AGI via "deep logic"

甄景贤 (King-Yin Yan)   Dmitri Tkatch   肖达 (Da Xiao)   Juan Carlos Kuri Pinto

General.Intelligence@Gmail.com

Abstract. An integration of deep learning and logic is proposed, based on the Curry-Howard isomorphism. Under this interpretation, it turns out that Google BERT, where current state-of-the-art language models are derived from, can be regarded as a special form of logic. Furthermore, this idea can be incorporated under the reinforcement-learning framework to provide a minimalist AGI architecture.

Keywords: cognitive architecture, reinforcement learning, deep learning, logic-based artificial intelligence

0   Summary

We propose an AGI architecture:

1. with **reinforcement learning** (RL) as top-level framework
   - State space = mental space
2. **Logic** structure is imposed on the **knowledge representation** (KR)
   - State transitions are given by logic rules = actions in RL
   - The logic state x is decomposable into **propositions** (§5.4)
3. The set of logic rules is approximated by a deep neural network
   - Just the most basic kind of feed-forward neural network (FFNN) is required
   - Logic conjunctions are **commutative**, so working-memory elements can be presented in any order (§5.5)
   - **Stochastic** actions are represented by **Gaussian kernels** (radial basis functions) (§5.6), thus partly avoiding the curse of dimensionality

The rest of this paper will explain these design features in detail.

1   Prior research

1.1   Cognitive architectures and reinforcement learning

AIXI.

Reinforcement learning.

1.2   Neuro-symbolic integration

Pascal Hitzler.

∂-ILP

Ron Sun.

GLOM.

2 Reinforcement-learning architecture

The comparison of RL with neuroscience helps to crack the brain code:

- What constitute the brain's **state**?
- What is the **state transition function**?
- What enables **learning** (in the state transtion function)?

3 The structure of logic

3.1 Curry-Howard isomorphism, dependent type theory

As the risk of being overly rudimentary, we would go over some basic ideas, that may help the readers who are not familiar with this area of mathematics.

The Curry-Howard isomorphism

Pair of pants, quantum logic, ...

3.2 Topos theory

The logic of sheaves is intuitionistic.

3.3 BERT as a logic
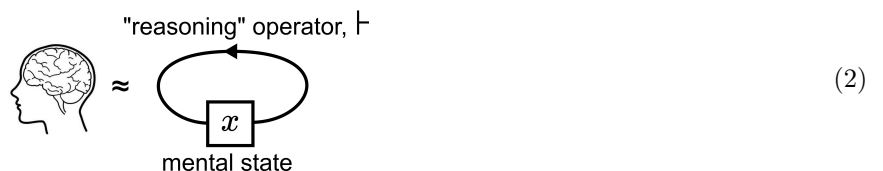
3.4 Neuroscience considerations

3.5 Experiments

4 Reinforcement-learning architecture

The **metaphor** in the title of this paper is that of RL controlling an autonomous agent to navigate the maze of "thoughts space", seeking the optimal path:

$$\tag{1}$$

The main idea is to regard "thinking" as a **dynamical system** operating on **mental states**:

$$\tag{2}$$

A mental state is a **set of propositions**, for example:

- I am in my room, writing a paper for AGI-2019.
- I am in the midst of writing the sentence, "I am in my room, ..."
- I am about to write a gerund phrase "writing a paper..."

Thinking is the process of **transitioning** from one mental state to another. As I am writing now, I use my mental states to keep track of where I am at within the sentence's syntax, so that I can construct my sentence grammatically.

4.1 Actions = cognitive state-transitions = "thinking"

Our system consists of two main algorithms:

1. Learning the transition function $\vdash$ or $\boldsymbol{F} : \boldsymbol{x} \mapsto \boldsymbol{x}'$. $\boldsymbol{F}$ represents the **knowledge** that constrains thinking. In other words, the learning of $\boldsymbol{F}$ is the learning of "static" knowledge.
2. Transitioning from $\boldsymbol{x}$ to $\boldsymbol{x}'$. This corresponds to "thinking" under the guidance of the static knowledge $\boldsymbol{F}$.

In our architecture, $\boldsymbol{F}$ can be implemented as a simple feed-forward neural network (where "deep" simply means "many layers"):

$$ \boldsymbol{x} \quad \boldsymbol{F} = \text{deep NN} \quad \boldsymbol{x}_{t+1} \tag{3} $$

Since a recurrent NN is Turing-complete, this can be viewed as a minimalist AGI. But its learning may be too slow without further **inductive bias** (*cf* the "no free lunch" theorem [**?**]) — so we will further modify $\boldsymbol{F}$ by imposing the logic structure of reasoning on it (§5.4 and §5.5).

In principle, every state is potentially **reachable** from every other state, if a logic rule exists between them. Now we use a deep FFNN to represent the set of all logic rules. This is a key efficiency-boosting step, because <u>deep neural networks allows to use a polynomial number of parameters to represent an exponential number of mappings</u>.

Note that parts of the state $\boldsymbol{x}$ would be reserved and directly connect to the **input** and **output** of the AGI system.

4.2 Comparison with AIXI [?]

AIXI's environmental setting is the same as ours, but its agent's internal model is a universal Turing machine, and the optimal action is chosen by maximizing potential rewards over all programs of the UTM. In our (minimal) model, the UTM is <u>constrained</u> to be a neural network, where the NN's **state** is analogous to the UTM's **tape**, and the optimal weights (program) are found via Bellman optimality.

5 Logic structure

5.1 Logic is needed as an inductive bias

We know that the transition function $\boldsymbol{F}$ is analogous to $\vdash$, the logic consequence or entailment operator. So we want to impose this logic structure on $\boldsymbol{F}$.

By logic structure we mean that $\boldsymbol{F}$ would act like a **knowledge base** [KB] containing a large number of logic **rules**, as in the setting of classical logic-based AI.

5.2 Geometry induced by logic rules

A logic rule is a conditional formula with variables. For example:

$$ \forall X \; \forall Y \; \forall Z. \quad \text{father}(X, Y) \wedge \text{father}(Y, Z) \Rightarrow \text{grandfather}(X, Z) \tag{4} $$

where the red lines show what I call "linkages" between different appearances of the same variables.

Quantification of logic variables, with their linkages, result in **cylindrical** and **diagonal** structures when the logic is interpreted *geometrically.* This is the reason why Tarski discovered the **cylindric algebra** structure of first-order

predicate logic [**?**] [**?**] [**?**] [**?**] [**?**]. Cylindrical shapes can arise from quantification as illustrated below:

$$\tag{5}$$
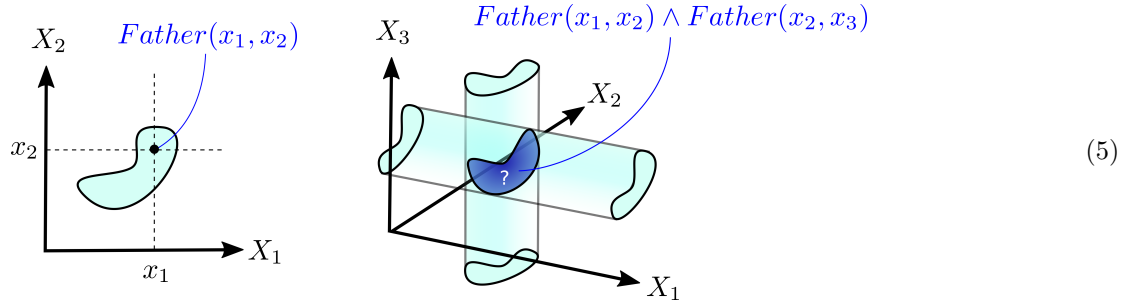
And "linkages" cause the graph of the $\vdash$ map to *pass through* diagonal lines such as follows:

$$\tag{6}$$

We are trying to use neural networks to approximate such functions (*ie*, these geometric shapes). One can visualize, as the shape of neural decision-boundaries approximate such diagonals, the matching of first-order objects gradually go from partial to fully-quantified $\forall$ and $\exists$. This may be even better than if we fix the logic to have exact quantifications, as quantified rules can be learned gradually. There is also *empirical* evidence that NNs can well-approximate logical maps, because the *symbolic* matching and substitution of logic variables is very similar to what occurs in *machine translation* between natural languages; In recent years, deep learning is fairly successful at the latter task.

5.3 Form of a logic rule

So what exactly is the logic structure? Recall that inside our RL model:

- state $\boldsymbol{x} \in \mathbb{X}$ = mental state = set of logic propositions $\mathsf{P}_i \in \mathbb{P}$
- environment = state space $\mathbb{X}$ = mental space
- actions $\boldsymbol{a} \in \mathbb{A}$ = logic rules

For our current prototype system, an action = a logic **rule** is of the form:

$$\overbrace{\mathsf{C}_1^1\,\mathsf{C}_2^1\,\mathsf{C}_3^1 \;\wedge\; \underbrace{\mathsf{C}_1^2\,\mathsf{C}_2^2\,\mathsf{C}_3^2}\;\wedge\; .... \;\wedge\; \mathsf{C}_1^k\,\mathsf{C}_2^k\,\mathsf{C}_3^k}^{\text{conjunction of } k \text{ literal propositions}} \;\Rightarrow\; \overbrace{\mathsf{C}_1^0\,\mathsf{C}_2^0\,\mathsf{C}_3^0}^{\text{conclusion}} \tag{7}$$

each literal made of m atomic concepts, $m = 3$ here

where a **concept** can be roughly understood as a **word vector** as in Word2Vec [**?**]. Each $\mathsf{C} \in \mathbb{R}^d$ where d is the dimension needed to represent a single word vector or concept.

We use a "free" neural network (*ie*, standard feed-forward NN) to approximate the set of *all* rules. The **input** of the NN would be the state vector $\boldsymbol{x}$:

$$\mathsf{C}_1^1\,\mathsf{C}_2^1\,\mathsf{C}_3^1 \;\wedge\; \mathsf{C}_1^2\,\mathsf{C}_2^2\,\mathsf{C}_3^2 \;\wedge\; .... \;\wedge\; \mathsf{C}_1^k\,\mathsf{C}_2^k\,\mathsf{C}_3^k. \tag{8}$$

We fix the number of conjunctions to be k, with the assumption that conjunctions of length $< k$ could be filled with "dummy" (always-true) propositions.

The **output** of the NN would be the conditional **probability** of an action:

$$P(\text{action} \mid \text{state}) := \pi(\,\mathsf{C}_1\,\mathsf{C}_2\,\mathsf{C}_3 \mid \boldsymbol{x}). \tag{9}$$

Note that we don't just want the action itself, we need the **probabilities** of firing these actions. The **Bellman update** of reinforcement learning should update the conditional probabilities over such actions (§**??**).

5.4 Structure of a logic-based AI system

Besides the intrinsic structure of a logic, the AI system has a structure in the sense that it must perform the following operations iteratively, in an endless loop:

$$(10)$$
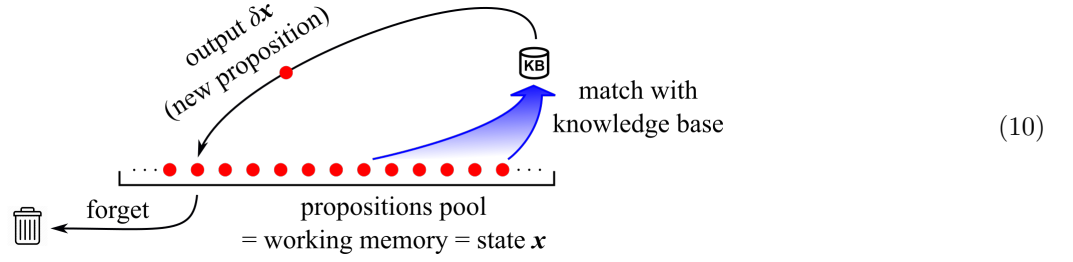
- **Matching** — the 🗄 of rules is matched against the current state $\boldsymbol{x}$, resulting in a (stochastically selected, *eg* based on ϵ-greedy) rule:

$$\boxed{\text{Match}} \quad (\boldsymbol{x} \overset{?}{=} \text{🗄}) \; : \mathbb{X} \to (\mathbb{X} \to \mathbb{P})$$
$$\boldsymbol{x} \mapsto \boldsymbol{r} \tag{11}$$

— In categorical logic, matching is seen as finding the **co-equalizer** of 2 terms which returns a **substitution** [?] [?] [?] [?]. The substitution is implicit in our formulation and would be *absorbed* into the neural network in our architecture.

— Matching should be performed over the entire **working memory** = the state $\boldsymbol{x}$ which contains k literals. This is combinatorially time-consuming. The celebrated ***Rete*** algorithm [?] turns the set of rules into a tree-like structure which is efficient for solving (11).

- **Rule application** — the rule is applied to the current state $\boldsymbol{x}$ to produce a new literal proposition $\delta\boldsymbol{x}$:

$$\boxed{\text{Apply}} \quad \boldsymbol{r} : \mathbb{X} \to \mathbb{P}$$
$$\boldsymbol{x} \mapsto \boldsymbol{r}(\boldsymbol{x}) = \delta\boldsymbol{x} \tag{12}$$

- **State update** — the state $\boldsymbol{x}$ is *destructively* updated where one literal $\mathsf{P}_j \in \boldsymbol{x}$ at the j-th position is **forgotten** (based on some measure of attention / interestingness) and over-written with $\delta\boldsymbol{x}$:

$$\boxed{\text{Update}} \quad \boldsymbol{x} = (\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_j, ..., \mathsf{P}_k) \mapsto (\mathsf{P}_1, \mathsf{P}_2, ..., \delta\boldsymbol{x}, ..., \mathsf{P}_k) \tag{13}$$

All these operations are represented by functions parametrized by some variables Θ and they must be made *differentiable* for gradient descent.

5.5 Commutativity of logic conjunctions

One basic characteristic of (classical) logic is that the conjuction $\wedge$ is **commutative**:

$$\mathsf{P} \wedge \mathsf{Q} \quad \Leftrightarrow \quad \mathsf{Q} \wedge \mathsf{P}. \tag{14}$$

This restriction may significantly reduce the size of the search space. If we use a neural network to model the deduction operator $\vdash: \mathbb{P}^k \to \mathbb{P}$, where $\mathbb{P}$ is the space of literal propositions, then this function should be **symmetric** in its input arguments.

I have considered a few solutions to this problem, including an algebraic trick to build "symmetric" neural networks (but it suffers from combinatorial inefficiency), and using Fourier transform to get a "spectral" representation of the state, which remained rather vague and did not materialize.

As of this writing [1] I have settled on the "carousel" solution: All the propositions in working memory will enter into a loop, and the reasoning operator acts on a hidden state $\boldsymbol{H} = \bullet$ and one proposition $\mathsf{P}_i = \bullet$ at a time:

$$(15)$$

---
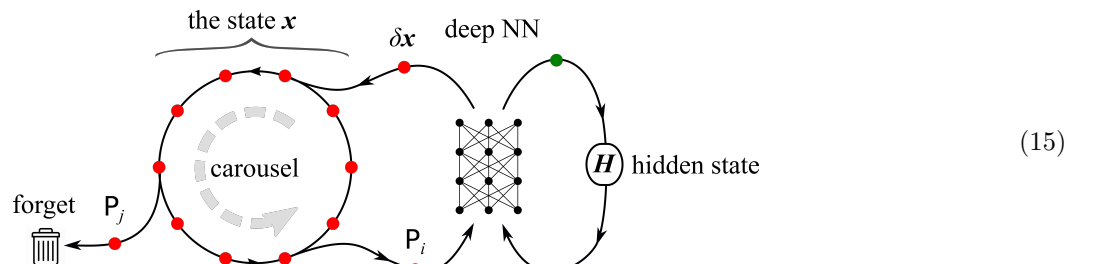
[1] Convolutional NNs are only *translation*-invariant. The Transformer [?] architecture is *equivariant* under permutations (meaning permuted inputs give permuted outputs), but it implicitly contains a recurrence similar to ours.

Notice that the working memory $\boldsymbol{x}$ is itself a hidden state, so $\boldsymbol{H}$ can be regarded as a *second-order* hidden state.

This architecture has the advantage of being simple and may be biologically plausible (the human brain's working memory).

I believe in the maxim: *Whatever can be done in time can be done in space.* The diagram (15), when unfolded in time, can be expressed in this functional form:

$$\boldsymbol{F}_{\text{sym}}(\mathsf{P}_0, ..., \mathsf{P}_k) = \boldsymbol{f}(\mathsf{P}_k, \boldsymbol{f}(\mathsf{P}_{k-1}, \boldsymbol{f}(..., \boldsymbol{f}(\mathsf{P}_0, \vec{\emptyset})))). \tag{16}$$

$\boldsymbol{F}_{\text{sym}}$ means that the function is invariant under the action of the symmetric group $\mathfrak{S}_k$ over propositions. Such symmetric NNs have been proposed in [?] [?] [?] [?] [?] [?] [?].

5.6 Logic actions

The output of a logic rule is a proposition $\delta\boldsymbol{x} \in \mathbb{P}$, the space of propositions. This is a continuous space (due to the use of Word2Vec embeddings).

From the perspective of reinforcement learning, we are performing an action $\boldsymbol{a}$ (the logic rule) from the current state $\boldsymbol{x}$. The neural network in (15) is a parametrized function $\boldsymbol{F}_\Theta$ that accepts $\boldsymbol{x}$ and outputs $\delta\boldsymbol{x}$. We want to **gradient-descent** on Θ to get the optimal set of actions, ie, a **policy**.

To solve the RL problem, 2 main options are: **value-iteration** (*eg* Q-learning) and **policy-iteration**.

In **Q-learning**, we try to learn the action-value function $Q(\boldsymbol{a}|\boldsymbol{x}) \to \mathbb{R}$. The policy is obtained by choosing $\arg\max_{\boldsymbol{a}} Q(\boldsymbol{a}|\boldsymbol{x})$ at each step. In our logic setting, the action space $\mathbb{A} \ni \boldsymbol{a}$ is continuous. As is well known, if we use an NN to represent $Q(\boldsymbol{a}|\boldsymbol{x})$, the evaluation of $\arg\max_{\boldsymbol{a}}$ would be rather awkward, which is why Q-learning is widely seen as ill-suited for continuous actions.

It is much easier for the NN to directly output (a fixed number of) stochastic actions, thus avoiding the *curse of dimensionality*. Such a function is a **stochastic policy** $\pi(\boldsymbol{a}|\boldsymbol{x})$.

In other words, we can use a fixed number of Gaussian kernels (radial basis functions) to approximate the conditional probability distribution of π over $\mathbb{A}$. For each state $\boldsymbol{x}$, our NN outputs a probabilistic *choice* of c actions. So we only need to maintain c "peaks" given by Gaussian kernels. Each peak is determined by its mean $\delta\boldsymbol{x}_i$ and variance σ. Both parameters are to be learned.

The size of the FFNN in (15) seems well within the capacity of current hardware.

5.7 Forgetting uninteresting propositions

In (10) and (15), some propositions need to be forgotten based on some measure of **interestingness**. One way to measure interestingness is through the value function of a state, $V(\boldsymbol{x})$, where $\boldsymbol{x}$ consists of propositions $\boldsymbol{p}_i$. Suppose that $V(\boldsymbol{x})$ is learned by a neural network, then it may be possible to extract (backwards) the weight by which a proposition $\boldsymbol{p}_i$ contributed to the value V. For this to work, the function $V(\boldsymbol{x})$ should be deliberately **regularized** so that it would *generalize broadly*. Notice that $V(\boldsymbol{x})$ would also be *symmetric* in the $\boldsymbol{p}_i$'s so it would have the architecture of (15). $V(\boldsymbol{x})$ is very similar to $Q(\boldsymbol{a}|\boldsymbol{x})$ but should be learned separately because they serve different purposes.

6 Remaining work

- Replace the recurrent NN architecture with symmetric NNs
- In this minimal architecture there is no **episodic memory** or **meta-reasoning** ability, but these can be added to the architecture and are not bottleneck problems. For example, meta-reasoning can be added via turning the input to introspection.
- Implementation of the system is currently under way.

Acknowledgements