know it: Consider a function f that maps objects of type A to type B, the type of such a function is denoted by $A \to B$. Assume, one has an object $a: A$, i.e., an object a of type A. Then one can do the following type-inference: $a: A$ and $f: A \to B$ implies $f(a): B$. Restricting to the "type part" one has A and $A \to B$ imply B, which is exactly modus ponens in proofs theory of propositional calculus.

Obviously, this kind of argument can be done with different programming languages. The theoretical minimal example is the lambda calculus. Expressions of this calculus consist of variables (symbols, such as x), functions $\lambda x.\, a$ and function application $(f\ g)$ where f is a lambda expression and g is any expression. $\lambda x.\, a$ denotes a function in the variable x which is defined by the expression a. For example, $\lambda x.\, x$ is the identity function. The calculus has essentially the following rules:

- Function application: $(\lambda x.\, a\ b) \to a[x \coloneqq b]$ (all occurrences of x in a are replaced by b)
- Renaming of variables $\lambda x.\, a \to \lambda y.\, a[x \coloneqq y]$ (if y does not appear free in a)

The rules both for forming syntactically correct expressions as well as the transformation rules of the calculus are local, in the sense that only a limited part of the whole expression needs to be considered. Essentially, only $(\lambda x.\, a\ b)$ (which might be a local part of a much larger expression) need to fit into "working memory" and all substitutions are applied within this expression. Of course, local does not necessarily mean small: In $(\lambda x.\, a\ b)$ the expression a might be rather long, already exceeding the length of what a LLM can store in its current sequence of input tokens. However, many typical reasonings work with limited length expressions.

For the Curry-Howard correspondence, typed versions are needed: In typed lambda calculus a function takes the form $\lambda x: T_1\, a: T_2$ where x is a variable of type T_1 with result type T_2. Of course, typing judgements restrict the set of correct expressions.

It was found that proofs in intuitionistic logic and expression (i.e., programs) in typed lambda calculus. The correspondence in detail is displayed in table 1. It is important to see that this isomorphism conserves the structure in the sense that sub-proofs map to sub-expressions (sub-programs). This is important here as it implies that long proofs map to lo long programs and vice versa.

Table 1. The Curry-Howard-correspondence

Logic	Programming
Proof	expression

Proposition	data type
Implication	function
Unprovable proposition	empty type
Conjunction	type of cartesian pairs
Disjunction	injection type

Intuitionistic logic is interesting because it leads to constructive mathematics, but most of standard mathematics is based on classical logic which has the law of double negation ($\neg\neg a = a$), or equivalently the law of "excluded middle" ($a \vee \neg a =$true). About 30 years ago, it was realized that the correspondence can be extended to classical logic by including non-local control operations in the lambda calculus. There are various forms of this (because there are also different versions of classical logic), but the most often cited version is lambda-mu calculus by Parigot (1992). This calculus adds another kind of variables in order to store "addresses" and enable "jumps" in the code, and therefore allows non-local relations.

More formally, lambda-mu calculus extends ordinary lambda-calculus by a second set of variables ($\alpha, \beta,$ ...), another class of expressions (named terms, in addition to the unnamed terms of standard lambda calculus) and two syntactic elements:

Unnamed terms of the form $\mu\alpha.\, e$ where e is a named term.

Named terms of the form $[\alpha]a$, where a is an unnamed term (and all named terms are of this form)

The idea is that $[\alpha]a$ names a term a by α so that it can be inserted somewhere else by application of the μ operator. Therefore, two new reduction rules were introduced by Parigot (1992):

- $(\mu\alpha.\, a\ b) \rightarrow \mu\alpha.\, a[[\beta]c \rightarrow [\beta](c\ b)]$ where a and c are named terms and b is unnamed, and the substitution is recursively applied in every subexpression of a.
- $[\alpha]\mu\beta.\, a \rightarrow a[\beta \rightarrow \alpha]$

While the reduction rule $(\lambda x.\, a\ b) \rightarrow a[x \coloneqq b]$ of the ordinary lambda-calculus applies the substitution only to the variables and removes the λ, the structural recursion substitution keeps the μ-environment and substitutes larger subexpressions, not just atomic variables. Moreover, it does this by a recursive procedure that must examine every sub-expression. This is more complex even when working with tree-like expression structures than standard substitution, and it is even more demanding when working with sequences of tokens where the relevant places need to be identified.

De Groote (1994) has shown that the lambda-mu-calculus can be reduced to lambda calculus with continuation passing style. However, this reduction implies that each function application must also provide the continuation, which is an expression of the same order of length as the whole program. Thus, this accumulates to giant expressions, as it captures the content of the stack when the expression is evaluated on a stack machine. The role of the stack is thus to provide reference to information that would otherwise blow up the expressions.

This complexity explosion comes from the fact that the reduction rules give lambda-mu-calculus the same non-local flow-control as realized by continuations in some functional programming languages, such as Scheme – see Hofmann and Streicher (1997) for a proof. These non-local jumps require a management of addresses and in realizations working with sequences of tokens both start point and end point of non-local operations must be present in the string under consideration.

To make the argument, that reducing expressions in lambda-mu-calculus typically requires to focus on longer expressions than reducing lambda-calculus expressions, consider the following example (taken from https://www.pls-lab.org/en/Lambda_mu_calculus) that realizes Pierce's law in (typed) lambda-mu calculus:

$$\lambda f.\mu\alpha.[\alpha](f(\lambda y.\mu\beta.g([\alpha]y)):\big((A \to B) \to A\big) \to B$$

In this example the types of sub-expressions are $\alpha: A, y: A, \beta: B, \quad \lambda y.\mu\beta.g([\alpha]y): A \to B, g: B \to B$. Here, g may be arbitrarily complex and the "continuation" α defined on the left must be considered throughout reducing g.

THE ARGUMENT

Based on the theory sketched above, the main argument of this paper can be given in a very compact form: Full classical logic (in contrast to intuitionistic logic) cannot be realized by calculi that work locally on expressions but requires non-local flow of control in the transformation of expressions. This argument lives on the "program side" of the Curry-Howard correspondence, but due to the correspondence carries over to logic. On an intuitive level, this is clear as well: Assume a proof starts with considering $a \vee \neg a$ and goes on to infer from$\neg a$ which, after a longer derivation, gives a contradiction, then one has to "jump back" to $a \vee \neg a$ and can infer a.

CONSEQUENCES AND DISCUSSION

The paper has argued that the way of learning that is realized in current large language models limits themselves to intuitionistic logic. Classical logic, especially for longer argumentations, requires non-local relations that are beyond the scope of current LLMs. It is, however, difficult to demonstrate this result empirically, i.e. to prompt e.g. ChatGPT with logical reasoning tasks that are based on classical logic and demonstrate that it fails. This has several reasons. The argumentation only applies to larger logical derivations that do not fit into the finite sequence of tokens and such arguments may fail for many different reasons. Moreover, of course, all such models learned many results from classical logical reasoning which exists in the training data without the ability to perform the reasoning by themselves.

There may be a constructive lesson to be learned from the observation above: The limitation of a LLM in logical reasoning comes from its limited length of strings of tokens. Maybe LLM reasoning could be improved by augmenting the string of input tokens by some limited string of symbols that refer to some place in the input string and gets set during the operation of the network. Or, alternatively, a part of the input sequence should be used as a kind of stack similar to the standard execution of functional programs including continuations.

This work has theoretical implications as well: The discussed limitation faced by LLM should also be relevant for the idea of language learning by observing language use, and more generally, to Wittgenstein's idea that meaning is created by the language game (Wittgenstein, 1953). The fact that the training process of LLMs takes much more computational power and storage than a brain, and that it takes excessive more training material to reach near human language competence needs an explanation than can be given e.g. by theories of embodiment, interactionism or inherited structures for language learning. It also strengthens the point of view that reference is important in logical reasoning and thinking, and this is important for debate about the correct interpretation of quantifiers (Hand, 2007).

Last not least, the argument may have relevance for the learning of logical reasoning by humans as well, e.g. it may explain why proofs by contradictions are so difficult for students.

In summary, this paper shall have relevance for philosophy, computer science, and education.

Acknowledgements: I thank Y for discussion.

REFERENCES

de Groote, Ph. 1994. A CPS-translation of the λμ-calculus. In: Tison, S. (eds) *Trees in Algebra and Programming — CAAP'94*. CAAP 1994. Lecture Notes in Computer Science, vol 787. Berlin, Heidelberg; Springer. https://doi.org/10.1007/BFb0017475

Evans, R. et al. 2018. Can Neural Networks Understand Logical Entailment? arXiv:1802.08535.

Friedman, R. 2023. Large Language Models and Logical Reasoning. *Encyclopedia* 3(2), 687-697. https://doi.org/10.3390/encyclopedia3020049

Girard, J.-Y., Lafont, Y., and Taylor, P. 1989. *Proofs and Types.* Cambridge: Cambridge University Press.

Hand, M. 2007. Objectual and Substitutional Interpretations of the Quantifiers. In D. Jacquette (Ed.) *Handbook of the Philosophy of Science, Philosophy of Logic*, North-Holland. 10.1016/B978-044451541-4/50020-8

Hofmann, M., Streicher, Th. 1997. Continuation Models are Universal for Lambda-Mu-Calculus. *Proceedings, 12th Annual {IEEE} Symposium on Logic in Computer Science*, 387-395, doi 10.1109/LICS.1997.614964

Liu, H. et al. 2023.Evaluating the Logical Reasoning Ability of ChatGPT and GPT-4. https://doi.org/10.48550/arXiv.2304.03439

Mimram, S. 2020. *Program=Proof.* Independently published. http://www.lix.polytechnique.fr/Labo/Samuel.Mimram/teaching/INF551/course.pdf

Pan, L. et al. 2023. LOGIC-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning". arXiv:2305.12295v1

Parigot, M. 1992. Lambda-my-calculus: An algorithmic interpretation of classical natural deduction". In *Proceedings of the International Conference on Logic Programming and Automated Reasoning, LPAR '92*, 190–201, London: Springer.

Radford, A., Narasimhan, K., Rocktäschel, T., et al. 2019. Improving Language Understanding by Generative Pre-training". OpenAI Blog 1 (8). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Sørensen, M., Urzyczyn, P. 2006. *Lectures on the Curry–Howard isomorphism*. Studies in Logic and the Foundations of Mathematics, vol. 149. Elsevier.

Vaswani, A. et al. 2017. Attention is All you Need", arXiv:1706.03762

Wittgenstein, L. 1953. *Philosophical Investigations.* Hoboken: Blackwell.