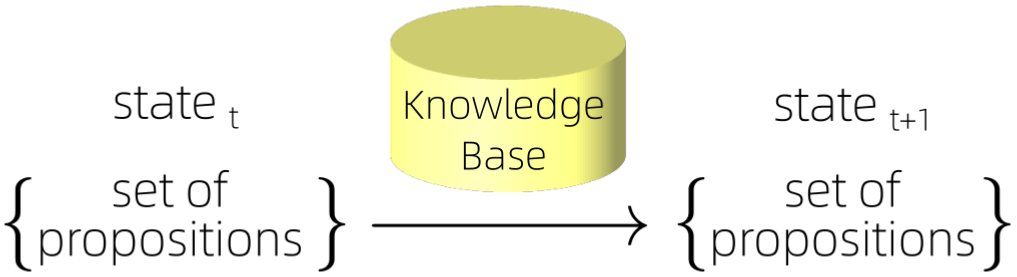
①

Comparison of Logic AI and Deep Learning

This is the most basic mode of operation of classical logic AI:



state t

set of propositions

Knowledge Base

state t+1

set of propositions

(1)

It actually contains two algorithms:

- **matching** (unification):
  Logical rules are conditional propositions involving variables,
  For example $\forall x.\mathit{human}(x) \Rightarrow \mathit{mortal}(x)$.
  Unification determines whether a rule can be applied to a logical proposition,
  For example: *human(Socrates)* can unify the left side of the formula.
  The result of Matching is to get a proposition that is instantiated (specialized, that is, does not contain variables).
- **forward- or backward-chaining** (resolution):
  Deduce new conclusions from known facts, or conversely, judge whether a given new conclusion is established.
  For example: *human (Socrates)* $\wedge$ *human(Socrates)* $\Rightarrow$ *mortal(Socrates)*
  It can be deduced: *mortal (Socrates)*.

The characteristics of deep learning are the

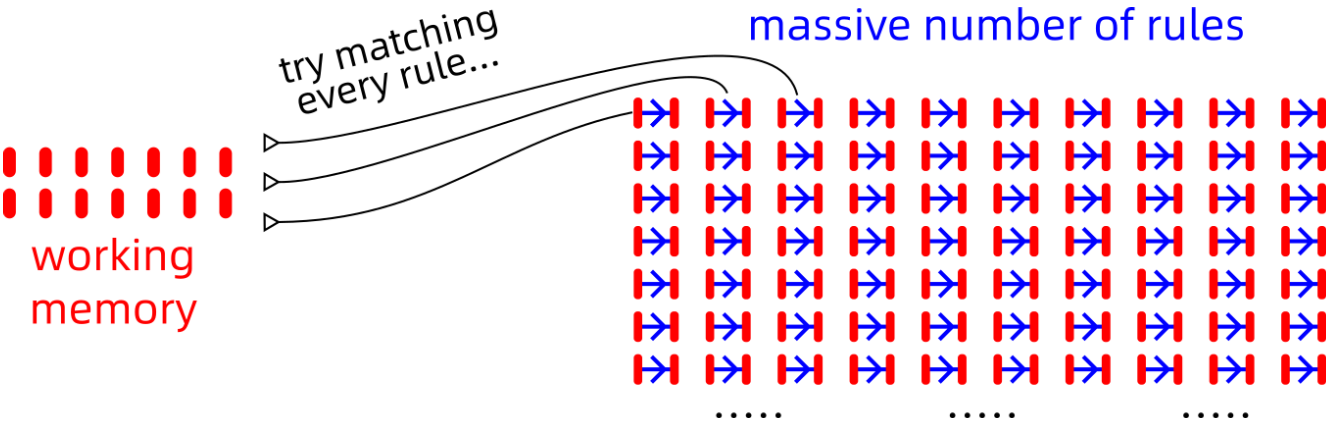
$$\text{state}_t \vdash \text{state}_{t+1} \tag{2}$$

The logical derivation process is all incorporated into a very complex nonlinear function (= deep neural network). After doing so, the above logical structure is "mingled" together so that it's hard to tell. But it is precisely because of this "hodgepodge" that the deep neural network compresses a set of complex combination algorithms into a small number of layer-by-layer parameters. It can do both learning and inference at the same time. This simple and crude method is actually very efficient, and it is not easy to surpass its speed!

We know (or speculate) that an intelligent system should have the structure of symbolic logic. Can this knowledge be used to constrain/accelerate deep neural networks? The answer seems to be possible. The current state-of-the-art CNN for vision and BERT for text both have an internal structure, **rather than fully-connected**, and this internal structure corresponds to the structure of the processed data. Therefore, we have reason to believe that the logical structure can be used to constrain the structure of the deep neural network to achieve acceleration.

Next, let's look at the structure of the logical system in detail:

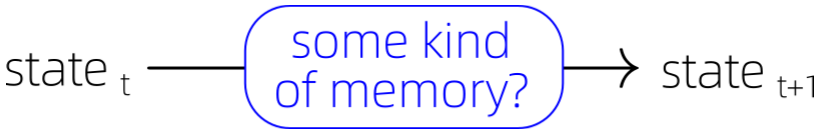
There are many rules in the Knowledge Base, and the system needs to match these rules one by one with the propositions in the system state (= working memory):



(3)

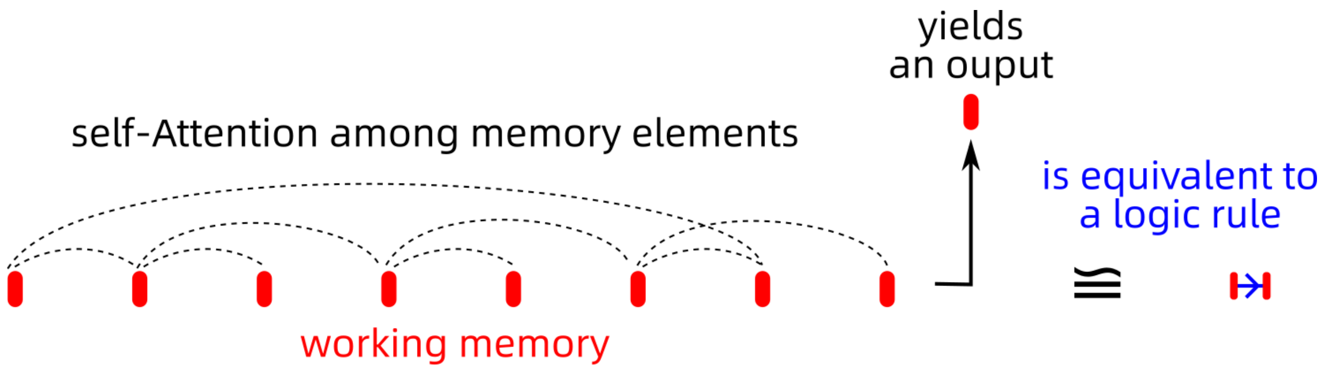
Successfully matched rules can derive new conclusions and add them to the state of working memory.

This complex operation is completely replaced by a neural network. Or more abstractly:



(4)

For Transformer, it is a kind of memory between the input elements **between** (this memory is stored in the Q, K, V matrices), and it **implicitly** achieves the role of rules:



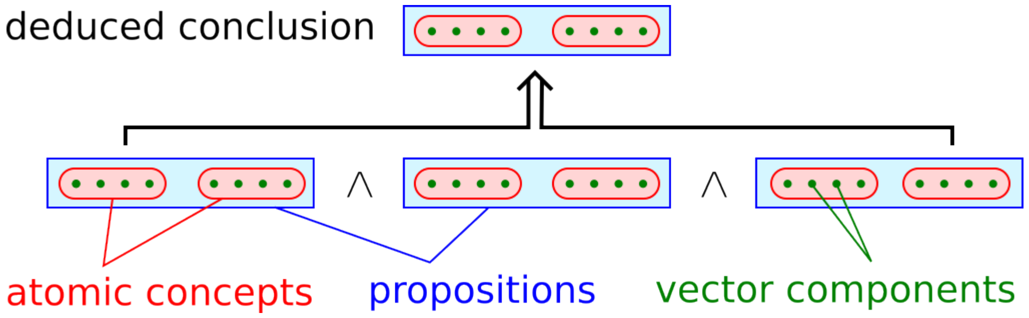
(5)

In other words, there is some sort of (distorted) structure of logical rules inside the Transformer. The natural question then is: Can more structures of logic/logic systems be discovered? That is, what kind of algebraic structure constraints can the formula (4) have? For this question, you can refer to the theory of category logic and the theory of classic logic-based AI systems.

We wish to sketch out the algebraic constraints that the formula (4) needs to have, but it is easier to describe in words for now:

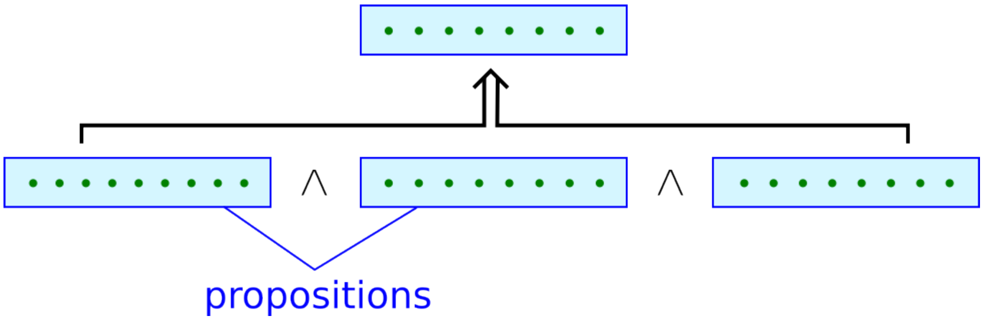
- The state is **granularized**, it is an element of a certain set, and the elements can be exchanged, that is, the equivariance of Transformer. (Note: Transformer has equivariance, but equivariance does not necessarily have to be realized by Transformer)
- **Deep structure**: For example, a multi-layer network, each layer is a composition of functions. Transformer also uses a deep structure.
- Logic includes granularity at the **proposition** level and at the **inside a proposition** level. The latter is the structure of **predicate** (predicate) logic, for example: *loves(John, Mary)*, it can also be simply regarded as the **product** between **algebraic elements** , for example: *John • loves • Mary*, the latter is also called "word". The details are not important.) The focus now is on how to apply the granular structure of these two layers to deep neural networks.
- Each step of logical derivation only produces **a** new conclusion (or its probability distribution), and then this new conclusion is added to the old state as an element of a proposition set, and the old state also needs to be textbfforget Some propositions, otherwise infinite memory is required. This is a bit different from Transformer outputting **column** tokens at a time (although we are not sure whether Transformer tokens correspond to propositions or predicate/atomic concepts).
- logical rule is usually only related to some premises, other premises are **irrelevant**, for example: *eyes beautiful* ∧ *nose beautiful* ∧ *mouth beautiful* ⇒ *handsome*, where *rich* or *poor* is irrelevant. Transformer's **softmax** structure also seems to be able to exclude the influence of some irrelevant tokens.
- (and possibly other structural features.....)

According to my theory, the ideal logical form is this (the number of various elements is purely indicative):



(6)

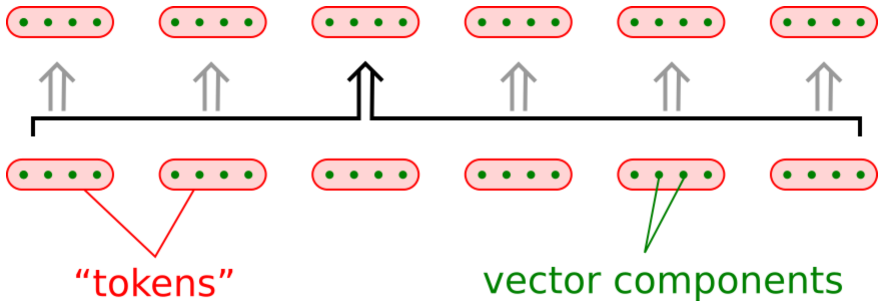
In contrast, the algebraic relation $p \wedge q = p \wedge q$ that we can currently write expresses only this structure:



(7)

Compared to figure (7), figure (6) adds ···· constraints, but How can constraints be represented algebraically?

And the way **Transformer** handles propositions and concepts is this:

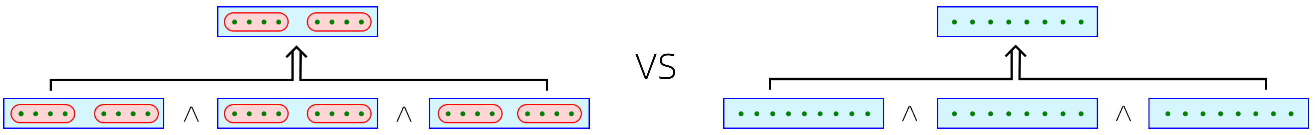


(8)

It does not have an explicit propositional structure, but uses a special token to represent the **end** of the sentence, and of course there are "tricks" such as positional encoding. So, Transformer is a rather *ad hoc* design, and we should be able to improve it.

Now we try to answer the the most important question: how to express in algebraic form "a proposition is composed of conceptual atoms"?

That is to say, what is the difference between the following two structures? How to express this difference algebraically?



(9)

It's like asking $0...9 \times 0...9$ vs. $00...99$ (basically no difference, they're isomorphic).

Similarly,

$$\{ \text{John, Mary} \} \times \{ \text{human, god, worm} \} \tag{10}$$

versus

$$\{ \text{John is human , Mary is human , ...} \} \tag{11}$$

These $2 \times 3 = 6$ propositions are also isomorphic. But the former is a combination of two different concepts, and its components can be quantified by ∀ or ∃; the latter is propositional logic, and those propositions are indivisible and cannot be separated and quantified.

But since $\boxed{\cdots}$ ∈ is a non-commutative free group (the group with the least structure), it does not have a symmetry formula like $a \cdot b = b \cdot a$.

After some analysis I got the following condition for "propositions are made of conceptual atoms":

Atomic Condition (AC). *Each proposition P_i is made up of K atoms:*

$$P_i = a_{i1} \cdot ... \cdot a_{iK} \tag{12}$$

where optionally some atoms can be **copied** *to other locations (with a non-linear transformation τ, if they are copied to the output layer) via:*

$$a_{ih} = a_{jk} \quad \textit{or} \quad a_{ih} = \tau(a_{jk}) \tag{13}$$

and the transformation τ has to accord with ∀ or ∃ as adjunctions to a substitution functor.

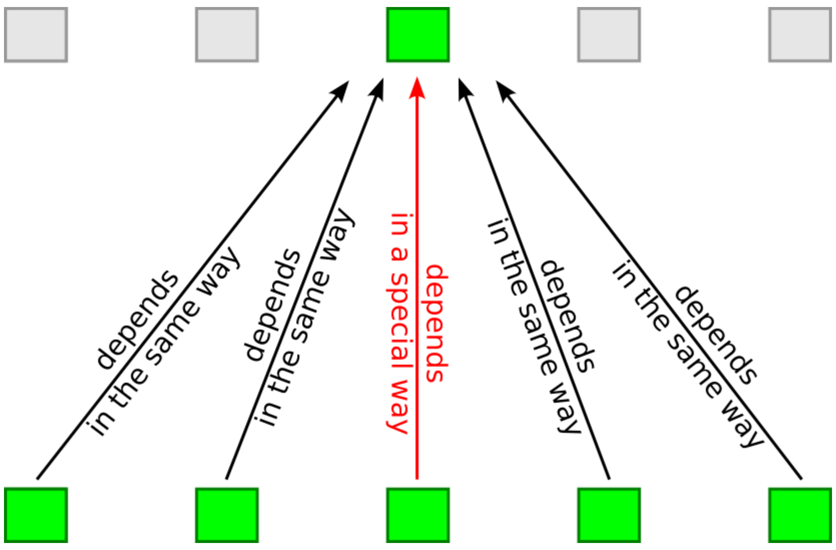
In fact, τ only needs to be a continuous function to meet the above conditions. So the focus of Atomic Condition lies in the two **equations** of (12) and (13), which are actually very simple. The category-theoretic description of ∀ and ∃ as adjoint functors is complicated, and we explain them in appendix Ⓐ.

So where does the "=" in the equation (13) come from? In fact, it is too obvious, it is the action of "syntactically moving" the variable in the logic rule:

$$\forall X, Y, Z.\ \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y) \wedge \text{father}(Y, Z) \tag{14}$$

It is these "movements" that constitute the structure of "propositions are composed of concepts".

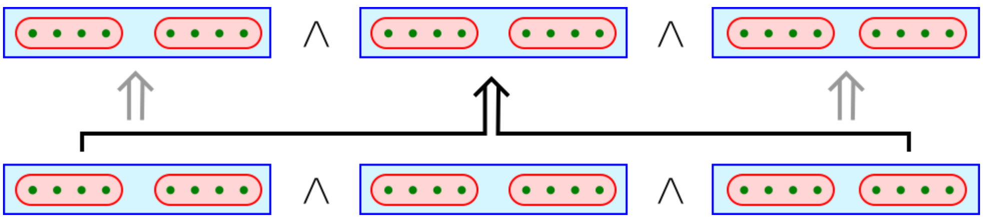
The essence of **Self-Attention** can be understood as follows (abstract attention structure):



(15)

This vertical "axis" structure (red) is repeated on each axis. So, when the input elements are **swapped**, the output is also swapped. This is why self-Attention can achieve the **equi-variant** effect.

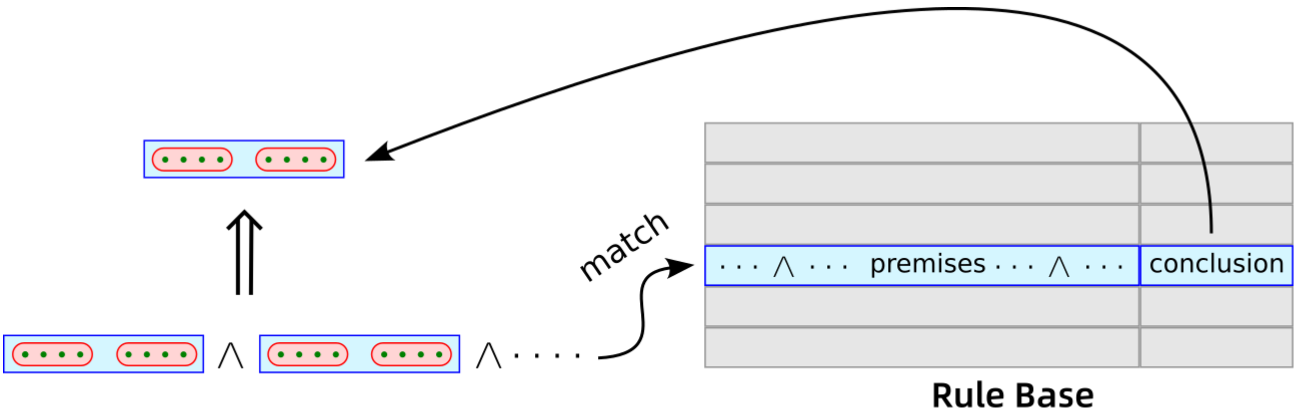
And we want to use a method similar to the above self-Attention to solve the problem of logical structure:



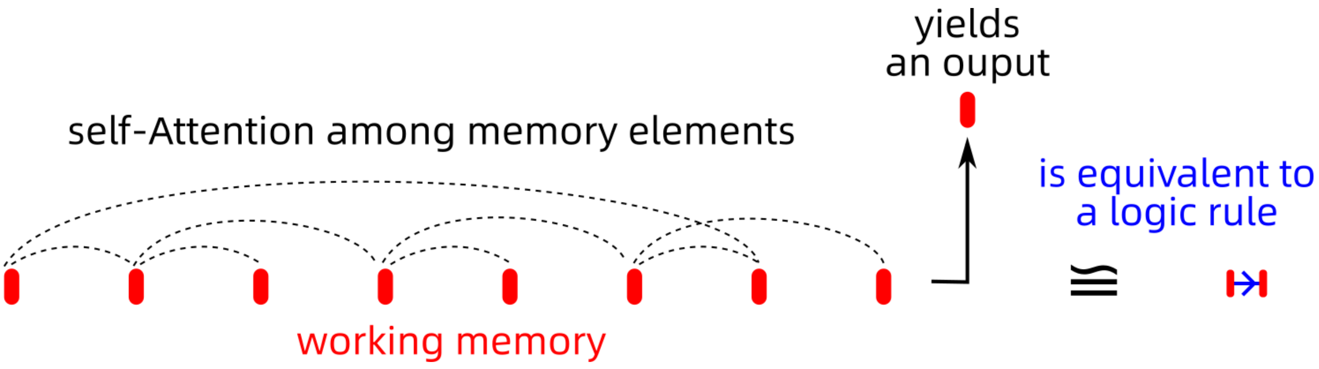
(16)

Here is a very important point[1]: The predecessor of self-Attention is **content-addressable memory** from Graves *et al.*'s "Neural Turing Machine". We have good reason to regard it as a kind of **memory** .

We want to compare two approaches. The former is a simple and direct classic rule base structure, and the latter uses self-Attention instead of rule base:



(17)



(5)

Everyone should feel that Transformer is a very "twisted" way of dealing with rules matching. This correspondence is not obvious, so that it is difficult for us to tell what the rules on the Transformer side look like. However, I think the designers of Transformer may be somewhat aware of its similarity to rule-based systems. In particular, look at the following logical rule:

$$\forall X, Y, Z.\ \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y) \wedge \text{father}(Y, Z) \tag{14}$$

The premise of this rule has two conditions, the variable Y that appears twice must be equal (red), and the matching is considered successful. And this kind of **comparison** operation inside the premise of the rule is exactly what self-Attention can do conveniently. But self-Attention also ignores the symmetry of $A \wedge B$, and there may be room for improvement.

[1] Thank you "Ziyu" for telling me this important information.

I think the key questions to be answered right now are:

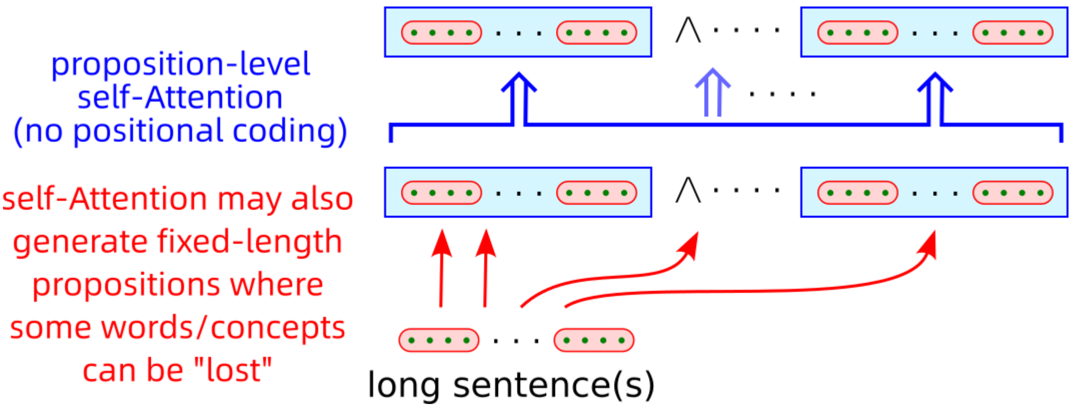
- According to e.g. Qian Liu's paper [1], Transformer can't do some logical and grammatical operations, where is the problem? It doesn't seem like Transformer can't learn that syntax at all, but that it can't do it purely from prompts. But does prompt actually have a deeper meaning, or is it just a hack? We didn't explicitly "tell" the Transformer what it should do, so is it a real shortcoming that it can't do it? I find it difficult to judge, and the research direction of prompt is shrouded in fog.
- Now consider the naïve learning algorithm for the graph (17) that is the rule base. This algorithm is of course very slow, since two sets (working memory and rule head) similarity. Assuming that the size of the two sets is fixed at N, then $N \times N$ times of dot products are needed, and this is just a comparison of a rule. All rules need to be added with softmax. When the rule base When it is very large, this algorithm seems impractical.
- figure (17) may also have the problem of the old logic rule learning algorithm, that is "plateau problem". For example, write append in Prolog language function:

  ```
  append(X,Y,Z) :-
          list(X), head(X,X1), tail(X,X2), append(X2,Y,W), cons(X1,W,Z).
  ```

  This rule has 5 premises. When the rule is learned, the premise is added one by one, but the "score value" of the rule is always zero, until the last premise is added, the score suddenly rises to 100%. For machine learning, this situation is Terrible. And the Transformer twists the rules together. Will this approach help avoid being trapped in the local minima?
- May have an algorithm between Transformer and naïve rule base, it has a stronger logical structure than Transformer, but uses more similar self-Attention's matrix operation to speed up?

[1] Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. *Compositional Generalization by Learning Analytical Expressions.* Advances in Neural Information Processing Systems 33 (2020).

A more feasible architecture that I think of now:

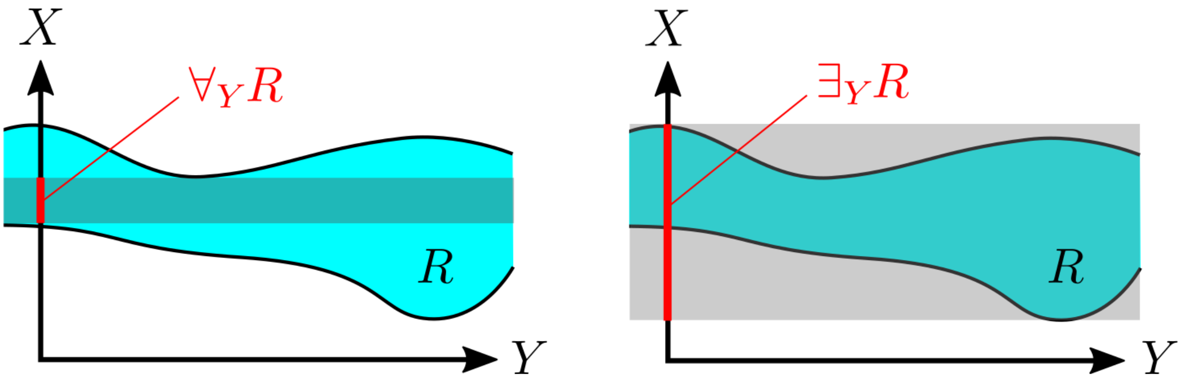


(18)

- In the blue structure, each proposition shall consist of fixed # of concept atoms.
- But the abstract self-Attention structure can also allow **variable length** propositions, as long as the **dot product** (similarity) between the propositions can be calculated, and the fixed-length **matrix** memory.
- red structure: how to generate propositions of varying number and length from sentences of indefinite length? This is the sequence to sequence-of-sequences problem discussed before. This can also be solved with "lossy" self-Attention.

□□□□□□□□□□□□□□□□□□□□□□□□□ ∀ □ ∃ □□□□□

□□□□□□□□□ R□□□□Y□X□□X,Y □□□□□□□□□□□□□□□□□□□□□□ □□ □□□ □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□ □□□□ □□□□□□□□□□□□□□□□□□□□ □□□□ R □□□□ □□ □ X □□□□□□□□□ $\forall_Y$ □ $\exists_Y$ □□□□ $\forall_Y$ □□ □□□□□□□□□□□□□□ X□$\exists_Y$ □□□□□□□□□□□□□□□□□ X:



(19)

□□□□□□ Lawvere □□□∀ □ ∃ □□□□□□ weakening functor □□□□□□□□□□□□□ weakening □□□□□□□□□ X □□□□□□ □□□□□ X,Y □□□□□□□□□□□□□□ Y □□□□□□ dummy variable:

$$\begin{tikzcd}[column sep=huge] (X) \arrow[r, "\text{weakening}"] & (X,Y) \arrow[l, bend right, "\forall_Y"'] \arrow[l, bend left, "\exists_Y"] \end{tikzcd} \tag{20}$$

□□ Love(Y,X) □□□□□□□□□□□□□□□□□□ $\forall Y.$ Love(Y,X) □□□□□□ Y □□□□□□□□□□□□ ∀ □□□□

□□ □□ (adjoint) □□□□□□ □□□□□□□□□□□□□□□□ □□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ morphism, □□□□ Set □□□□□□□□□□ set inclusion.

□□□□□ □□□□□□□□□□□□ weakening □□□□ ∀ □ ∃ □□□□□□

Lawvere’s work made the ∀ and ∃ quantifiers more general: the “simple” weakening functor is based on the Cartesian product $X \times Y$, but Lawvere expanded it to arbitrary substitution functors. (I don’t know of examples of this, so I’m not sure what advantages this could bring to our application.)

In categorical logic there are **Beck-Chevalley** conditions and **Frobenius** conditions, perhaps the symmetry we need? But after a closer look, I found that the problem still cannot be solved... For completeness, I will describe it, and you can skip it if you are not interested.

Consider first the easier-to-understand **Frobenius** condition. Logically, it is equivalent to saying:

$$\exists x.[\phi \wedge \psi(x)] \equiv \phi \wedge \exists x.\psi(x). \tag{21}$$

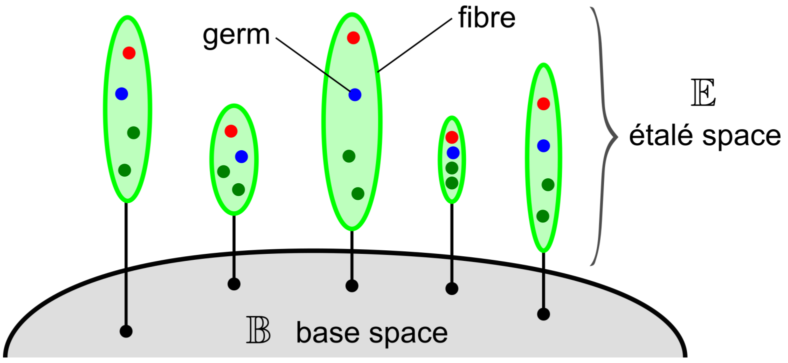
Since classical logic AI generally uses ∀ and ignores ∃, I rewrite the above formula as:

$$\forall x.[\phi \vee \psi(x)] \equiv \phi \vee \forall x.\psi(x). \tag{22}$$

But the problem is, the left and right sides of the formula (22), the corresponding neural network (6) is the same (no difference). In other words, this difference may be too subtle, and it does not affect the neural network we actually implement.

As said before, predicate logic leads to fibration or indexing constructs. The Beck-Chevalley and Frobenius conditions basically say that the fiber structure is "preserved by re-indexing functors".

Here is a diagram of the fibration structure:



(23)

This whole structure is called bundle, and sheaf is bundle plus some special topology.

fibred product of A and B over I can be defined on top of the two bundles (A, f) and (B, g), denoted as $A \times_I B$:

$$\begin{tikzcd}
A \times B \arrow[r, "q"] \arrow[d, "p"'] \arrow[dr, "h"] & B \arrow[d, "g"] \\
A \arrow[r, "f"'] & I
\end{tikzcd} \qquad (24)$$

where $h = f \circ p = g \circ q$. This is also a pullback.

The Beck-Chevalley condition says that the following image commute:

$$\begin{tikzcd}
K \times J \arrow[r, "u \times id"] \arrow[d, "\pi"'] & I \times J \arrow[d, "\pi"] \\
K \arrow[r, "u"'] & I
\end{tikzcd} \qquad (25)$$

where π is the projection representing the quantifier $\forall$ or $\exists$, which are the accompanying maps of the weakening map π^*.

The Beck-Chevalley condition is not entirely hollow; it may not hold. There is a counter-example from Pitts: Consider $X \times Y$, where $X = Y = \mathbb{N} \cup \{\infty\}$ is the natural number plus ∞ as top element; but Y uses discrete order, that is, all orders are = order. A is the relationship on $X \times Y$: $A = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid x \leq y\}$. Then $\exists y.(x, y) \in A$ will be the entire set of X. If we consider the DCPO category, we require the fibration of Scott-closed subsets (ordered by inclusion) over DCPO. The condition for Scott closure of $\exists y.A$ is that it is a lower set closed under directed joins; and this Scott closure condition seems to be violated, thus causing the graph (25) not to commute. (I do not understand the details of Scott closure)

First express the self-Attention structure in a functional way:

$$\boxed{\text{output proposition } O_i \text{ is composed of atoms } b_i} \quad O_i = [b_1 ... b_K] \tag{26}$$

$$\boxed{\text{input proposition } P_i \text{ is composed of atoms } a_i} \quad P_i = [a_1 ... a_K] \tag{27}$$

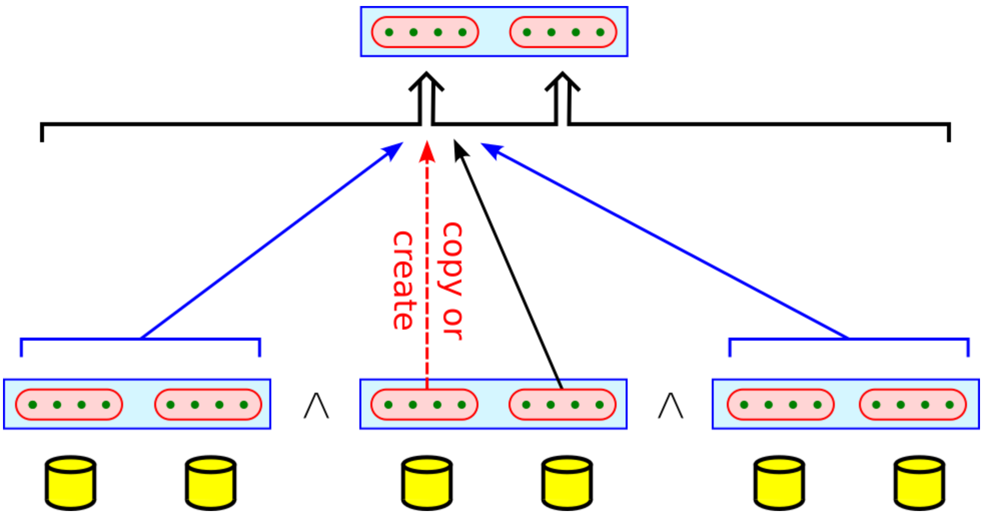
$$\boxed{\text{Self-Attention}} \quad O_i = \alpha(P_i \,;\, P_1 ... \hat{P}_i ... P_N) \tag{28}$$

$P_1 ... \hat{P}_i ... P_N$ means $P_1 ... P_N$ except P_i.
$\alpha(P_i \,; ...)$ is the function structure of the graph (15), and it can also be understood as the self Attention with P_i as the query.
How to measure similarity between (P_i, Q_i)?

$$\arg\min_i \sum_i \min_j \langle P_i, Q_j \rangle \tag{29}$$

- First notice that the pivot structure is only useful at the propositional level, and its effect does not extend to the conceptual atomic level. However, since Transformer does not take full advantage of the exchange invariance, it becomes very efficient because it uses matrix multiplication, so from an efficiency point of view, there are also reasons to extend the axis structure to the atomic level.
- Another idea to try is: direct hard-code copying mechanism. How can this be done with Attention? It has been analyzed before, copy is not easy, because winner takes all.
- but purely using Hopfield network lacks depth. But it seems that in the RL scenario, breadth is also important.
  In fact, as long as there is an input/output functional relationship,
- is equivalent to a KB library with logic rules. The question is what method it uses to reach its conclusions.

Self-Attention already meets our requirements, but we want to improve it. There are two main ideas: one is a more direct copy mechanism, and the other is a more detailed function dependency.



(30)

Copy:

- Copy requires no special mechanism, the output is the input. But the question is how to combine with "create" operation.
- can of course use the familiar softmax: α copy $+ \beta$ create, where α, β are outputs of softmax.
- Another idea is content-addressable. Use table-lookup to find executable rules. But there is a problem with variable matching.

Create:

- What kind of function is needed?