AGI via Combining Logic with Deep Learning

甄景贤 (King-Yin Yan)

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's BERT, which many currently state-of-the-art language models are derived from, can be regarded as a special form of logic. Moreover, this structure can be incorporated under a reinforcement-learning framework to form a minimal AGI architecture. We also mention some insights gleaned from category and topos theory that may be helpful to practitioners of AGI.

Keywords: deep learning, symbolic logic, logic-based AI, neural-symbolic integration, Curry-Howard isomorphism, category theory, topos theory, fuzzy logic

(To the editor: Diagrams in this paper can be re-organized in the traditional format, please contact the author if this is desired. Some references are yet to be filled in.)

0 Introduction: the Curry-Howard Isomorphism

As the risk of sounding too elementary, we would go over some basic background knowledge, that may help those readers who are unfamiliar with this area of mathematics.

The Curry-Howard isomorphism expresses a connection between logic **syntax** and its underlying **proof** mechanism. Consider the mathematical declaration of a **function** f with its domain and co-domain:

$$f : A \to B. \tag{1}$$

This notation comes from type theory, where A and B are **types** (which we can think of as sets or general spaces) and the function f is an **element** in the function space $A \to B$, which is also a type.

What the Curry-Howard isomorphism says is that we can regard $A \to B$ as a **logic** formula $A \Rightarrow B$ (an implication), and the function f as a **proof** process that maps a proof of A to a proof of B.

The following may give a clearer picture:

$$\begin{array}{ll} \boxed{\text{logic}} & A \Longrightarrow B \\ & \text{- - - - - - - - - -} \\ \boxed{\text{program}} & \blacksquare \overset{f}{\longmapsto} \blacksquare \quad . \end{array} \tag{2}$$

What we see here is a logic formula "on the surface", with an underlying proof mechanism which is a **function**. Here the ■'s represent proof objects or witnesses. The logic propositions A and B coincide with the **domains** (or **types**) specified by type theory. Hence the great educator Philip Wadler calls it "propositions as types". [1] Other textbooks on the Curry-Howard isomorphism include: []

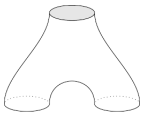
The gist of our theory is that Deep Learning provides us with neural networks (ie. non-linear functions) that serve as the proof mechanism of logic via the Curry-Howard isomorphism. With this interpretation, we can impose the mathematical structure of logic (such as symmetries) onto neural networks. Such constraints serve as **inductive bias** that can accelerate learning, according to the celebrated "No Free Lunch" theory.

In particular, logic propositions in a conjunction (such as $A \wedge B$) are commutative, ie. invariant under permutations, which is a "symmetry" of logic. This symmetry essentially decomposes a logic "state" into a set of propositions, and seems to be a fundamental feature of most logics known to humans. Imposing this symmetry on neural networks gives rise to symmetric neural networks, which can be easily implemented thanks to separate research on the latter topic. This is discussed in §3.

As an aside, the Curry-Howard isomorphism also establishes connections to diverse disciplines. Whenever there is a space of elements and some operations over them, there is a chance that it has an underlying "logic" to it. For

[1] See his introductory video: https://www.youtube.com/watch?v=IOiZatlZtGU .

example, in quantum mechanics, Hilbert space and Hermitian operators. Another example: in String Theory, strings and cobordisms between them (The following is the famous "pair of pants" cobordism, representing a process in time that splits one string into two):



(3)

1 Prior Research

1.1 Neuro-Symbolic Integration

There has been a long history of attempts to integrate symbolic logic with neural processing, with pioneers such as Ron Sun, Dov Gabbay, among others. We consider 2 approaches below.

Pascal Hitzler (*et. al.*)'s "Core Method" [7] is model-based (as in model theory in logic). An interpretation $\mathcal{I}$ is a function that assigns truth values to the set of all possible ground atoms. To a logic program P is associated a semantic operator $\mathcal{T}_P : \mathcal{I}_P \to \mathcal{I}_P$, where $\mathcal{I}_P$ is an interpretation endowed with the topology of the Cantor set. Then P is approximated by a neural network $f : \mathcal{I}_P \to \mathbb{R}$.

Pedro Domingos' ∂-ILP [**Domingos**] is also model-based, and moreover it is focused on the learning problem. A valuation is a vector $[0, 1]^n$ mapping each ground atom to a real number $\in [0, 1]$. Each clause is attached with a Boolean flag (actually relaxed to be a continuous value) to indicate whether it is included in the results or not. From each clause c one can generate a function $\mathcal{F}_c$ on valuations that implements a single step of forward inference. Then gradient descent is used to learn which clauses should be included.

The author believes that representing and learning relational knowledge is perhaps the final piece of the puzzle in AGI research. We would like to mention a recent paper, **Geoffrey Hinton's GLOM theory** [**Hinton**], which addresses the problem of representing a hierarchy of visual structures.

1.2 Cognitive Architectures and Reinforcement Learning

Reinforcement Learning (RL). In the 1980's, Richard Sutton [14] introduced reinforcement learning as an AI paradigm, drawing inspiration from Control Theory and Dynamic Programming. In retrospect, RL already has sufficient generality to be considered an AGI theory, or at least as a top-level framework for describing AGI architectures.

Relation to AIXI. AIXI is an abstract AGI model introduced by Marcus Hutter in 2000 [**Hutter2000**]. AIXI's environmental setting is the external "world" as observed by some sensors. The agent's internal model is a universal Turing machine (UTM), and the optimal action is chosen by maximizing potential rewards over all programs of the UTM. In our (minimal) model, the UTM is constrained 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.

Relation to Quantum mechanics and Path Integrals. At the core of RL is the Bellman equation, which governs the update of the utility function to reach its optimal value. This equation (in discrete time) is equivalent to the Hamilton-Jacobi equation in differential form. Nowadays they are unified as the Hamilton-Jacobi-Bellman equation, under the name "optimal control theory" [11]. In turn, the Hamilton-Jacobi equation is closely related to the Schrödinger equation in quantum mechanics:

$$\boxed{\text{Bellman eqn.}} \; - - - \; \boxed{\text{Hamilton-Jacobi eqn.}} \; - - - \; \boxed{\text{Schrödinger eqn.}} \tag{4}$$

but the second link is merely "heuristic"; it is the well-studied "quantization" process whose meaning remains mysterious to this day. Nevertheless, the path integral method introduced by Richard Feynmann can be applied to RL algorithms, eg. [**Kappen**].

The Hamilton-Jacobi equation gives the RL problem a "symplectic" structure; Such problems are best solved by so-called symplectic integrators. Surprisingly, in the RL / AI literature, which has witnessed tremendous growth in recent years, there is scarcely any mention of the Hamilton-Jacobi connection, while the most efficient heuristics (such as policy gradient, Actor-Critic, etc.) seem to exploit other structural characteristics of the "world".

2 The Mathematical Structure of Logic

Currently, the most mathematically advanced and satisfactory description of logic seems to base on category theory, known as categorial logic and topos theory. This direction was pioneered by William Lawvere in the 1950-60's. The body of work in this field is quite vast, but we shall briefly mention some points that are relevant to AGI. A more detailed tutorial on categorical logic, with a focus on AGI, is in preparation [**Yan**].

2.1 Predicates and Dependent Type Theory

The Curry-Howard isomorphism identifies *propositional* intuitionistic logic with type theory. As such, the arrow $\to$ in type theory is "used up" (it corresponds to the implication arrow $\Rightarrow$ in intuitionistic logic). However, predicates are also a kind of functions (arrows), so how could we accomodate predicates in type theory such that Curry-Howard continues to hold? This is the idea behind Martin Löf's dependent type theory.

Predicate logic (in one form or another) is a highly desirable feature as AGI knowledge representation, due to its expressiveness. So it seems necessary to incoporate dependent type theory into our logic. From a categorical perspective, predicates can be regarded as **fibers** over a base set; This is the treatment given by Bart Jacob's book [8]. The author has not yet mastered this knowledge adequately to say what implications it may have on AGI design, but it seems to offer an interesting angle.

2.2 (Fuzzy?) Topos Theory

The author's previous paper [21] proposed a fuzzy-probabilistic logic where probabilities are distributed over fuzzy truth values. To this day, he still believes that regarding fuzziness as a generalization of binary truth is philosophically sound. Thus it behoves to develop a generalization of standard topos theory to the fuzzy case.

The most important commutative diagram in Topos theory is this one:

$$\begin{array}{ccc} X & \xrightarrow{!} & 1 \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle \text{true}} \\ Y & \xrightarrow[\chi_m]{} & \Omega \end{array} \tag{5}$$

It can be understood as saying that every set is a pullback of the true map, in analogy to the idea of a "moduli space". Following this idea, is it true that every fuzzy set should be the pullback of the fuzzy true map? It seems that the theory of fuzzy topos is still an open research problem [].

3 Permutation Symmetry and Symmetric Neural Networks

From the categorical perspective, we make the following correspondence with logic and type theory:

$$\begin{array}{ccccc} \boxed{\text{product}} & A \times B & \leftrightsquigarrow & A \wedge B & \boxed{\text{conjunction}} \\ \boxed{\text{function}} & A \to B & \leftrightsquigarrow & A \Rightarrow B & \boxed{\text{implication}}. \end{array} \tag{6}$$

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

$$\mathsf{P} \wedge \mathsf{Q} \quad \Leftrightarrow \quad \mathsf{Q} \wedge \mathsf{P}. \tag{7}$$

This remains true of probabilistic logic, where $\wedge$ and $\vee$ are unified as conditional probability tables (CPTs) for the nodes in Bayesian networks.

Once we know the symmetry, the question is how to impose this symmetry on deep neural networks. Interestingly, the answer already comes from an independent line of research (namely, PointNet [**Qi2017a**] and Deep Sets [**Zaheer2017a**]) that deals with visual object recognition of point clouds:

 (8)

In a point cloud, it does not matter the order in which the points are presented, as inputs to the classifier function. Such a function needs to be permutation invariant to a huge number of points.

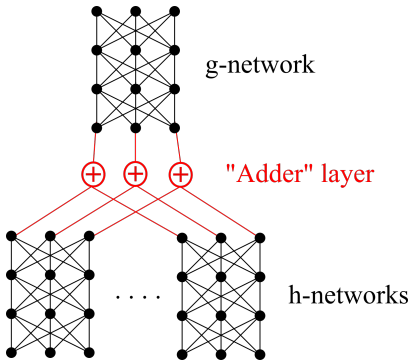
From [**Zaheer2017a**]: the **Kolmogorov–Arnold representation theorem** states that every multivariate continuous function can be represented as a sum of continuous functions of one variable:

$$f(x_1, ..., x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{9}$$

It can be specialized to such that every symmetric multivariate function can be represented as a sum of (the same) functions of one variable:

$$f(x_1, ..., x_n) = g(h(x_1) + ... + h(x_n)) \tag{10}$$

This leads to the following implementation using neural networks:

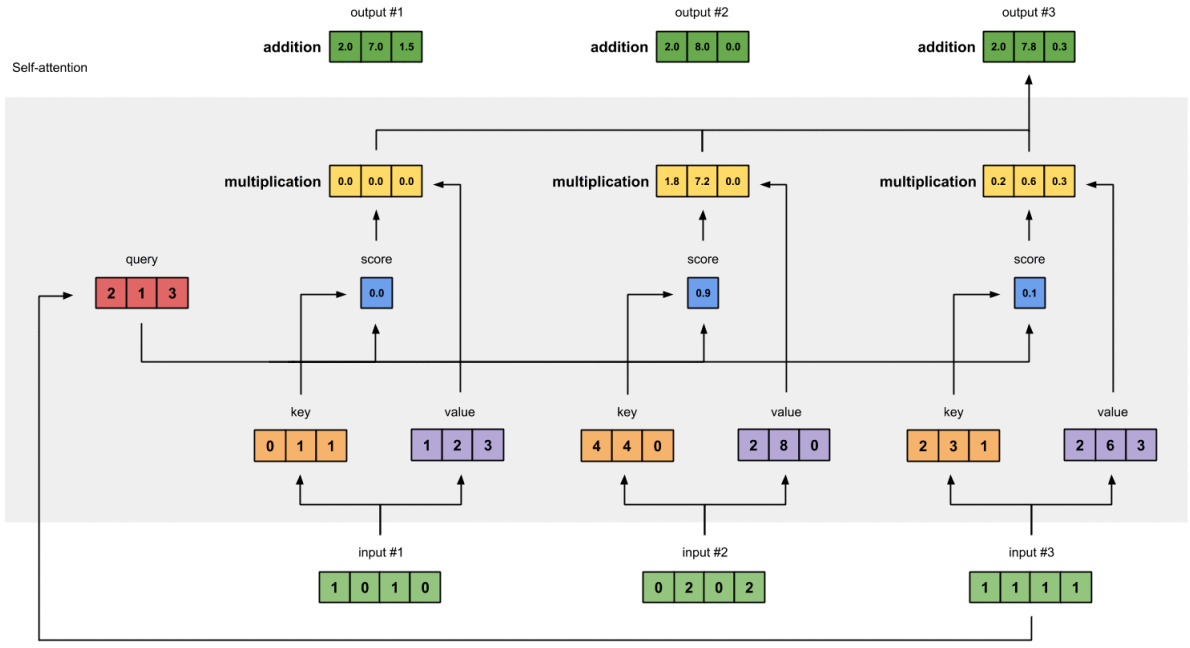


(11)

And this can be easily implemented with a few lines of Tensorflow (see §5).

3.1 Why BERT is a Logic

In the following diagram, taken from [], observe that the Transformer is permutation-invariant (or more precisely, **equivariant**). That is to say, for example, if input #1 and #2 are swapped, then output #1 and #2 would also be swapped:



(12)

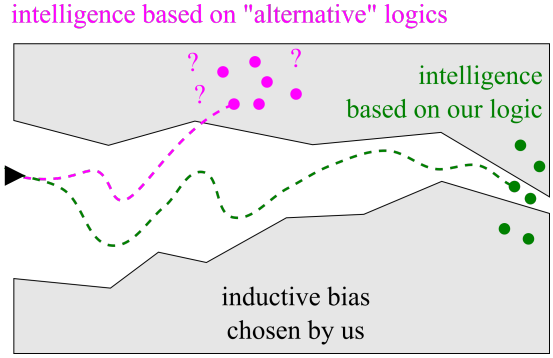
In other words, each Transformer layer takes N inputs and produces N equivariant outputs. That is the same as saying that *each* output is permutation-invariant in all its inputs. As we explained in the last section, permutation invariance is the symmetry that characterizes a logic as having *individual* propositions.

In **Multi-Head Attention**, the intermediate computations are duplicated multiple (eg, $M = 8$) times, each with their own weight matrices. From the logic point of view, this amounts to duplicating M logic rules per output. But since the next layer still expects N inputs, the M outputs are combined into one, before the next stage. Thus, from the logic point of view this merely increased the parameters *within* a single logic rule, and seems not significant to increase the power of the logic rule-base. Indeed, experimental results seem to confirm that multi-head attention is not particularly gainful towards performance.

A comment is in order here, about the choice of the word "head". In logic programming (eg Prolog), one calls the conclusion of a logic rule its "head", such as `P` in `P :- Q,R,S`. Perhaps the creators of BERT might have logic rules in mind?

4 "No Free Lunch" Theory

The following conceptual diagram illustrates the possibility that there might exist some form of logic that is drastically different from the symbolic logic currently known to humans:



(13)

but there is no efficient algorithm to find them. The permutation symmetry proposed in this paper forces our logic to be decomposable into **propositions**. Such a logical form allows a mental state to be enumerated as a list of sentences (proposition), same as the "linear" nature of human **languages**. If the AGI knowledge representation is linear (in the sequential sense) and symbolic, then it would not be far from our formulation – all these logics belong to one big family.

But could there be drastically different logics? One observes that pictures and music are not easily described by words, indeed they are 2-dimensional structures. This suggests that the brain may use **multi-dimensional** matrices of features to represent the world. Such a "logic" would be very different from sequential logic and it would be interesting and fruitful to analyze the relation between them.

5 Experiment

A simple test of the symmetric neural network, under reinforcement learning (policy gradient), has been applied to the Tic-Tac-Toe game. [2]

The state of the game is represented as a set of 9 propositions, where all the propositions are initialized as "null" propositions. During each step of the game, a new proposition is added to the set (ie. over-writing the null propositions). Each proposition encodes who the player is, and which square (i, j) she has chosen. In other words, it is a predicate of the form: `move(player,i,j)`. The neural network takes all 9 propositions as input, and outputs a new proposition; Thus it is a permutation-invariant function.

In comparison, the game state of traditional RL algorithms (eg. AlphaGo []) usually is represented as a vector of dimension same as the chessboard (eg. 3×3 in Tic-Tac-Toe and 8×8 in Chess). This state vector remains the same constant length even if there are very few pieces on the chessboard. Our logic-based representation may offer some advantages over the board-vector representation, and likely induces a different way of "reasoning" about the game.

In our Tic-Tac-Toe experiment, convergence of learning is observed, but the algorithm fell short of achieving the highest score (19 instead of 20), and the score displayed unstable oscillating behavior after it got near the optimal value. Further investigation is required, but it seems to be a promising start.

[2] Code with documentation is on GitHub: https://github.com/Cybernetic1/policy-gradient

6 Conclusion and Future Directions

We described a minimal AGI with a logic that can derive one new proposition per iteration. This seems sufficient to solve simple logic problems such as Tic-Tac-Toe. As a next step, we would consider inference rules with multi-proposition conclusions. The latter seems essential to **abductive** reasoning. For example, one can deduce the concept "apple" from an array of visual features; Conversely, the idea of an "apple" could also evoke in the mind a multitude of features, such as color, texture, taste, and the facts such as that it is edible, is a fruit, and that Alan Turing died from eating a poisoned apple (a form of episodic memory recall), and so on. This many-to-many inference bears some similarity to the brain's computational mechanisms [**Rolls**] [**Rolls**] [**Boraud**]. It may seem surprising, but there may be a rough correspondence between the biological brain and a logic-baseed, reinforcement learning agent. The author is embarking on an abstract unifying AGI theory that makes references to (but not necessarily copying) brain mechanisms.

Acknowledgements

Thanks Ben Goertzel for suggesting that neural networks are advantageous over pure symbolic logic because they have fast learning algorithms (by gradient descent). That was at a time when "deep learning" was not yet a popular word. Thanks Dmitri Tkatch for pointing me to existing research of symmetric neural networks. Thanks Dr. 肖达 (Da Xiao) for explaining details of BERT.

Also thanks to the following people for invaluable discussions over many years: Ben Goertzel, Pei Wang (王培), Abram Demski, Russell Wallace, Juan Carlos Kuri Pinto, SeH, Jonathan Yan, and others. Also thanks to all the university professors and researchers in Hong Kong (especially in the math departments, and their guests), strangers who taught me things on Zhihu.com (知乎), Quora.com, and StackOverflow.

References

[1] Andreka, Nemeti, and Sain. "Handbook of philosophical logic". In: *Handbook of philosophical logic*. Springer, 2001. Chap. Algebraic logic, pp. 133–247.

[2] de Bie, Peyré, and Cuturi. "Stochastic deep networks". In: (2019). `https://arxiv.org/pdf/1811.07429.pdf`.

[3] Gens and Domingos. "Deep symmetry networks". In: *Advances in neural information processing systems* 27 (2014), pp. 2537–2545.

[4] Goguen. "What is unification - a categorical view of substitution, equation and solution". In: *Resolution of equations in algebraic structures* 1: algebraic techniques (1989), pp. 217–261.

[5] Halmos. *Algebraic logic*. Chelsea, 1962.

[6] Halmos. *Logic as algebra*. Math Asso of America, 1998.

[7] Hitzler and Seda. *Mathematical aspects of logic programming semantics*. CRC Press, 2011.

[8] Bart Jacobs. *Categorical logic and type theory*. Elsevier, 1999.

[9] Lawvere. "Functorial semantics of algebraic theories". PhD thesis. Columbia university, 1963.

[10] Lawvere and Rosebrugh. *Sets for mathematics*. Cambridge, 2003.

[11] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton Univ Press, 2012.

[12] Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in Neural Information Processing Systems* (2017), pp. 5105–5114.

[13] Ravanbakhsh, Schneider, and Poczos. "Equivariance through parameter-sharing". In: (2017). `https://arxiv.org/abs/1702.08389`.

[14] Sutton. "Temperal credit assignment in reinforcement learning". University of Massachusetts, Amherst, 1984.

[15] Tarski, Henkin, and Monk. *Cylindric algebras, Part I*. 1971.

[16] Tarski, Henkin, and Monk. *Cylindric algebras, Part II*. 1985.

[17] Vaswani et al. "Attention is all you need". In: (2017). `https://arxiv.org/abs/1706.03762`.

[18] Wikipedia. *No free lunch theorem*. `https://en.wikipedia.org/wiki/No_free_lunch_theorem`. URL: `https://en.wikipedia.org/wiki/No_free_lunch_theorem`.

[19] Wikipedia. *Rete algorithm*. `https://en.wikipedia.org/wiki/Rete_algorithm`. URL: `%5Curl%7Bhttps://en.wikipedia.org/wiki/Rete_algorithm%7D`.

[20] Wikipedia. *Word2vec*. `https://en.wikipedia.org/wiki/Word2vec`. URL: `%5Curl%7Bhttps://en.wikipedia.org/wiki/Word2vec%7D`.

[21] King-Yin Yan. "Fuzzy-probabilistic logic for common sense reasoning". In: *Artificial general intelligence 5th international conference, LNCS 7716* (2012).

[22] Zaheer et al. "Deep sets". In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 3391–3401.

7 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)?

We propose an AGI architecture:

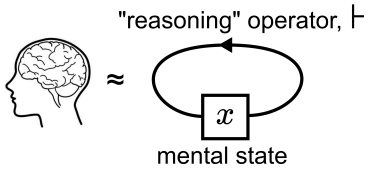
1. with **reinforcement learning** (RL) as top-level framework
   - State space = mental space
2. **Logic** structure is imposed on the **knowledge representation** (KR)
   - State transitions are given by logic rules = actions in RL
   - The logic state $\boldsymbol{x}$ is decomposable into **propositions** (§8.4)
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 (§3)
   - **Stochastic** actions are represented by **Gaussian kernels** (radial basis functions) (§8.6), thus partly avoiding the curse of dimensionality

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

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:

(14)

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



(15)

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..."

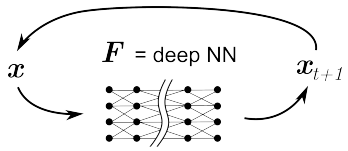
Thinking is the process of **transitioning** from one mental state to another. As I am writing now, I use my mental states to keep track of where I am at within the sentence's syntax, so that I can construct my sentence grammatically.

7.1 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"):



(16)

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 [18]) — so we will further modify $\boldsymbol{F}$ by imposing the logic structure of reasoning on it (§8.4 and §3).

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 deep neural networks allows to use a polynomial number of parameters to represent an exponential number of mappings.

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

8 Logic structure

8.1 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.

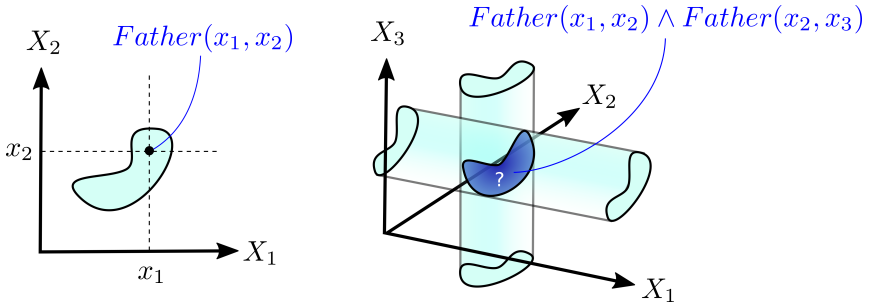
8.2 Geometry induced by logic rules

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

$$\forall X\ \forall Y\ \forall Z.\quad \text{father}(X,Y) \wedge \text{father}(Y,Z) \Rightarrow \text{grandfather}(X,Z) \tag{17}$$

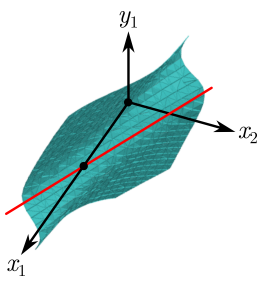
where the red lines show what I call "linkages" between different appearances of the same variables.

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 [15] [16] [1] [5] [6]. Cylindrical shapes can arise from quantification as illustrated below:



(18)

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



(19)

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.

8.3 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:

$$\overbrace{\mathsf{C}_1^1\,\mathsf{C}_2^1\,\mathsf{C}_3^1 \;\wedge\; \underbrace{\mathsf{C}_1^2\,\mathsf{C}_2^2\,\mathsf{C}_3^2}_{\text{each literal made of } m \text{ atomic concepts, } m=3 \text{ here}} \;\wedge\; \ldots. \;\wedge\; \mathsf{C}_1^k\,\mathsf{C}_2^k\,\mathsf{C}_3^k}^{\text{conjunction of } k \text{ literal propositions}} \quad \Rightarrow \quad \overbrace{\mathsf{C}_1^0\,\mathsf{C}_2^0\,\mathsf{C}_3^0}^{\text{conclusion}} \tag{20}$$

where a **concept** can be roughly understood as a **word vector** as in Word2Vec [20]. 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 $\boldsymbol{x}$:

$$\mathsf{C}_1^1\,\mathsf{C}_2^1\,\mathsf{C}_3^1 \;\wedge\; \mathsf{C}_1^2\,\mathsf{C}_2^2\,\mathsf{C}_3^2 \;\wedge\; \ldots. \;\wedge\; \mathsf{C}_1^k\,\mathsf{C}_2^k\,\mathsf{C}_3^k. \tag{21}$$

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.

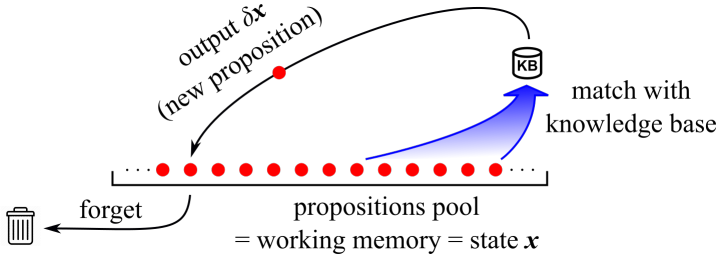
The **output** of the NN would be the conditional **probability** of an action:

$$P(\text{action} \mid \text{state}) := \pi(\,\mathsf{C}_1\,\mathsf{C}_2\,\mathsf{C}_3 \mid \boldsymbol{x}). \tag{22}$$

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 (§**??**).

8.4 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:



(23)

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

$$\boxed{\text{Match}} \quad (\boldsymbol{x} \stackrel{?}{=} \text{KB}) \;:\; \mathbb{X} \to (\mathbb{X} \to \mathbb{P}) \qquad \boldsymbol{x} \mapsto \boldsymbol{r} \tag{24}$$

  — In categorical logic, matching is seen as finding the **co-equalizer** of 2 terms which returns a **substitution** [4] [9] [10] [8]. 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 [19] turns the set of rules into a tree-like structure which is efficient for solving (24).

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

$$\boxed{\text{Apply}} \quad \begin{aligned} &\boldsymbol{r} : \mathbb{X} \to \mathbb{P} \\ &\boldsymbol{x} \mapsto \boldsymbol{r}(\boldsymbol{x}) = \delta\boldsymbol{x} \end{aligned} \tag{25}$$

- **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 $\delta\boldsymbol{x}$:

$$\boxed{\text{Update}} \quad \boldsymbol{x} = (\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_j, ..., \mathsf{P}_k) \mapsto (\mathsf{P}_1, \mathsf{P}_2, ..., \delta\boldsymbol{x}, ..., \mathsf{P}_k) \tag{26}$$

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

8.5 Commutativity of logic conjunctions

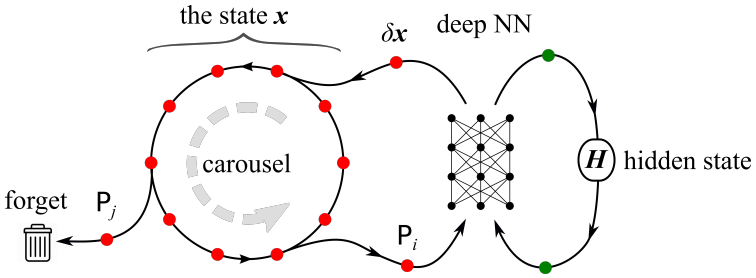
One basic characteristic of (classical) logic is that the conjuction $\wedge$ is **commutative**:

$$\mathsf{P} \wedge \mathsf{Q} \quad \Leftrightarrow \quad \mathsf{Q} \wedge \mathsf{P}. \tag{27}$$

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 [3] 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:



(28)

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 (28), when unfolded in time, can be expressed in this functional form:

$$\boldsymbol{F}_{\text{sym}}(\mathsf{P}_0, ..., \mathsf{P}_k) = \boldsymbol{f}(\mathsf{P}_k, \boldsymbol{f}(\mathsf{P}_{k-1}, \boldsymbol{f}(..., \boldsymbol{f}(\mathsf{P}_0, \vec{\emptyset})))). \tag{29}$$

$\boldsymbol{F}_{\text{sym}}$ 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 [3] [2] [**Ravanbakhsh2016**] [13] [**Qi2016**] [12] [22].

8.6 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).

[3] Convolutional NNs are only *translation*-invariant. The Transformer [17] architecture is *equivariant* under permutations (meaning permuted inputs give permuted outputs), but it implicitly contains a recurrence similar to ours.

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 (28) 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 (28) seems well within the capacity of current hardware.

8.7 Forgetting uninteresting propositions

In (23) and (28), 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 (28). $V(\boldsymbol{x})$ is very similar to $Q(\boldsymbol{a}|\boldsymbol{x})$ but should be learned separately because they serve different purposes.

9 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.