

AGI architecture white paper 2018-2

甄景贤 (King-Yin Yan)

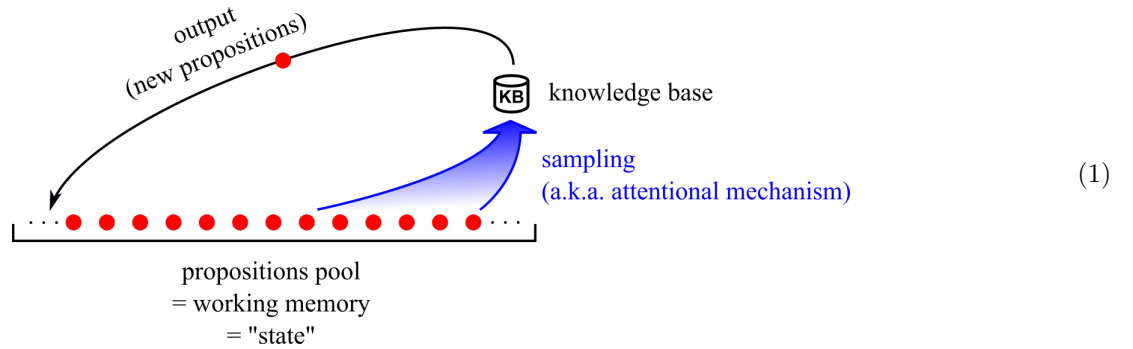
June 11, 2018

Abstract

My latest attempt to combine logic-based AI and deep learning. The main point is accepting the propositional structure of logic. Under this premise, this is the simplest architecture I can design. Perhaps other readers can improve upon it.


0 Basic architecture

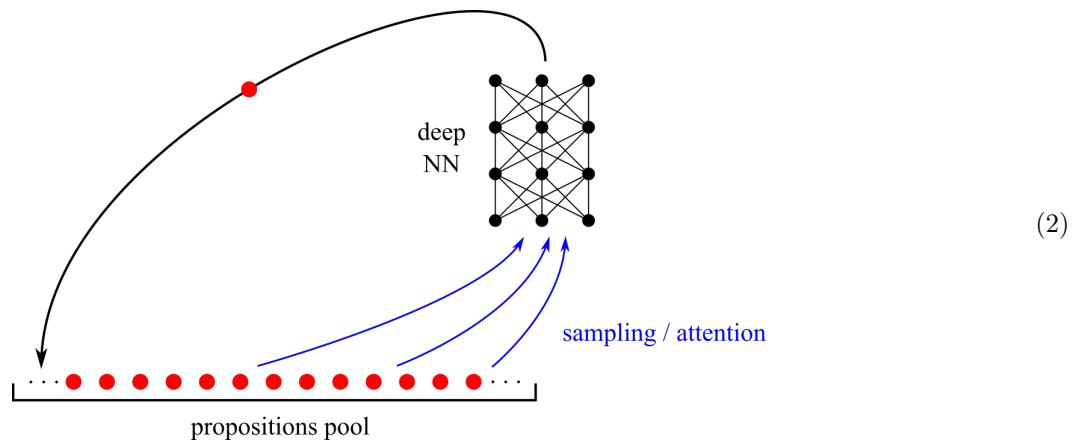
First, let's look at this basic architecture of classical logic-based AI:



- working memory is a set of propositions
- KB stores the totality of knowledge in the system
- The idea of attention is a current hot topic in deep learning. In our architecture, attention coincides with the same notion in cognitive science.

The theoretical background can be found in any classical AI textbook.

The architecture I propose is to use a deep neural network to replace the work of :



The Achilles' heel of classical logic-based AI is that learning is too slow, it uses **inductive logic learning**, the prominent researchers in this area include Stephen Muggleton, Luc de Raedt, Ivan Bratko, A. Srinivasan, and others (sorry that I am just recalling from memory). The field lacked significant breakthroughs, that led to the "winter" of logic-based AI.

Why are neural networks so powerful?

Deep learning is the most powerful machine learning technique nowadays. Its strength lies in the fact that a deep neural network can realize a huge family of functions using a relatively small number of parameters. Assume that the network maps from $\mathbb{R}^n \rightarrow \mathbb{R}^n$, or the binarized form $\{0, 1\}^n \rightarrow \{0, 1\}^n$, which are the vertices of the n -dimension **hypercube**. The number of parameters in the neural network is:

$$L n^2$$

where $L = \#(\text{layers})$, assuming every layer is a square matrix, fully connected. Whereas, the number of functions that exists between the input and output spaces is:

$$2^n \rightarrow 2^n = (2^n)^{2^n} \quad (3)$$

This is essentially “infinity” in computer science. The number of functions grows exponentially, but a deep neural network can handle them because, as the number of layers grows, the possible shapes of the network function also increases exponentially (the complexity of shapes can be measured by a “topological degree” such as the Leray-Schauder degree). Deep learning is the only learning machine with this property.

Now that we have a well-defined function, and we have the weapon, the problem’s solution should not be far away. Someone just needs to combine the two aspects. Here we propose one such attempt.

The propositional structure in logic

In this architecture, the deep NN inputs a set of propositions (encoded as vectors), and outputs some new propositions (the result of logical inference).

The special point of this architecture is that it respects the propositional structure of logic. In philosophical logic, the notion of the proposition is of central importance. Basically, a proposition is an entity that we can assign true or false. If we abandon even this notion, there wouldn’t be much left of classical logic.

The essential property of propositions is their **commutativity**:

$$A \wedge B = B \wedge A \quad (4)$$

which is why, in the propositions pool, we can arrange the order of propositions arbitrarily, and the order in which they are presented to the deep NN is unimportant. I have previously proposed a kind of **symmetric NN** of the functional form $F(a, b) = F(b, a)$, but I have abandoned this idea for the reason explained in the next section....

1 Rule matching

“Finding rules with facts”

In classical logic-based AI there used to be a critical problem: given some facts, how to find the applicable rules in the $\boxed{\text{KB}}$? This is solved in the 1970s by the Rete algorithm (*rete* in Latin means something like “threads”). The main idea in Rete is: searching rules by facts, instead of searching facts by rules. The reason is simple: the number of facts in working memory is relatively small, but the number of rules in $\boxed{\text{KB}}$ is enormous.

Later, Rete is employed in the SOAR architecture, a famous cognitive architecture cum rule engine, developed in Carnegie-Mellon University. Of course, SOAR did not achieve AGI, as I explained earlier, the Achilles’ heel is in learning.

In our architecture, learning is delegated to the deep NN, but rule matching is still a very time-consuming operation. Even if one believes in the “unreasonable effectiveness” of recurrent neural networks, there is a strong reason to implement rule matching explicitly, instead of letting the RNN learn it from scratch, which could still be too slow.

The solution I propose, is along the same thinking as Rete: searching rules by facts. For example, we may have the following rule:

$$A \wedge B \wedge C \rightarrow D \quad (5)$$

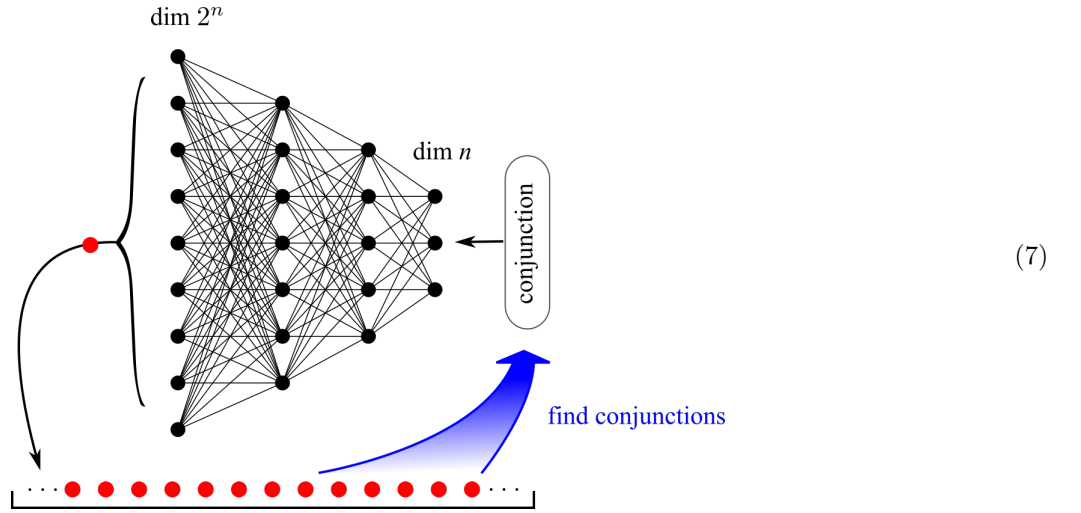
where A, B, C, D are propositions. The trick is to “invert” the conjunction $A \wedge B \wedge C$ by representing it as a **point**, whereas the propositions A, B, C are represented by some **regions**. If the region A includes \bullet , that means A participates in this conjunction:



The intention is: each proposition is a region which can also be regarded as **characteristic function**: it inputs a certain **point** location, and outputs true or false depending if that point belongs to the region. In other words it is like an “agent”, input a conjunction, it outputs if it participates in this particular conjunction. In other words, it is “searching rules by facts”.

Actual implementation is as follows: As we need many points to represent conjunctions, we could take the 2^n vertices of the hypercube. And as each rule of the form (5) is associated with one conjunction, the number of vertices of the hypercube is the total number of rules in $\boxed{\text{KB}}$. In other words, the job of the deep NN is just to learn, store and **apply** the rules that have been found. The rule matching task is separated out.

Now the architecture becomes like this (but it still has a problem):



The NN receives its input as a single vertex of the hypercube, which represents a conjunction. In other words, the input is a binary vector of dimension n . Output is a proposition; and from our setup, a proposition is a **subset** of vertices of the hypercube. That is to say, output $\in 2^{2^n}$. If this output is also represented as a binary vector, its length would be 2^n . Clearly, when $n > O(10)$ this becomes rather infeasible. For instance, you may believe that human intelligence requires a $\boxed{\text{KB}}$ of size $\approx 2^{100}$ to 2^{1000} or even more, but if it is $2^{20} = 1048576$ then it might be too small. Therefore, the NN’s output representation of the proposition needs to be **compressed**.

Compression means to encode the vector of length 2^n by a shorter vector. This encoding could be an approximating function $f : 2^n \rightarrow \mathbf{2}$ (input is a certain vertex = conjunction, output = yes / no)¹, for example:

- NN outputs a series of weights, which constructs a **small neural network** $F : 2^n \rightarrow \mathbf{2}$
In other words, each proposition is a micro-neural network.
- use multi-variate discrete Fourier transform truncate low-energy terms

This compression step is rather troublesome. The first method is essentially to “use big NN to output small NNs”, whose workability remains to be proven. The need for compression comes from the too-long length of the proposition representation. One idea is to transfer the complexity of the proposition representation to the conjunction representation. But is this useful? We’d just be transferring the complexity from the output representation to the input representation of the same NN. This is my thinking so far.²

¹In practice, a conjunction may have a variable number k of conjuncts, so we may need to use $2^n \rightarrow \mathbb{R}$ or $2^n \rightarrow [0, 1]$. Also, we need to avoid counting a proposition twice in the proposition pool.

²About this compression problem, I have previously encountered it in a similar form, while designing a different AGI architecture. Then, my goal was to achieve $X^X \cong X$ of a Cartesian closed category. My solution is to let the NN write to its own weights. But the number of weights in an NN is clearly \gg the dimension of the output vector, so we needed compression in that situation as well. My feeling is that the need for compression is an indication of the AGI bottleneck.

Stochastic local search for matching

One remaining detail: according to the above design, given a set of propositions, how do we find the satisfying conjunctions? An inspiration comes from the GSAT and WalkSAT algorithms for propositional satisfiability, the famous NP-complete problem. SAT is very similar to our problem, perhaps even isomorphic (?)

The G in GSAT means greedy, whereas WalkSAT refers to random walk. It has been proven in practice that greedy + stochastic search are highly effective heuristics for solving SAT.

As mentioned earlier, each proposition is a micro-function $f : \text{vertex} \rightarrow \text{true} / \text{false}$. We can **clamp** the output value to true, and use **backward propagation** to obtain some instances of satisfying vertices (these are not unique). Use a list to store these vertices.

And then perhaps we can use a **genetic algorithm** to evolve these vertices until we find a satisfied conjunction.

2 Learning

Learning is simply the application of **deep reinforcement learning**, via the **Bellman update**. Training requires rewards and punishment information from a teacher or training data set. This is a well-researched topic, for example there is an online course on DRL from UC Berkeley.

The **credit assignment** problem: rewards and punishment is not instantly received after a single inference step, but may appear after a long **inference chain**. This inference chain plays the same role as the **eligibility trace** in reinforcement learning. In other words, the entire chain needs to be rewarded or punished during learning (in other words, increase or decrease the weights that were **responsible** in the inference chain). All the weights in the NN can be re-normalized periodically.

A note about natural language

To generate natural language from internal representations, we only need additional logical inference. However, to learn language generation while learning the internal representation and knowledge at the same time might be too difficult, so we would advice against trying these as first experiments. However, even for humans, talking about things that one is not confident about, often results in stammering and other speech disturbances. This may be evidence that rewarding / punishing the inference chains for language and for real knowledge at the same time, may be “normal”



3 Conclusion

One thing we are not satisfied with is the need for compression of the proposition representation. Representing a proposition as a micro-NN, changing its weights may have very small and indirect effects, thus the vanishing gradient is a worry. But I am glad to see the goal of AGI getting more and more achievable.

Acknowledgements

Mixing neural networks with von Neumann-style architecture programming, while biologically inelegant, may be highly effective. I saw an example of this from Andrew Ng *et al*'s paper on recursive neural networks (not to be confused with recurrent neural networks).