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 to the AI era is like what the steam engine is to the industrial era. We propose a minimalist architecture and consider efficient learning algorithms under it. There are 2 possible variations: one uses a simple deep learning module, the other uses more advanced, stochastic Hamiltonian control theory.

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
2. **Logic** structure is imposed on the **knowledge representation** (KR)
   - State transitions are given by logic rules
   - Actions in RL = right-hand side of logic rules
3. The set of logic rules is approximated by a deep-learning neural network (**deep NN**)
   - Logic conjunctions are **commutative**, so working-memory elements can be presented in any order (§2.5)
   - **Policy-gradient** methods (and variants) may be employed to speed up learning
   - Logic propositions are embedded in "continuous" space, so we have **continuous actions** in RL. The probability distribution over actions can be modeled by **Gaussian kernels** (radial basis functions).

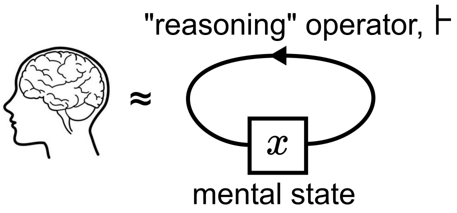
The rest of this paper will explain these design features in detail.

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)

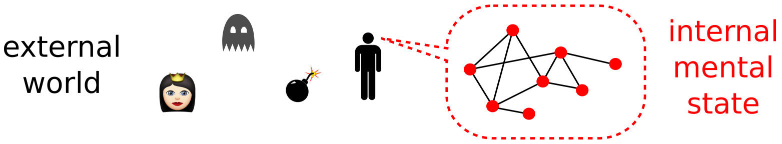
A mental state is a **set of propositions**, for example:

- 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:



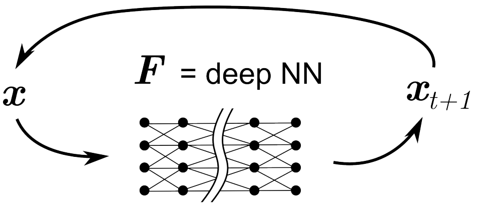
(3)

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 §2.5.

In this minimal architecture there is no **episodic memory**, but this does not seem to be a bottleneck problem.

1.3 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 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. For ease of discussion, sometimes I mix discrete-time and continuous-time notations.

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 [1]):

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 × 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:

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

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

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

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

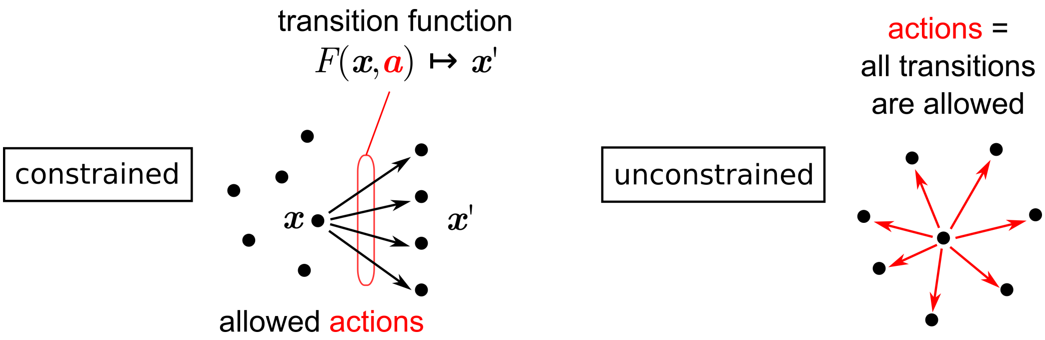
$$\boxed{\text{Hamilton-Jacobi-Bellman}} \qquad \frac{\partial S^*}{\partial t} = -\inf_u H = -\inf_u \left\{ L + \left\langle \frac{\partial S^*}{\partial \boldsymbol{x}}, \boldsymbol{u} \right\rangle \right\}. \tag{14}$$

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 in 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 [1].

1.7 Policy gradient

In recent years, the **policy gradient** method and its variants (*eg* Actor-Critic) 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 $\varTheta$ and is updated via:

$$\varTheta \overset{\pm}{=} \eta \, \nabla_{\varTheta} \widetilde{V} \tag{18}$$

[1] 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 formulation, the concept of "mass" is unnecessary, but L must depend on $\dot{\boldsymbol{x}}$ (thus $\boldsymbol{u}$) or else the variational problem becomes trivial.

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:

$$\widetilde{V} = \mathop{\mathbb{E}}_{\tau}[\, V(\tau)\,]. \tag{19}$$

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

$$\nabla_{\Theta}\widetilde{V} = \nabla_{\Theta}\mathop{\mathbb{E}}_{\tau}[\, V(\tau)\,] = \mathop{\mathbb{E}}_{\tau}[\, \nabla_{\Theta} \sum_{t} \log \pi(\boldsymbol{a}_t | \boldsymbol{x}_t ; \Theta) V(\tau)\,]. \tag{20}$$

Current deep-learning RL literature seems to focus on using the reward (*ie*, Lagrangian), so they have objective functions like the form in (20), 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 Discrete mechanics and control theory

According to [2], a formulation of discrete mechanics also begins with the definition of the Lagrangian, but instead of defining on the tangent bundle $T\mathbb{X}$, it is defined on $\mathbb{X} \times \mathbb{X}$, where $\mathbb{X}$ is the configuration manifold, and the discrete velocity $\dot{\boldsymbol{x}}$ is seen as $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t$.

Then one can derive the **discrete Euler-Lagrange equation**:

$$D_2 L_d(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, h) + D_1 L_d(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, h) = 0 \tag{21}$$

where D_1 and D_2 denote partial differentiation w.r.t. the first and second arguments of L_d (the discrete Lagrangian), respectively, and h is a time step.

1.9 Connection with quantum mechanics

Recently, I accidentally discovered [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!

[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.

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. 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.

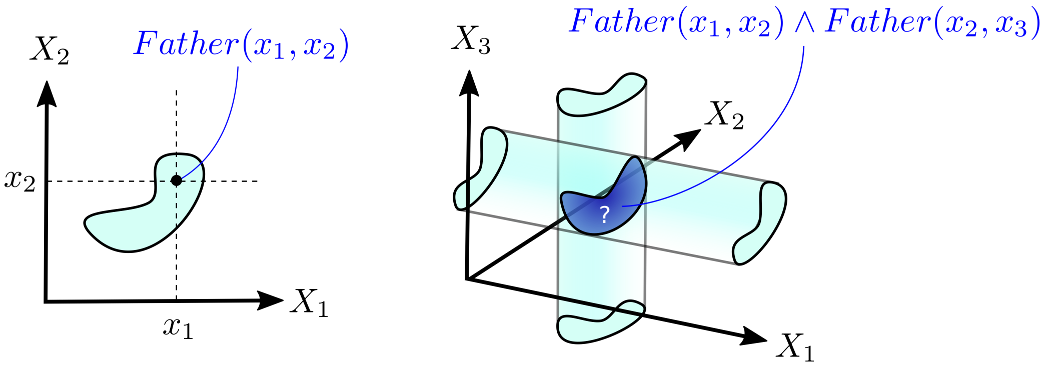
2.2 Geometry induced by logic rules

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

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

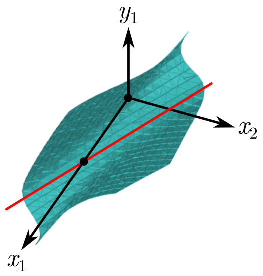
where the red lines show what I call “linkages” between different appearances of the same variables.

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. 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)

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 $\boldsymbol{p} \in \mathbb{P}$
- 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. 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.

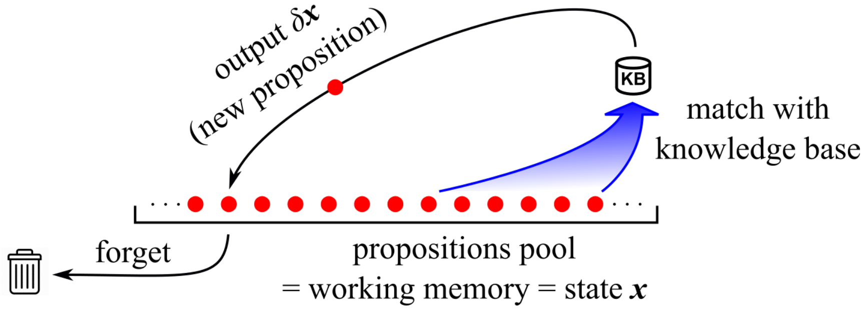
The **output** of the NN would be the conditional **probability** of an action:

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

Note that we don't just want the action itself, we need the **probability distribution** over these actions. The **Bellman update** of reinforcement learning should update the probability distribution over such actions.

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:


output $\delta\boldsymbol{x}$
(new proposition)
KB
match with
knowledge base
forget
propositions pool
= working memory = state $\boldsymbol{x}$

(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

$$\boxed{\text{Match}} \qquad \begin{aligned} (\boldsymbol{x} \overset{?}{=} \text{KB}) \ &: \mathbb{X} \to (\mathbb{X} \to \mathbb{P}) \\ \boldsymbol{x} &\mapsto \boldsymbol{r} \end{aligned} \tag{30}$$

  — 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 (30).

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

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

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

$$\boxed{\text{Update}} \qquad \boldsymbol{x} = (\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_k) \mapsto (\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \delta\boldsymbol{x}, ..., \boldsymbol{p}_k) \tag{32}$$

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

2.5 Commutative structure of logic conjunctions

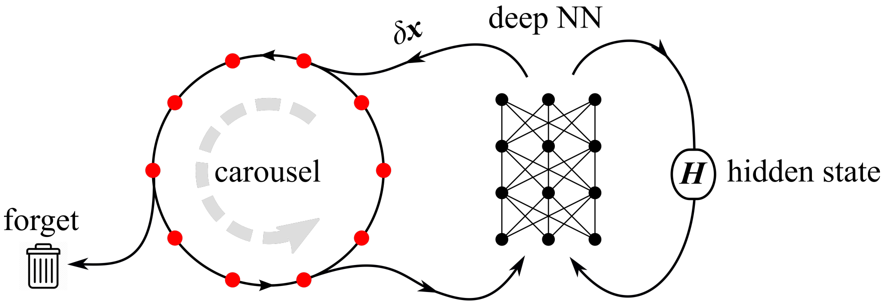
The logic conjuction $\wedge$ is **commutative**:

$$\boldsymbol{p} \wedge \boldsymbol{q} \quad \Leftrightarrow \quad \boldsymbol{q} \wedge \boldsymbol{p}. \tag{33}$$

If we want to use a neural network to model the deduction operator $\vdash : \mathbb{P}^k \rightarrow \mathbb{P}$, 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 and one proposition at a time:



(34)

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

2.6 Probability distribution over continuous actions

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

$$\begin{aligned} \pi : \mathbb{X} \times \mathbb{A} &\rightarrow [0,1] \in \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{a}) &\mapsto P(\boldsymbol{a} \mid \boldsymbol{x}) \end{aligned} \tag{35}$$

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:

$$\pi : \mathbb{X} \rightarrow \mathbb{R}(\mathbb{A}) = \mathbb{R}^{\mathbb{A}}. \tag{36}$$

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 $\pi(\boldsymbol{a}|\boldsymbol{x})$:

$$P(\boldsymbol{a}|\boldsymbol{x}) \approx \hat{P}(\boldsymbol{a}|\boldsymbol{x}) := \frac{1}{Nh} \sum_{i=1}^{N} \Phi\left(\frac{\boldsymbol{a} - \boldsymbol{a}_i}{h}\right) \tag{37}$$

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

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 h. 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 $\boldsymbol{p}$, *ie*, $\mathbb{A} = \mathbb{X} \rightarrow \mathbb{P}$. 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:

$$\pi()(\boldsymbol{x}) : (\mathbb{P} \times \mathbb{H}) \rightarrow (\mathbb{P} \times \mathbb{R})^c \times \mathbb{H} \quad = \quad \mathbb{R}^{3d+H} \rightarrow \mathbb{R}^{(3d+1)c+H}. \tag{38}$$

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

3 Conclusion and future directions

This paper has 3 main contributions:

- Outline of an AGI architecture
- The use of Hamiltonian-maximization gradient descent for deep reinforcement learning
- "Carousel" architecture of working memory to achieve commutativity

References

1. Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction.* Princeton Univ Press, 2012.
2. Marsden and West. Discrete mechanics and variational integrators. *Acta Numerica*, pages 357–514, 2001.