Wandering in the Labyrinth of Thinking
– a minimalist cognitive architecture combining reinforcement learning, deep learning, and logic structure

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

Abstract. The bottleneck algorithm in AGI is the inductive learning of knowledge. The importance of this algorithm in our AI era is like what the steam engine was to the industrial revolution. This paper: 1) outlines a minimalist AGI architecture; 2) defines the Lagrangian and Hamiltonian for it, thus opening new algorithms based on control theory; 4) defines a logic structure for AGI systems; 5) imposes the logic structure on the control system, as an inductive bias to speed up learning.

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
   - The external environment is turned "inward"
   - State space = mental space
   - **Policy-gradient** and **Hamiltonian** methods are employed to speed up learning (§1.4 - §1.7)
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 (§2.4)
3. The set of logic rules is approximated by a deep-learning neural network (**deep NN**)
   - Just the most basic kind of feed-forward neural network is required
   - Logic conjunctions are **commutative**, so working-memory elements can be presented in any order (§2.6)
   - **Stochastic** actions are represented by **Gaussian kernels** (radial basis functions) (§2.7), thus partly avoiding the curse of dimensionality

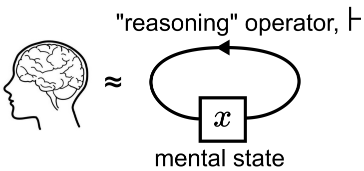
The rest of this paper will explain these design features in detail.

1 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:

(1)

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



(2)

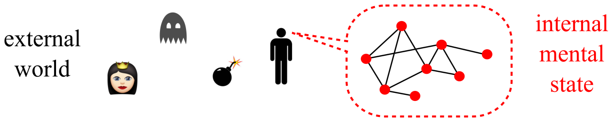
A mental state is a **set of propositions**, for example:

- 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.

1.1 "Introspective" view of reinforcement learning

Traditionally, RL deals with acting in an *external* environment; value / utility is assigned to *external* states. In this view, the *internal* mental state of the agent may change without any noticeable change externally:



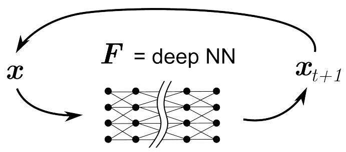
(3)

1.2 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 implemented as a simple feed-forward neural network (where "deep" simply means "many layers"):



(4)

However, this naive idea has to be modified by the logic structures in §2.4 and §2.6.

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

1.3 Comparison with AIXI [20]

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 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.

1.4 Control-theoretic setting

The cognitive state is a vector $\boldsymbol{x} \in \mathbb{X}$ where $\mathbb{X}$ is the space of all possible cognitive states, the reasoning operator $\vdash$ or $\boldsymbol{F}$ is an **endomorphism** (an **iterative map**) $\mathbb{X} \to \mathbb{X}$.

Mathematically this is a **dynamical system** that can be defined by:

$$\boxed{\text{discrete time}} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{5}$$

$$\text{or } \boxed{\text{continuous time}} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{6}$$

where $\boldsymbol{f}$ and $\boldsymbol{F}$ are different but related [1]. For ease of discussion, sometimes I mix discrete-time and continuous-time notations.

[1] See *eg* [3] §8.2.3.

A **control system** is a dynamical system added with the control vector $\boldsymbol{u}(t)$:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{7}$$

The goal of control theory is to find the optimal $\boldsymbol{u}^*(t)$ function, such that the system moves from the initial state $\boldsymbol{x}_0$ to the terminal state $\boldsymbol{x}_\perp$.

1.5 Lagrangians and Hamiltonians

In **reinforcement learning**, we are concerned with two quantities:

- $r(\boldsymbol{x}, \boldsymbol{u})$ = **reward** of doing action $\boldsymbol{u}$ in state $\boldsymbol{x}$
- $U(\boldsymbol{x})$ = **utility** or **value** of state $\boldsymbol{x}$

where **utility** is the integral of instantaneous **rewards** over time:

$$\boxed{\text{utility } U} = \int \boxed{\text{reward } r}\, dt. \tag{8}$$

There is a well-known correspondence between control theory, dynamic programming (reinforcement learning), and analytical mechanics, observed by Kalman and Pontryagin, among others (*cf* the textbook [12]):

Reinforcement learning	Control theory	Analytical mechanics
value V or utility U	cost J	action S
instantaneous reward r	running cost L	Lagrangian L
action $\boldsymbol{a}$	control $\boldsymbol{u}$	(external force)
	Lagrange multiplier $\boldsymbol{\lambda}$	momentum $\boldsymbol{p}$
$U = \int R\, dt$	$J = \int L\, dt + \Phi(\boldsymbol{x}_\perp)$	$S = \int L\, dt$

(9)

Φ is the **terminal cost**, ie, the value when the terminal state $\boldsymbol{x}_\perp$ is reached.

Interestingly, the reward r corresponds to the **Lagrangian** in physics, whose unit is "energy"; In other words, "desires" or "happiness" appear to be measured by units of "energy", this coincides with the idea of "positive energy" in pop psychology. Whereas, long-term value is measured in units of [energy $\times$ time].

The **Hamiltonian** can be defined by

$$H := \langle \boldsymbol{p}, \boldsymbol{f} \rangle - L \tag{10}$$

which arises from the **Lagrange multiplier** $\boldsymbol{\lambda} \equiv \boldsymbol{p}$. It can be shown that, at the optimum,

$$\boldsymbol{\lambda} = \frac{\partial J^*}{\partial \boldsymbol{x}} \quad \text{or} \quad \boldsymbol{p} = \frac{\partial S}{\partial \boldsymbol{x}} \tag{11}$$

where * refers to the extremum, and $\langle \cdot, \cdot \rangle$ is the inner product.

In the classical **variational calculus**, the **optimal path** is given by the condition:

$$\boxed{\text{variational calculus}} \quad \frac{\partial H}{\partial \boldsymbol{u}} = 0 \tag{12}$$

which is later generalized by Pontryagin as the **maximum principle**:

$$\boxed{\text{Pontryagin}} \quad H^* = \inf_{u \in \mathbb{U}} H(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{u}, t) \tag{13}$$

and also roughly equivalently to the **Hamilton-Jacobi-Bellman equation**:

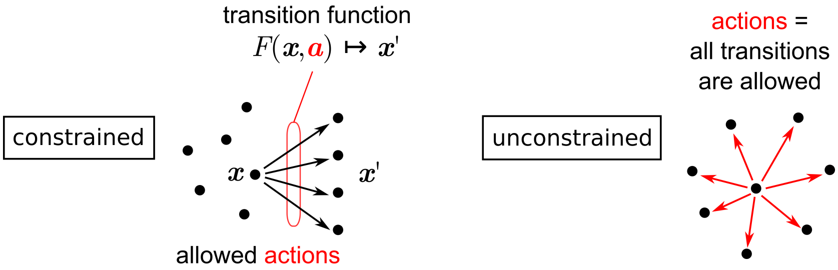
$$\boxed{\text{Hamilton-Jacobi-Bellman}} \quad \frac{\partial S^*}{\partial t} = -\inf_u H = -\inf_u \left\{ L + \left\langle \frac{\partial S^*}{\partial \boldsymbol{x}}, \boldsymbol{u} \right\rangle \right\}. \tag{14}$$

Traditional logic-based AI systems are discrete-time; changing them to continuous-time seems to merely increase the computational burden and is *ungainful*. But the time-critical step is the learning of $\boldsymbol{u}$, which may be solved via (12), (13), or (14).

From the author's limited knowledge of control theory, it seems we currently don't have efficient algorithms to sovle the HJB equation, but the maximum principle (13) is more useful in practice, though it requires the Hamiltonian to be defined in addition to the Lagrangian.

1.6 Constrained vs unconstrained dynamics

In our formulation, every state is potentially **reachable** by some logic rule, this corresponds to the picture on the right:



(15)

Under this view, the control rule (7) simplifies to:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) = \boldsymbol{u}(t) \tag{16}$$

and then the Lagrangian multiplier $\boldsymbol{\lambda} \equiv \boldsymbol{p}$ follows from (10) and (12) to be:

$$\boldsymbol{\lambda}^* \equiv \boldsymbol{p}^* = \frac{\partial L(t, \boldsymbol{x}^*, \boldsymbol{u}^*)}{\partial \boldsymbol{u}} = \frac{\partial L}{\partial \dot{\boldsymbol{x}}} \tag{17}$$

which recovers the classical definition of **momemtum**.

Remember, the control-theoretic problem begins with the definition of the Lagrangian L as a function of $(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t)$, where $\dot{\boldsymbol{x}} = \boldsymbol{u}$ in our unconstrained case. In this setup, the concept of "mass" is unnecessary, but L must be a function of $\dot{\boldsymbol{x}}$ (thus $\boldsymbol{u}$), or else the variational problem becomes trivial. A crucial fact is that $L(\boldsymbol{x}(t), \boldsymbol{u}(t))$ is the same as $R(\boldsymbol{x}|\boldsymbol{a})$, ie, the **reward** obtained at state $\boldsymbol{x}$ after performing action $\boldsymbol{a}$.

The general procedure is to solve for the control $\boldsymbol{u}$ via the variational principle (12). This leads to first solving for $\boldsymbol{p}$ via the **Euler-Lagrange equation**:

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{x}}} = \dot{\boldsymbol{p}} \tag{18}$$

which is basically Newton's $\boldsymbol{F} = m\boldsymbol{a}$. Then we can solve for $\boldsymbol{u}$ via (17).

Symplectic integrators preserve qualitatively the geometry of Hamiltonian flows. For example, the phase-space trajectory of a pendulum is a closed loop, but some integration methods such as **Runge-Kutta** may give trajectories that move towards the origin or diverge to ∞.

For more advanced theories on *discrete* mechanics and control, *cf* [13], [8], [14], *etc*.

1.7 Policy gradient

In recent years, the **policy gradient** method and its variants (*eg* Actor-Critic [2] [16]) has made spectacular success in deep reinforcement learning (DRL). Basically, the **stochastic** policy $\pi(\boldsymbol{a}|\boldsymbol{x})$ is expressed as a function parametrized by Θ and is updated via:

$$\Theta \stackrel{\pm}{=} \eta \, \nabla_\Theta \widetilde{V} \tag{19}$$

where η is the **learning rate**, and $\widetilde{V}$ is the objective function, which is the **expectation** of the total reward or value V along *all possible* trajectories τ starting from an initial position:

$$\widetilde{V} = \mathop{\mathbb{E}}_{\tau}[\ V(\tau)\ ]. \tag{20}$$

The gradient of $\widetilde{V}$ can be derived as this formula familiar to practitioners of DRL [17]:

$$\nabla_\Theta \widetilde{V} = \nabla_\Theta \mathop{\mathbb{E}}_{\tau}[\ V(\tau)\ ] = \mathop{\mathbb{E}}_{\tau}[\ \nabla_\Theta \sum_t \log \pi(\boldsymbol{a}_t|\boldsymbol{x}_t; \Theta) V(\tau)\ ]. \tag{21}$$

Current deep-learning RL literature seems to focus on using the reward (*ie*, Lagrangian), so they have objective functions like the form in (21), which is cumbersome as the total value V is itself a summation inside another summation over all trajectories. Gradient descent $\nabla_{\boldsymbol{u}}$ against the Hamiltonian H may be computationally more efficient.

1.8 Connection with quantum mechanics

Recently, I accidentally discovered [24] [2] a precise transition from the classical H-J equation to the **Schrödinger equation** in quantum mechanics, via a simple substitution $\Psi = e^{iS/\hbar}$,

$$\boxed{\text{Hamilton-Jacobi}} \quad \frac{\partial S}{\partial t} = -H \quad \xrightarrow{\Psi = \exp\{iS/\hbar\}} \quad i\hbar \frac{\partial \Psi}{\partial t} = H\Psi \quad \boxed{\text{Schrödinger}}. \tag{22}$$

This implies that the Schrödinger equation is an alternative way of expressing the optimality condition for RL! It is also known that the Schrödinger equation in *imaginary time* becomes the **diffusion equation** and is related to stochastic processes (*cf* [15] Ch.6); This may lead to new algorithms.

2 Logic structure

2.1 Logic is needed as an inductive bias

The transition function $\boldsymbol{F}$ appearing in (5) is "free" without further restrictions. The learning of $\boldsymbol{F}$ may be slow without further **induction bias**, *cf* the "no free lunch" theorem [21]. But we know that the transition function 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** containing a large number of logic **rules**, as in the setting of classical logic-based AI.

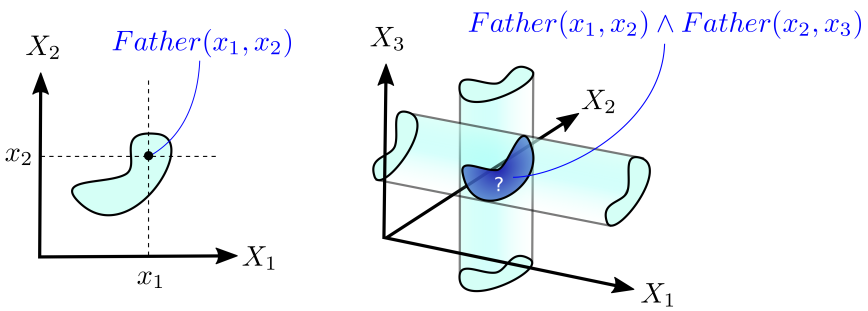
2.2 Geometry induced by logic rules

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

$$\forall X\ \forall Y\ \forall Z. \quad \text{father}(X, Y) \wedge \text{father}(Y, Z) \Rightarrow \text{grandfather}(X, Z) \tag{23}$$

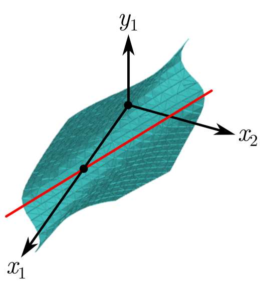
where the red lines show what I call "linkages" between different appearances of the same variables.

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 [18] [19] [1] [5] [6]. Cylindrical shapes can arise from quantification as illustrated below:



(24)

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



(25)

[2] The relation $S = i\hbar \log \Psi$ appeared in one of Schrödinger's 1926 papers, but is dismissed by him as "incomprehensible". This formula seems to be overlooked by physicists since that time, possibly including Feynman. I have yet to discuss / verify this with physicists.

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.

2.3 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:

$$\overbrace{\mathsf{C}^1_1\,\mathsf{C}^1_2\,\mathsf{C}^1_3 \;\wedge\; \underbrace{\mathsf{C}^2_1\,\mathsf{C}^2_2\,\mathsf{C}^2_3}_{\text{each literal made of } m \text{ atomic concepts, } m=3 \text{ here}} \wedge \;....\; \wedge\;\; \mathsf{C}^k_1\,\mathsf{C}^k_2\,\mathsf{C}^k_3}^{\text{conjunction of } k \text{ literal propositions}} \quad\Rightarrow\quad \overbrace{\mathsf{C}^0_1\,\mathsf{C}^0_2\,\mathsf{C}^0_3}^{\text{conclusion}} \tag{26}$$

where a **concept** can be roughly understood as a **word vector** as in Word2Vec [23]. 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 $\boldsymbol{x}$:

$$\mathsf{C}^1_1\,\mathsf{C}^1_2\,\mathsf{C}^1_3 \;\wedge\; \mathsf{C}^2_1\,\mathsf{C}^2_2\,\mathsf{C}^2_3 \;\wedge\; .... \;\wedge\; \mathsf{C}^k_1\,\mathsf{C}^k_2\,\mathsf{C}^k_3. \tag{27}$$

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.

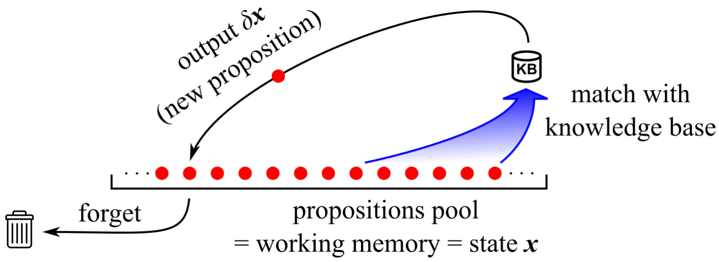
The **output** of the NN would be the conditional **probability** of an action:

$$P(\text{action} \mid \text{state}) := \pi(\,\mathsf{C}_1\,\mathsf{C}_2\,\mathsf{C}_3 \mid \boldsymbol{x}). \tag{28}$$

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 (§2.7).

2.4 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:



(29)

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

$$\boxed{\text{Match}} \quad (\boldsymbol{x} \stackrel{?}{=} \text{KB}) \;:\; \mathbb{X} \to (\mathbb{X} \to \mathbb{P}) \qquad \boldsymbol{x} \mapsto \boldsymbol{r} \tag{30}$$

  — In categorical logic, matching is seen as finding the **co-equalizer** of 2 terms which returns a **substitution** [4] [9] [10] [7]. 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 [22] turns the set of rules into a tree-like structure which is efficient for solving (30).

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

$$\boxed{\text{Apply}} \quad \begin{aligned} \boldsymbol{r} &: \mathbb{X} \to \mathbb{P} \\ \boldsymbol{x} &\mapsto \boldsymbol{r}(\boldsymbol{x}) = \delta\boldsymbol{x} \end{aligned} \tag{31}$$

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

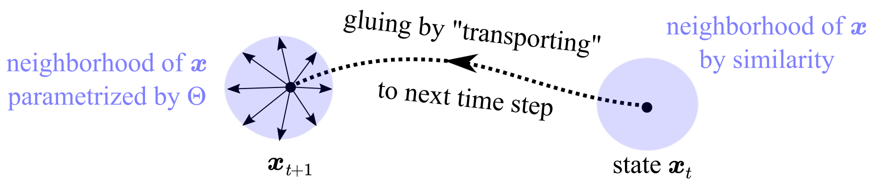
$$\boxed{\text{Update}} \quad \boldsymbol{x} = (\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_j, ..., \mathsf{P}_k) \mapsto (\mathsf{P}_1, \mathsf{P}_2, ..., \delta\boldsymbol{x}, ..., \mathsf{P}_k) \tag{32}$$

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

2.5 Topological / metric structure of the domain $\mathbb{X}$

In order to apply control theory, we need to calculate entities such as $\dot{\boldsymbol{x}}$ and $\dfrac{\partial L}{\partial \dot{\boldsymbol{x}}}$. For this, the domain $\mathbb{X}$ needs to have some kind of differentiable structure. So we proceed as follows:

- Time steps are discrete. Using continuous time increases the computational burden seemingly without any benefits.
- Atomic concepts C's are embedded in a vector space, thanks to the technique of **Word2Vec** [23]. We assume that such a "concept embedding" is sensible without further explaining its justification.
- Propositions $\mathsf{P} \in \mathbb{P}$ are composed of atomic concepts, hence they are also embeddable in vector space.
- The state $\boldsymbol{x} \in \mathbb{X}$ is a set (seen as a list) of propositions, thus also inherits a vector-space embedding.
- However, it is well-known that **syntactic** distance can be very different from **semantic** distance (sentences that appear similar may differ drastically in meaning). From the point of view of logic, the 2 metrics <u>must not be confused</u>.
- It is also well-known that the semantic distance (ie, how many logic steps are required to deduce one logic state from another) is related to **Kolmogorov complexity** [11] and is *incomputable*, which however can be *approximated*. It is my belief that all AGI systems must approximate this metric in one way or another.
- For each time step, the state $\boldsymbol{x}$ should move by one logic rule $\boldsymbol{u} \equiv \dot{\boldsymbol{x}}$; If a proposition has merely moved via syntactic similarity, this is not considered "genuine" movement in the dynamical / control theoretic sense. Each logic step is 1 unit of semantic distance.
- In our architecture, logic rules are parametrized by $\Theta \in \mathbb{R}^N$ which are the weights of a neural network $\boldsymbol{F}$. Thus, rules exist in a **continuum**.
- The output of a rule is a new proposition $\delta\boldsymbol{x} \in \mathbb{P}$ which is also embedded in vector space.
- Therefore, at each time step, a point $\boldsymbol{x}$ in state space $\mathbb{X}$ is only allowed to move *via* the continuum of rules parametrized by $\Theta \in \mathbb{\Theta}$, which is different from the neighborhood of $\boldsymbol{x}$ based on similarity:



(33)

(Now the neighborhood of $\boldsymbol{x}_{t+1}$ is a subset $\subset \mathbb{\Theta}$ which replaces $\mathbb{X}$)

- In short, what we have here is <u>discrete-time dynamics occurring in continuous space</u>.
- The objective function J rewards correct answers while penalizing logic path lengths, thus forcing the system to acquire intelligence. This knowledge is represented in the function $\boldsymbol{F}$ parametrized by Θ. It is intuitively obvious that $\boldsymbol{F}$ contains an implicit approximation of Kolmogorov complexity.

2.6 Commutative structure of logic conjunctions

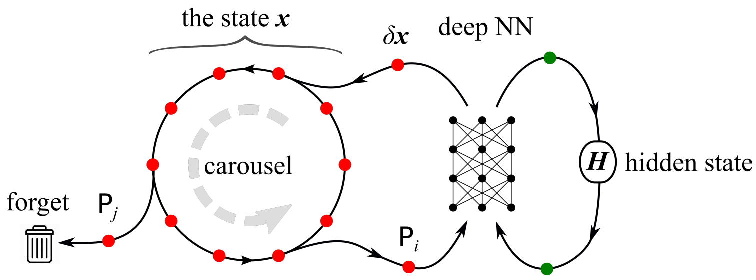
One basic characteristic of (classical) logic is that the conjuction $\wedge$ is **commutative**:

$$\mathsf{P} \wedge \mathsf{Q} \quad \Leftrightarrow \quad \mathsf{Q} \wedge \mathsf{P}. \tag{34}$$

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 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:



(35)

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

$$F_{\text{sym}}(\mathsf{P}_0, ..., \mathsf{P}_n) = f(\mathsf{P}_n, f(\mathsf{P}_{n-1}, f(..., f(\mathsf{P}_0, \emptyset)))). \tag{36}$$

It may be possible to solve the functional equation to eliminate the recurrent structure.

2.7 Probability distribution over actions

All the "knowledge" of the agent is contained in the **policy** function π:

$$\begin{aligned} \pi : \mathbb{X} \times \mathbb{A} &\to [0, 1] \in \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{a}) &\mapsto P(\boldsymbol{a} \mid \boldsymbol{x}) \end{aligned} \tag{37}$$

where $\mathbb{X}$ = state space, $\mathbb{A}$ = action space, $P(\cdot)$ = conditional probability.

In reinforcement learning in general, the function space of π is of shape:

$$\pi : \mathbb{X} \to \mathbb{R}(\mathbb{A}) = \mathbb{R}^{\mathbb{A}}. \tag{38}$$

For example, if $\mathbb{A}$ has finitely 10 discrete actions, $\mathbb{R}(\mathbb{A})$ would be $\mathbb{R}^{10}$. For logic-based agents, $\mathbb{A}$ would be the set of all logic rules, thus very large. It would be worse if $\mathbb{A}$ is continuously-valued, as when we embed logic rules in vector space.

So we may use **Gaussian kernels** (*ie*, radial basis functions) to approximate $\pi(\boldsymbol{a}|\boldsymbol{x})$:

$$P(\boldsymbol{a}|\boldsymbol{x}) \approx \hat{P}(\boldsymbol{a}|\boldsymbol{x}) := \frac{1}{N\sigma} \sum_{i=1}^{N} \Phi\left(\frac{\boldsymbol{a} - \boldsymbol{a}_i}{\sigma}\right) \tag{39}$$

where $\Phi(\boldsymbol{\xi})$ is the Gaussian kernel $\frac{1}{\sqrt{2\pi}} e^{-\boldsymbol{\xi}^2/2}$.

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 $\boldsymbol{a}_i$ and variance σ. Both parameters are to be learned.

An action $\boldsymbol{a} \in \mathbb{A}$ is a logic rule that takes the state $\boldsymbol{x}$ to a proposition P, *ie*, $\mathbb{A} = \mathbb{X} \to \mathbb{P}$. When a rule is applied to a state, it becomes a proposition, so the space of *applied* actions $\mathbb{A}(\boldsymbol{x})$ in our case is equivalent to $\mathbb{P}$.

Each applied action $\boldsymbol{a}(\boldsymbol{x}) \in \mathbb{P}$ is of the form $\mathsf{C}_1\,\mathsf{C}_2\,\mathsf{C}_3$ and is of size $\mathbb{R}^{3d}$, as in (26). Let the hidden state $\boldsymbol{H} \in \mathbb{H} = \mathbb{R}^h$. Thus our NN has the shape:

$$\pi()(\boldsymbol{x}) : (\mathbb{P} \times \mathbb{H}) \to (\mathbb{P} \times \mathbb{R})^c \times \mathbb{H} \quad = \quad \mathbb{R}^{3d+h} \to \mathbb{R}^{(3d+1)c+h}. \tag{40}$$

Judging from the dimensions, such a neural network is well feasible with current hardware.

3 Remaining work

- Eliminate the recurrent structure of the "carousel"
- 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

I would like to thank the following people for invaluable discussions over many years: Ben Goertzel, Pei Wang (王培), Abram Demski, Russell Wallace, SeH, Juan Carlos Kuri Pinto, Jonathan Yan, Xiao Da (肖达), Dmitri Tkatch, 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*, chapter Algebraic logic, pages 133–247. Springer, 2001.
2. Barto, Sutton, and Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, (13):834–846, 1983.
3. Dolotin and Morozov. *Introduction to non-linear algebra.* World Scientific, 2007.
4. Goguen. What is unification - a categorical view of substitution, equation and solution. *Resolution of equations in algebraic structures*, 1: algebraic techniques:217–261, 1989.
5. Halmos. *Algebraic logic.* Chelsea, 1962.
6. Halmos. *Logic as algebra.* Math Asso of America, 1998.
7. Bart Jacobs. *Categorical logic and type theory.* Elsevier, 1999.
8. Lall and West. Discrete variational hamiltonian mechanics. *J Phys A: Math Gen*, pages 5509–5519, 2006.
9. Lawvere. *Functorial semantics of algebraic theories.* PhD thesis, Columbia university, 1963.
10. Lawvere and Rosebrugh. *Sets for mathematics.* Cambridge, 2003.
11. Li, Vitányi, and Paul. *An introduction to Kolmogorov complexity and its application.* Springer, 3rd edition.
12. Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction.* Princeton Univ Press, 2012.
13. Marsden and West. Discrete mechanics and variational integrators. *Acta Numerica*, pages 357–514, 2001.
14. Stern and Desbrun. Discrete geometric mechanics for variational time integrators. *Discrete differential geometry: an applied introduction*, 2008.
15. Strocchi. *An introduction to the mathematics of quantum mechanics – a short course for mathematicians.* World Scientific, 2nd edition, 2008.
16. Sutton. Temperal credit assignment in reinforcement learning, 1984.
17. Sutton and Barto. *Reinforcement learning.* MIT, 2nd edition, 2018.
18. Tarski, Henkin, and Monk. *Cylindric algebras, Part I.* 1971.
19. Tarski, Henkin, and Monk. *Cylindric algebras, Part II.* 1985.
20. Wikipedia. AIXI. `https://en.wikipedia.org/wiki/AIXI`.
21. Wikipedia. No free lunch theorem. `https://en.wikipedia.org/wiki/No_free_lunch_theorem`.
22. Wikipedia. Rete algorithm. `https://en.wikipedia.org/wiki/Rete_algorithm`.
23. Wikipedia. Word2vec. `https://en.wikipedia.org/wiki/Word2vec`.
24. Yan. Classical interpretation of the quantum state ψ. `https://github.com/Cybernetic1/latex-2019/blob/master/meaning-of-quantum-state-Psi.pdf`, 2019.