From Transformers to AGI

YKY

February 24, 2023

Abstract

We're just applying ideas from classical logic-based AI to the new perspective of Transformers.

Language models and how they are trained

As of 2022, the most "intelligent" AI systems are **language models** such as BERT, GPT-3, ... and their variants.

These systems are trained to understand natural language, using a revolutionary technique by **masking out** words in a text corpus and asking the AI to **predict** them. For example:

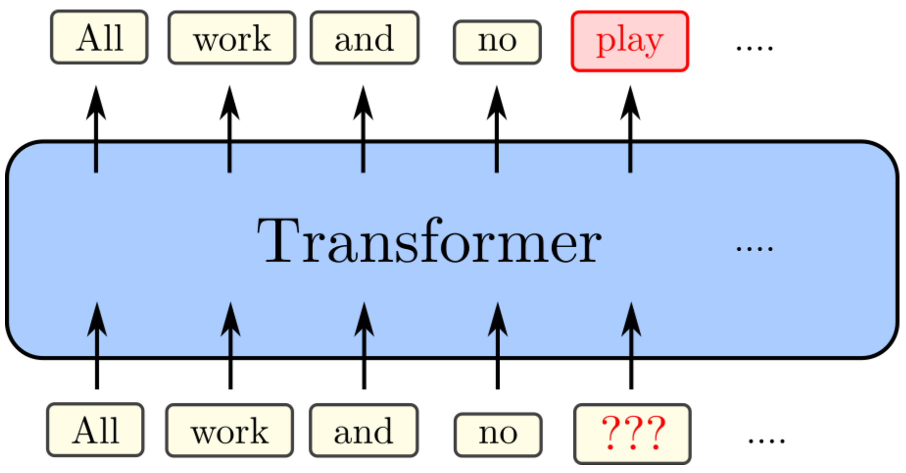
$$\begin{array}{c} \boxed{\text{All}}\ \boxed{\text{work}}\ \boxed{\text{and}}\ \boxed{\text{no}}\ \boxed{\text{play}}\ \boxed{\text{makes}}\ \boxed{\text{Jack}}\ \boxed{\text{a}}\ \boxed{\text{dull}}\ \boxed{\text{boy}} \\ \Downarrow \\ \boxed{\text{All}}\ \boxed{\text{work}}\ \boxed{\text{and}}\ \boxed{\text{no}}\ \boxed{???}\ \boxed{\text{makes}}\ \boxed{\text{Jack}}\ \boxed{\text{a}}\ \boxed{\text{dull}}\ \boxed{\text{boy}} \end{array} \tag{1}$$

When a neural network is **forced** to make such predictions, it will gradually start to acquire knowledge similar to human's. This knowledge is stored in the (massive number of) **weights** in the neural network.

Using the masked-word trick, there is no longer the need to prepare **annotated** data to train the AI, solving one of the biggest bottlenecks in AI development.

Transformers – the most advanced AI component

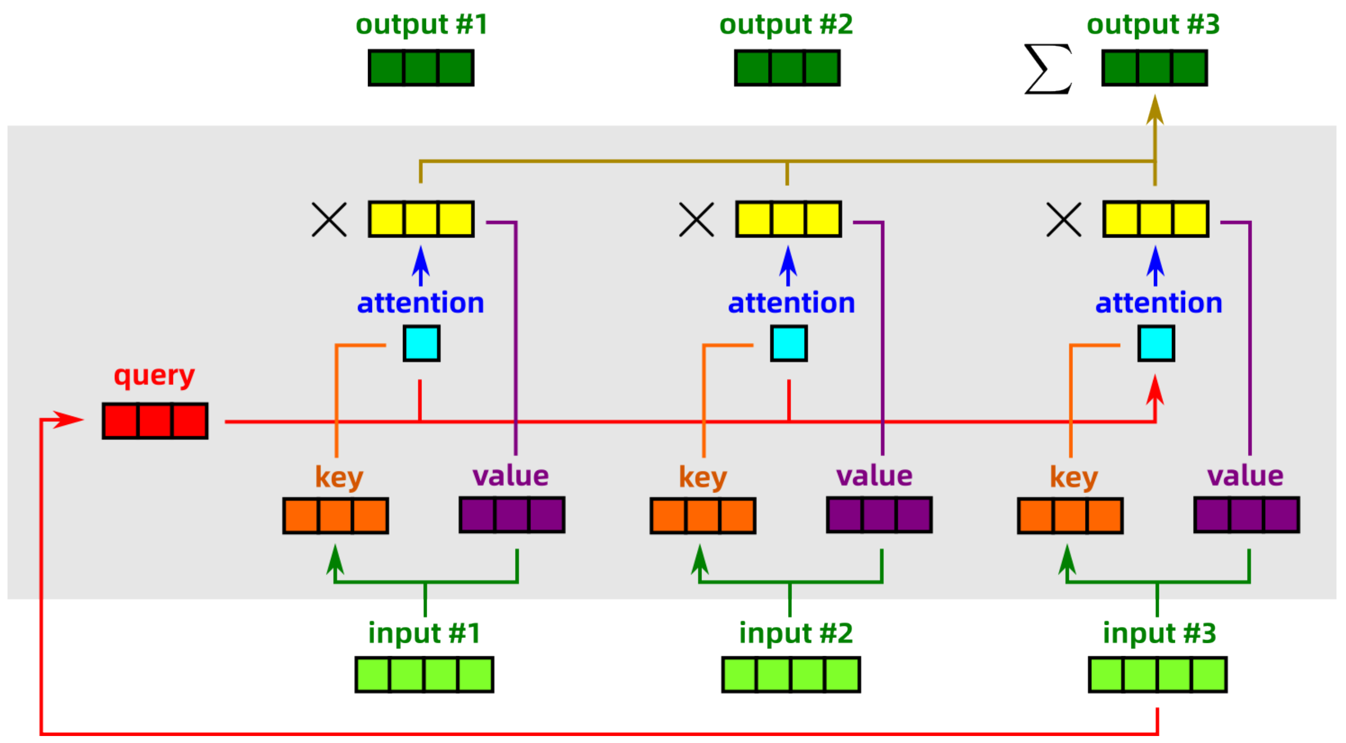
The above language models are based on a key component known as **Transformers**, invented by a research team in Google:



(2)

In principle, one only needs a "fully-connected" neural network and train it to predict the masked words. But such a network without **additional structure** is too inefficient to learn the desired knowledge. One of the key ideas in machine learning is to give a machine some "**bias**" to make it learn faster. An example of bias is to "cut" some connections in a fully-connected neural network, so that it appears as if partitioned into blocks.

The Transformer has an internal structure known as the **Attention** mechanism. It looks like this:



(3)

This is not widely accepted, but my hypothesis is that the Transformer performs logic reasoning.

Logic as a form of rewriting

The following are examples of **logic rules**:

"All humans are mortal":

$$\forall x.\ \text{human}(x) \Rightarrow \text{mortal}(x) \tag{4}$$

"The father's father is the grand-father":

$$\forall x, y, z.\ \text{father}(x, y) \wedge \text{father}(y, z) \Rightarrow \text{grand-father}(x, z) \tag{5}$$

The symbol $\forall$ means "for all", $\wedge$ means "and", $\Rightarrow$ means "imply".

In order for the logic formulas to work, variables such as x, y, z need to be **copied** from the left-hand-side **premise** to the right-hand-side **conclusion**. For example:

$$\begin{aligned} \text{human}(\text{Socrates}) &\Rightarrow \text{mortal}(\text{Socrates}) \\ \text{human}(\text{Plato}) &\Rightarrow \text{mortal}(\text{Plato}) \end{aligned} \tag{6}$$

These are "patterns" of a **rewriting system**. My hypothesis is that the Transformer is a kind of rewriting system. If we write the logic $\Rightarrow$ vertically as $\Uparrow$, it will look even more like the Transformer:

$$\begin{array}{c} \text{mortal}(x) \\ \Uparrow \\ \text{human}(x) \end{array} \tag{7}$$

A more complicated example:

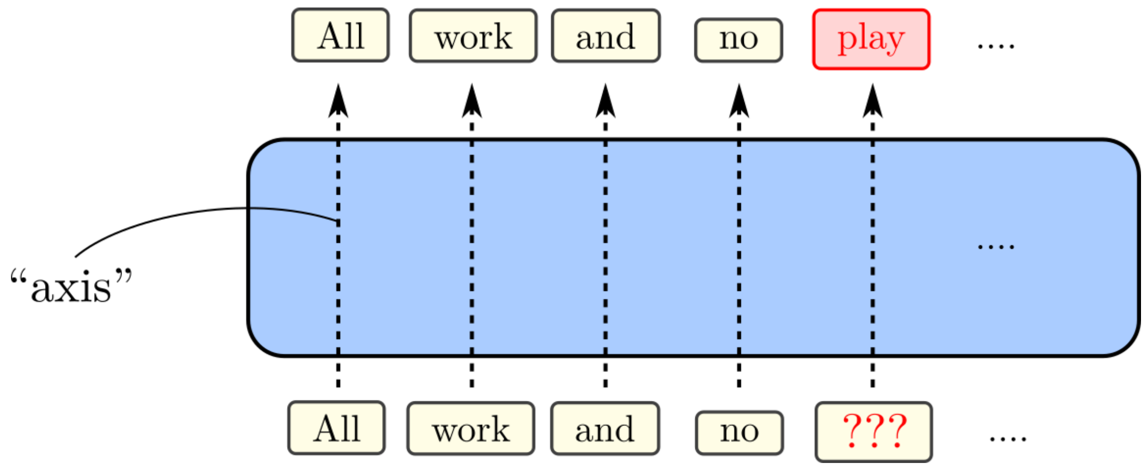
$$\begin{array}{c} \text{grand-father}(x, z) \\ \Uparrow \\ \text{father}(x, y) \wedge \text{father}(y, z) \end{array} \tag{8}$$

Notice that the bottom link $y - y$ signifies a requirement of **pattern matching**: the two y's must be substituted by the **same** object, otherwise the pattern does not match and the rule will not be applied.

How the Transformer performs rewriting

The Transformer's rewriting style is similar to the logical syntax above, but they differ in some details.

首先，在 Transformer 里，每个 token 轮流担任一次 Query 的角色。一个 token 担任 Query 做一次 Self-Attention，我们可以看成是一个垂直的**轴心** (axis). 每个轴心输出一个新的 token:



(9)

一个轴心可以看成是一条 **逻辑 rule**，但有如下差异：
一个 axis 的输入是整个序列的 tokens（其中一个是 Query），输出是一个 token:

$$\text{token}_1 \cdot \text{token}_2 \cdot \ldots. \to \text{token}_X \tag{10}$$

然而，逻辑 rule 的结构是这样的：

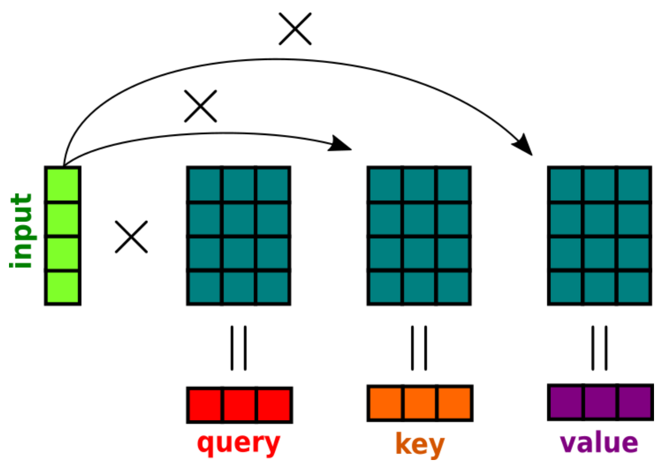
$$\text{proposition}_1 \wedge \text{proposition}_2 \wedge \ldots. \to \text{proposition}_X \tag{11}$$

而每个 proposition 由多个 symbols 组成，例如 John · loves · Mary.
可以这样看：Transformer 用一些 "micro-rules"，每次输出一个逻辑符号，从而组成输出的逻辑命题。

The Attention mechanism determines the **weights** by which the tokens combine, by summing up into a final token vector. Because of "softmax", the new token is effectively a **selection** of one token from amongst input tokens. In the logic view, a proposition is made up of multiple symbols, and the Attention mechanism **moves** tokens around (This ability is already present in the current Transformer).

The Transformer as a memory store

The Transformer uses **matrix multiplication** to calculate the Q, K, V values from the input. This involves three matrices which are like **look-up tables** or memory stores:



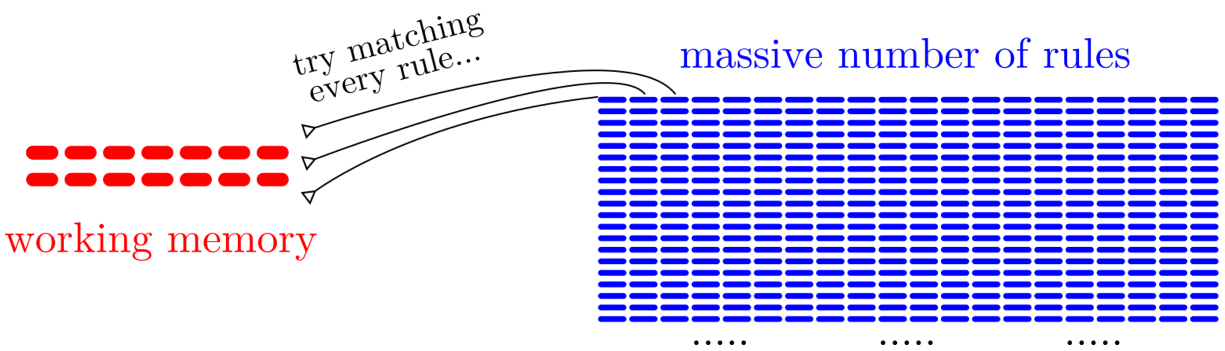
(12)

In other words, we can view these matrices as storing logic rules implicitly. The following idea helps explain how these rules are stored....

Rete algorithm for rules-matching

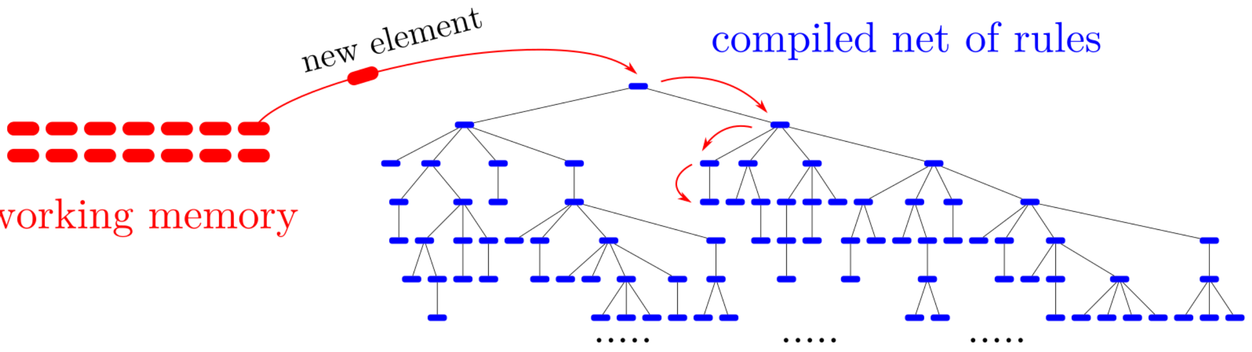
In classical logic-based AI, **rules-matching** is a very inefficient problem that is solved by the **Rete algorithm**.

Naively, one's first idea is to try to match *each and every* rule to Working Memory items, which of course is very inefficient:



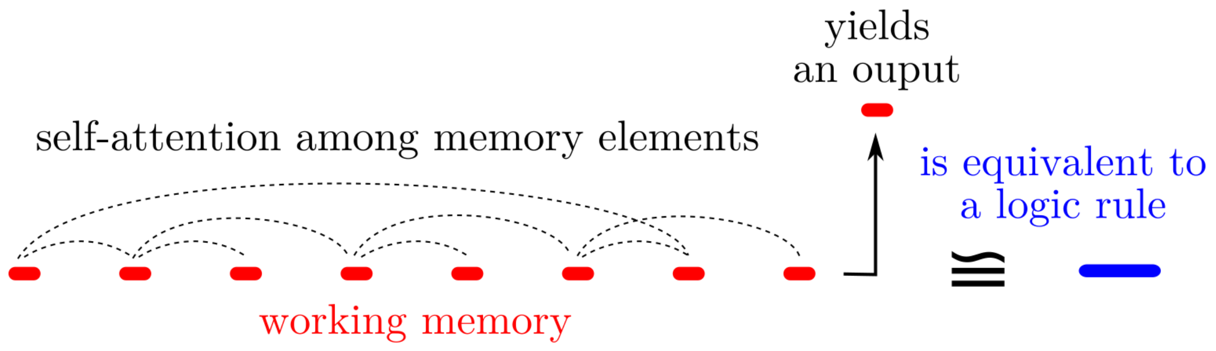
(13)

Instead, the **Rete algorithm** compiles the rules into a **classification tree**, so that each new Working Memory item is matched by "filtering" through the tree:



(14)

So how does the Transformer perform rules matching? It seems that rules are stored in the Q, K, V matrices. The Self-Attention mechanism is like a **message-passing** algorithm that yields a logic rule as its result:



(15)