Combining LLM and RL, and Logic Transformer

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 on the internet but not explicitly stated. The second idea is developed 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. Thus we introduce the Logic Transformer which may have advantages over traditional Transformers.

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

Part I. LLM + RL architectures

For "string diagrams" 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.

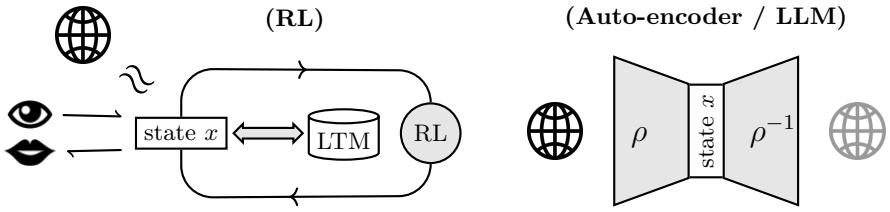


Fig. 1. The eye represents observations and the mouth (speech) actions. Because RL has to maximize rewards, its internal representation (the state) must eventually approach a good approximation of the world. LTM = long term memory, which works by associative recall and condensation, but will not concern 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 the LTM

module. 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, of which RLHF seems to be an instance. Here, the LLM simply loops over itself to form an RL loop. The 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 [**Montague**], 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. This leads to a novel idea: *the "representation" of human internal thought is just natural language.* 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**.

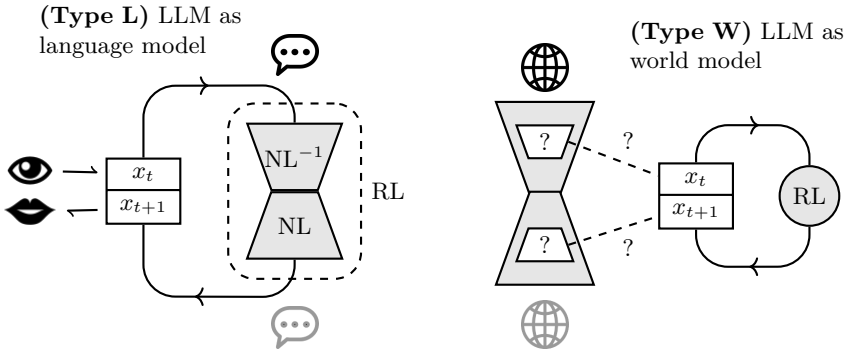


Fig. 2. Two types of RL + LLM architectures. NL^{-1} is a map that compresses natural language to a hidden representation (not shown explicitly).

Part II. Logic Transformer

Having convinced ourselves that LLMs can be a model of *internal* thoughts, we wonder if the Transformer can be re-formulated as some kind of logic engine or general **rewriting system**, defined by rewriting **rules** that can be learned via gradient descent?

In order for such rules to be **differentiable**, we can't have N rules at one moment and then suddenly $N + 1$ rules the next moment (that would not be

differentiable). The only solution I can think of is to maintain a *fixed* number, M, of rules. Every rule may potentially "morph" into any other rule in this space of rules. The set of rules would look like a *rectangular* matrix:

$$\begin{array}{ll}
\text{rule 1:} & \boxed{X^1_{11}...X^1_{1I}} \wedge \boxed{X^1_{21}...X^1_{2I}} \wedge \ ... \ \boxed{X^1_{K1}...X^1_{KI}} \rightarrow \boxed{X^1_{01}...X^1_{0I}} \\
\text{rule 2:} & \boxed{X^2_{11}...X^2_{1I}} \wedge \boxed{X^2_{21}...X^2_{2I}} \wedge \ ... \ \boxed{X^2_{K1}...X^2_{KI}} \rightarrow \boxed{X^2_{01}...X^2_{0I}} \\
.... & .... \\
\text{rule M:} & \boxed{X^M_{11}...X^M_{1I}} \wedge \boxed{X^M_{21}...X^M_{2I}} \wedge \ ... \ \boxed{X^M_{K1}...X^M_{KI}} \rightarrow \boxed{X^M_{01}...X^M_{0I}}
\end{array} \tag{1}$$

For a rule to be applicable, its atoms must match with propositions in Working Memory. An atom may contain constants (such as "Socrates", that must be matched correspondingly) or variables (such as x, that matches any entity). To achieve this, I introduce a trick called **cylindrification factor** (Fig.3).

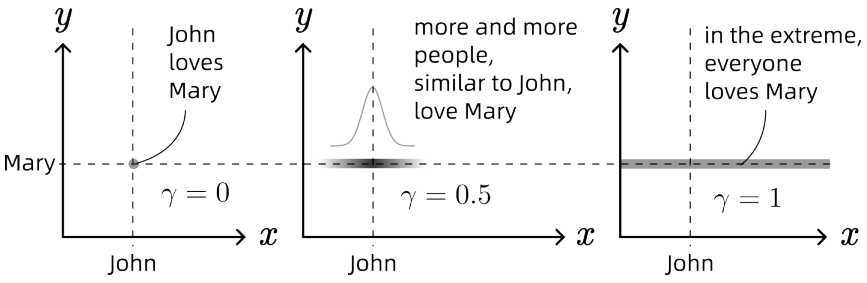
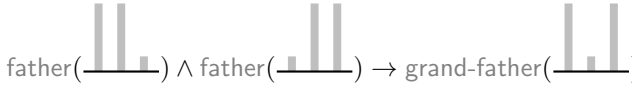


Fig. 3. Illustration of the cylindrification factor γ.

To make **variable substitution** differentiable is even trickier. Consider this logic rule:

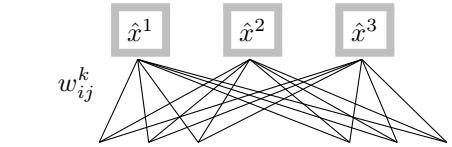
$$\text{father}(X_1, X_2) \wedge \text{father}(X_2, X_3) \rightarrow \text{grand-father}(X_1, X_3) \tag{2}$$

We have to imagine some "slots" where we "select" variables using weights and **softmax**:



(3)

The actual implementation leads to formulas like these:



$$\text{father}(x_{11}, x_{12}, \bullet_{13}) \wedge \text{father}(\bullet_{21}, x_{22}, x_{23}) \rightarrow \text{grand-father}(x_1, \bullet_2, x_3) \tag{4}$$

$$\hat{x}^k := \underset{\text{predicates}}{\forall i \in} \underset{\text{arguments}}{\forall j \in} \left\langle \underset{ij}{\text{soft}\max} w_{ij}^k \ , \ x_{ij} \right\rangle \tag{5}$$

which is similar to the **Self-Attention** of Transformers:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{soft}\max \frac{\langle \mathbf{Q}, \mathbf{K} \rangle}{\sqrt{d_k}} \mathbf{V} \tag{6}$$

except that softmax does not commute with inner products, so they are not exactly equivalent, but qualitatively similar. Let's pause for a moment to reflect that we tried to substitute variables (some kind of syntactic manipulation), and ended up with something like Self-Attention. Also recall that Self-Attention originated as a kind of **content-addressable memory** for Neural Turing Machines, invented for its differentiability.

A final obstacle concerns the limitations of first-order logic. As Ben Goertzel pointed out during an online discussion, my earlier design lacked mechanisms for handling **Skolem functions** and thus existential quantifiers. One way to get around this problem is to define $\exists$ via a set of **higher-order logic** rules. Without explicitly specifying how to do so (though I believe this can be done), we simply use a more powerful "higher-order" logic in the sense that rules admit substitutions in all (predicate or argument) positions, and hope that machine learning will learn the required axioms.

Armed with these ideas, it is not too difficult to work out a new kind of differentiable "Logic Transformer" (Fig.4). A detailed explanation is contained in my on-going research thesis [**<empty citation>**].

Conclusions

The significance of combining RL and LLM is such that the system can *think in loops*, eliminating thoughts that contradict accepted truths, thus achieving genuine, logically coherent understanding, instead of just parroting human speeches. This would also solve the problem of "hallucinations". The remaining question is how to train the Type L model *efficiently*, but this is more of an engineering rather than theoretical problem.

A very interesting question concerning Part II is how to handle **variable-length propositions**? If this could be done, the neural network may have a form very close to or even identical to current Transformers...

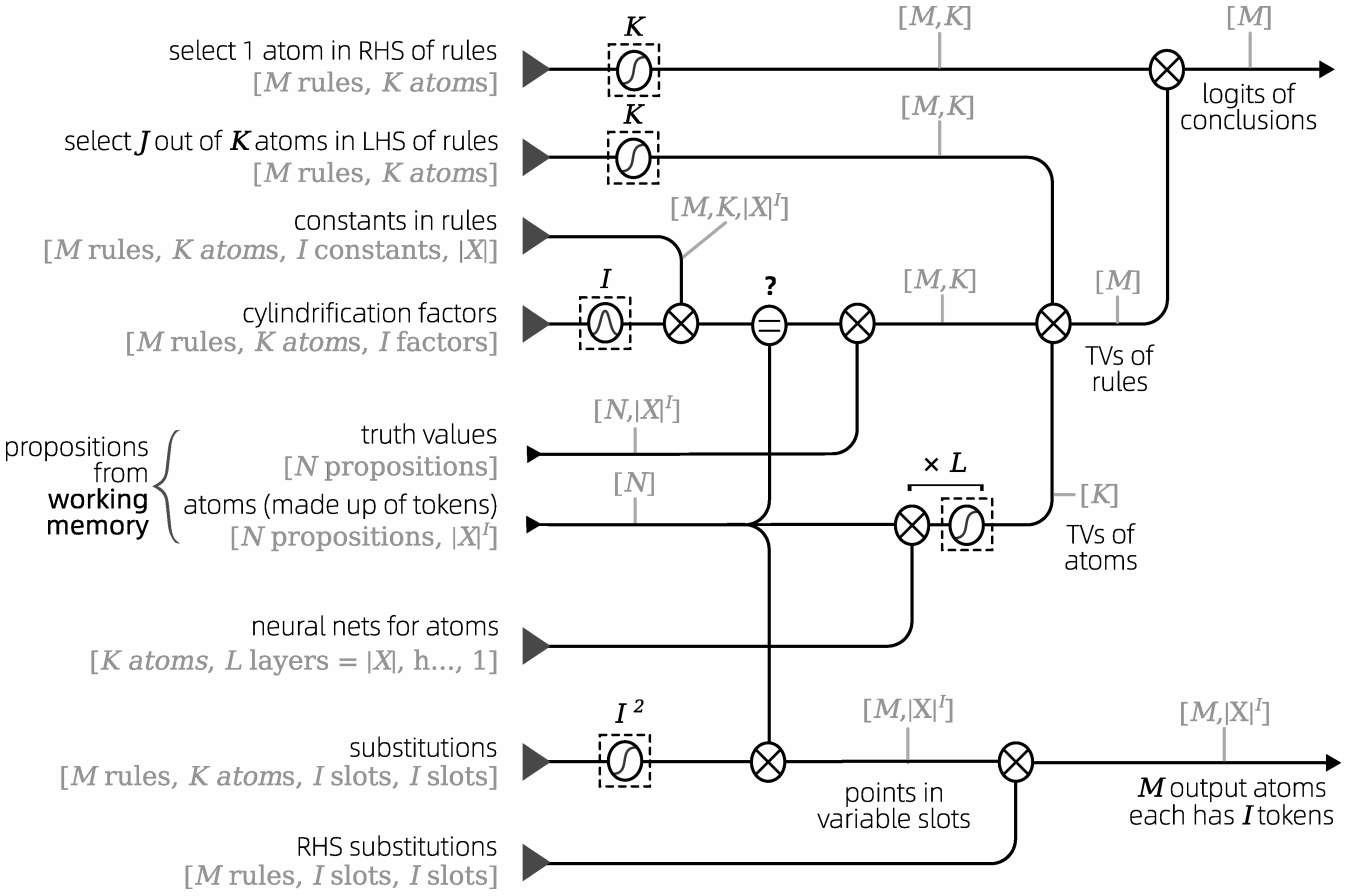


Fig. 4. String diagram of the Logic Transformer. The ▶ indicates **internal data** supplied by the Transformer, similar to the Q,K,V matrices in traditional Transformers. These are modified through "learning" or training. The 2 ▶'s are **inputs** to the Transformer, this is matched with the 2 **outputs** on the right of the diagram. Tags $[M, N, ...]$ indicate the **dimensions** of data streams.