Combining LLM, RL, and Logic

King-Yin Yan[0009−0007−8238−2442]

general.intelligence@gmail.com

Abstract. This paper has two main ideas: The first is to explain two types of LLM (large language model) + RL (reinforcement learning) architectures, which are probably known in "folklore" but not explicitly stated. The second idea is based on what we call "Type L" architectures. Under this perspective, formal logic is essentially the same as natural language, and so LLMs (language models) are the same as logic processors. We briefly considered differentiable "Logic Transformers" but realized that current Transformers have very strong advantages over other alternatives.

Keywords: AGI · large language models · reinforcement learning · neural-symbolic integration

Part I. LLM + RL architectures

For **string diagrams** [5] there are usually two conventions: 1) data are nodes, functions are edges: $(x) \xrightarrow{f} (y)$ or alternatively 2) functions are nodes, data are edges: $\xrightarrow{x} (f) \xrightarrow{y}$. In the following, I make explicit nodes for both functions (grey) and data (white), whereas edges merely represent linkages.

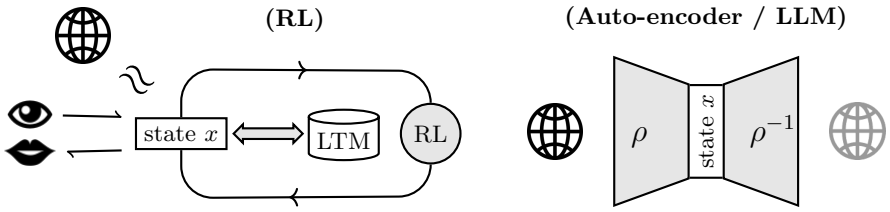


Fig. 1. The eye represents observations and the mouth (speech) actions. Because RL has to maximize rewards, its internal representation (the state) must eventually approach a good approximation of the world. LTM = long term memory, which works by associative recall and condensation, but will not concern us in this paper. The auto-encoder, of which LLMs are a special case, works by compressing world-data (via ρ) and de-compressing (via ρ^{-1}) to re-construct the data (grey world).

First let's recall the **fundamental forms** of RL (Fig.1 left) and auto-encoders (right). The question is how to combine them, for which we can ignore long-term

memory (which we will eventually see, is just a very large Working Memory). Both RL and LLM have the "state" x in common but there are subtle differences as to how to interpret these states (Fig.1). Initially my research favored **Type W**, where the LLM is interpreted as a **world model**. This approach suffers the problem that we don't really understand the internal state (marked "?") of LLMs, which consists of many layers of Transformers and their intermediate token outputs, so we don't know how to "merge" the RL state with the LLM state.

It seems that **Type L** would be easier to work with, and it is also the "mainstream" approach, along the direction of RLHF [8]. Here, the LLM simply loops over itself to complete the RL cycle. LLM models the "spoken thoughts" of human thinking, as found in text corpora. These thoughts are interpreted as internal states of the RL. Ever since Richard Montague's success in converting a fragment of English into formal logic [9] [10], the line between formal logic and natural language is blurred. We should not obsess with the idea that "logic" must be some kind of cryptic, undecipherable code. A new perspective is that *the "representation" of human internal thought is just natural language* (or very close to it).

Type L also has the advantage that we can directly examine the internal state of an AGI. What are currently called "prompts" is just **Working Memory**. We can think of the prompt as something akin to our "**inner speech**" when thinking [4].

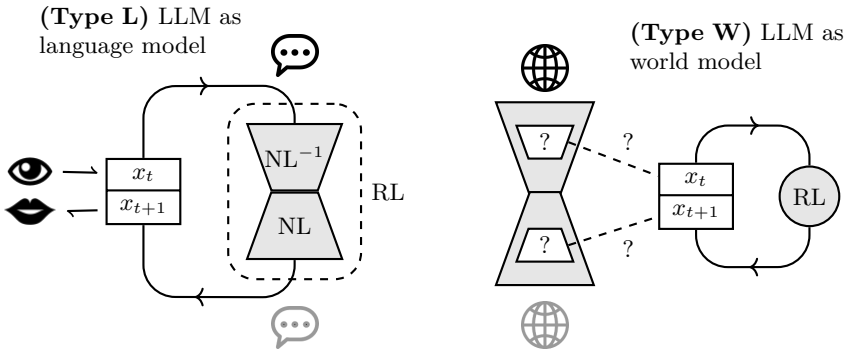
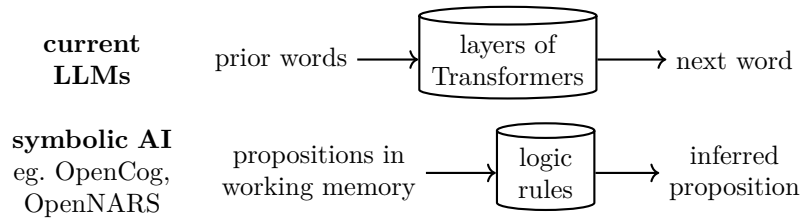


Fig. 2. Two types of RL + LLM architectures. NL^{-1} is a map that compresses natural language to a hidden representation (not shown explicitly).

The hallucination problem. The significance of combining RL and LLM is such that the system can *think in loops*. Reward maximization in the RL loop forces better thoughts to win, ie. those thoughts that are logically coherent with the rest of the learned knowledge. This solves the problem of "hallucinations" when LLMs are just parroting human speech.

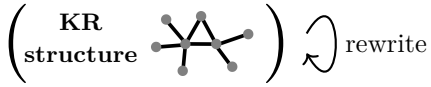
Part II. "Logic"

As we know, LLMs predict the next word per each iteration. For a long time I have been trying to reconcile the following two views:



until I had the insight that a logic-based system could output a proposition to rewrite part of a linguistic tree or graph, similar to the LLM outputting the next word to complete a sentence.

This leads us to the following basic architecture, which is also the setup proposed by Goertzel in [3]:



(1)

One way to define a rewrite rule [6] is as a pair of morphisms $L \leftarrow K \rightarrow R$, L and R are the left and right hand sides, and K is the interface graph. G is the input graph and H is the result of rewriting. The effect is that the part of L outside K is deleted from G and is replaced by the part of R outside K:

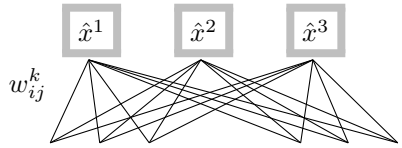
$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array} \tag{2}$$

This occurs in a category where graphs are objects and homomorphisms define some notion of "matching". Now if we imagine a **differentiable** version of rewriting rules (see (3) below [1]), all the rules would be represented by a set of parameters $\Theta \in \mathbb{R}^n$, and the parameterized rules *must* cover the entire family of rewrite rules as Θ vary continuously (in practice, we can limit the rules length to make that possible). This requires the use of "**discretizing**" functions such as sigmoid and softmax, and may be formulated by a fairly mechanical process. However, in doing so we have overlooked the possibility that a **deep** neural network such as Transformer may be able to aggregate all the contents of Working Memory, output some intermediate **distributive** representations, before finally

[1] with experimental code available on https://github.com/Cybernetic1/reinforcement-learning-experiments

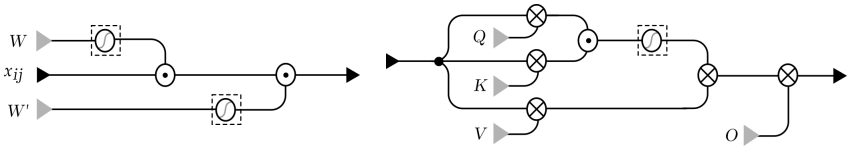
outputting some discrete tokens. Such a process would imply L matches the entire input graph G in (2).

Part of a differentiable rewriting rule may look like this:



$$\mathsf{father}(x_{11}, x_{12}, \bullet_{13}) \wedge \mathsf{father}(\bullet_{21}, x_{22}, x_{23}) \rightarrow \mathsf{grand\text{-}father}(x_1, \bullet_2, x_3) \tag{3}$$

which has a form quite similar to the **Self-Attention** of Transformers (right):



(4)

but the rule (3) neglected the fact that the variables in each father atom must match their variables with each other (a situation well-known in classical logic-based AI). This means we have to programmatically feed all *pairs* of Working-Memory atoms, ie. the Cartesian product $A \times A$, into the above string diagram. That makes it different from the Transformer which takes its input in plain form. The reason why Transformer can do that is because it has many **layers**, and the matching of $A \times A$ can be achieved over these layers.

Indeed, neural **associative memory** (such as Transformer, which are shown to be equivalent to Hopfield Networks [11]) may be a far superior form of "logic" than traditional, formal symbolic logic. Consider the logic example:

$$\text{loves}(X, Y) \wedge \neg\text{loves}(Y, X) \rightarrow ☹(X) \tag{5}$$

but we immediately sense that the human brain does not work like that. It is more like: the thought of unrequited love fires a massive number of neural activations which all point to the conclusion that this is an unhappy experience. The Transformer's associative memory does exactly that.

Also recall that Self-Attention originated as a kind of **content-addressable memory** for Neural Turing Machines, invented for its differentiability. In retrospect, it is also a very efficient implementation of associative memory. If we want to improve the Transformer, we should first understand it as an associative memory.

From a **functorial** point of view (eg.[2] [1] [7]), one can find a functor that transforms a category $\mathcal{C}$ of KR structure and rewrite morphisms into another category $\mathcal{D}$ with different KR structure and rewrite morphisms. It might just happen that one type of rewrite morphisms are computationally less costly than the other type, and so we have discovered a better, more efficient AGI system. But this maneuver seems not so useful in light of the fact that Transformer / Self Attention is a distributive associative memory that may not care too much about its input representation.

Conclusion

We proposed a simple AGI architecture that looks promising. If the Transformer cannot be improved much further, we should perhaps accept the current design and proceed to the "engineering" phase of AGI development.

References

[1] Vincent Abbott. *Neural Circuit Diagrams: Robust Diagrams for the Communication, Implementation, and Analysis of Deep Learning Architectures*. 2024. arXiv: 2402.05424 [cs.LG]. URL: https://arxiv.org/abs/2402.05424.

[2] Francesco Riccardo Crescenzi. *Towards a Categorical Foundation of Deep Learning: A Survey*. 2024. arXiv: 2410.05353 [cs.LG]. URL: https://arxiv.org/abs/2410.05353.

[3] Ben Goertzel. *The General Theory of General Intelligence: A Pragmatic Patternist Perspective*. 2021. arXiv: 2103.15100 [cs.AI]. URL: https://arxiv.org/abs/2103.15100.

[4] Pentti Haikonen. *Consciousness and Robot Sentience*. World Scientific, 2012.

[5] Ralf Hinze and Dan Marsden. *Introducing String Diagrams - the art of category theory*. Cambridge University Press, 2023.

[6] Kennaway et al. "An Introduction to Term Graph Rewriting". In: *Term Graph Rewriting - Theory and Practice*. Ed. by Sleep, Plasmeijer, and Eekelen. John Wiley & Sons, 1993.

[7] Nikhil Khatri et al. *On the Anatomy of Attention*. 2024. arXiv: 2407.02423 [cs.LG]. URL: https://arxiv.org/abs/2407.02423.

[8] Zihao Li, Zhuoran Yang, and Mengdi Wang. *Reinforcement Learning with Human Feedback: Learning Dynamic Choices via Pessimism*. 2023. arXiv: 2305.18438 [cs.LG]. URL: https://arxiv.org/abs/2305.18438.

[9] Richard Montague. "English as a formal language". In: *Edizioni di Communita.* (1970). Ed. by B. Visentini, pp. 188–221.

[10] Richard Montague. "The Proper Treatment of Quantification in Ordinary English". In: *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*. Ed. by K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes. Dordrecht: Springer Netherlands, 1973, pp. 221–242. ISBN: 978-94-010-2506-5. DOI: 10.1007/978-94-010-2506-5_10. URL: https://doi.org/10.1007/978-94-010-2506-5_10.

[11] Hubert Ramsauer et al. *Hopfield Networks is All You Need*. 2021. arXiv: 2008.02217 [cs.NE]. URL: https://arxiv.org/abs/2008.02217.