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

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

Abstract. Introducing a minimalist cognitive architecture based on reinforcement learning and deep learning. The system consists of an iterative function whose role is analogous to the consequence operator (⊢) in logic, and is approximated by a deep neural network (this is the key efficiency-boosting aspect).

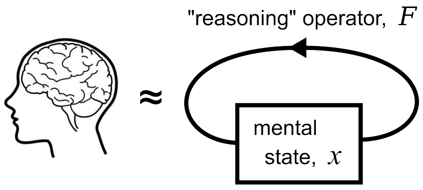
Keywords: reinforcement learning, control theory, deep learning, cognitive architecture

0 Main idea

The **metaphor** here is that of reinforcement learning 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)

For example, a mental state could be the following set of propositions:

- I am in my room, writing a paper for AGI-17.
- 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. Even as I am speaking now, I use my mental state to keep track of where I am at within the sentence's syntax, so that I can structure my sentences grammatically.

1 Control theory / dynamical systems theory

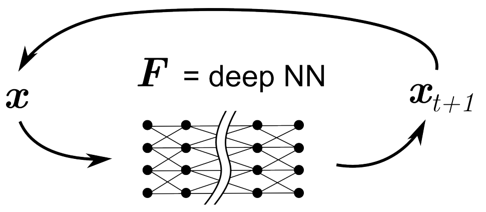
The cognitive state is a vector $\boldsymbol{x} \in \mathbb{X}$ where $\mathbb{X}$ is the space of all possible cognitive states, the reasoning operator $\boldsymbol{F}$ is an **endomorphism** (an **iterative map**) $\mathbb{X} \to \mathbb{X}$.

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

$$\boxed{\text{discrete time}} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{3}$$

$$\boxed{\text{continuous time}} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{4}$$

In our cognitive architecture design, $\boldsymbol{F}$ is implemented as a deep neural network (the word "deep" simply means "many layers"):



(5)

A **neural network** is a non-linear operator with many parameters (called "weights"):

each layer's **weight** matrix　　　total # of layers

$$F(\boldsymbol{x}) = \oslash(W_1 \oslash(W_2 ... \oslash(W_L\ \boldsymbol{x}))) \tag{6}$$

$\oslash$ is a sigmoid-shaped non-linear function, applied component-wise to the vectors.

If continuous-time, $\boldsymbol{f}$ can also be implemented as neural network, but $\boldsymbol{f}$ and $\boldsymbol{F}$ are different in nature, they are related by: $\boldsymbol{x}(t+1) = \boldsymbol{F}(\boldsymbol{x}(t))$. 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 $\boldsymbol{u}(t)$:

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

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 $\boldsymbol{x}_\perp$.

A typical control-theory problem is described by:

$$\boxed{\text{state equation}} \qquad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \tag{8}$$

$$\boxed{\text{boundary condition}} \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,,\ \boldsymbol{x}(t_\perp) = \boldsymbol{x}_\perp \tag{9}$$

$$\boxed{\text{objective function}} \qquad J = \int_{t_0}^{t_\perp} L[\boldsymbol{x}(t), \boldsymbol{u}(t), t]dt \tag{10}$$

and we seek the optimal control $\boldsymbol{u}^*(t)$.

According to control theory, the condition for **optimal path** is given by the Hamilton-Jacobi equation:

$$\boxed{\text{Hamilton-Jacobi equation}} \qquad 0 = \frac{\partial J^*}{\partial t} + \min_u H \tag{11}$$

In the next section I will explain the meaning of J, L and H.

2 Reinforcement learning / dynamic programming

Reinforcement learning is a branch of machine learning that is particularly suitable for controlling an **autonomous agent** who interacts with an **environment**. It uses **sensory perception** and **rewards** to continually modify its **behavior**. The exemplary image you should invoke in mind is that of a small insect that navigates a maze looking for food and avoiding predators:

A reinforcement learning system consists of a 4-tuple:

$$\boxed{\text{reinforcement learning system}} = (\boldsymbol{x} \in \text{States}, \boldsymbol{u} \in \text{Actions}, R = \text{Rewards}, \pi = \text{Policy}) \tag{12}$$

For details readers may see my *Reinforcement learning tutorial* [6].

U is the total rewards of a sequence of actions:

$$\overset{\text{total value of state 0}}{U(\boldsymbol{x}_0)} \quad = \quad \sum_t \overset{\text{reward at time } t}{R(\boldsymbol{x}_t, \boldsymbol{u}_t)} \tag{13}$$

For example, the value of playing a chess move is not just the immediate reward of that move, but includes the consequences of playing that move (eg, greedily taking a pawn now may lead to checkmate 10 moves later). Or, faced with delicious food, some people may choose not to eat, for fear of getting fat.

The central idea of **Dynamic programming** is the **Bellman optimality condition**. Richard Bellman in 1953 proposed this formula, while he was working at RAND corporation, dealing with operations research problems.

The **Bellman condition** says: "if we cut off a tiny bit from the endpoint of the optimal path, the remaining path is still an optimal path between the new endpoints."

$$\boxed{\text{Bellman equation}} \quad \overset{\text{value of entire path}}{U^*(\boldsymbol{x})} = \max_{\boldsymbol{u}} \{ \overset{\text{reward of choosing } \boldsymbol{u} \text{ at current state}}{R(\boldsymbol{u})} + \overset{\text{value of rest of path}}{U^*(\boldsymbol{x}_{t+1})} \} \tag{14}$$

This seemingly simple formula is the entire content of dynamic programming; What it means is that: When seeking the path with the best value, we cut off a bit from the path, thus reducing the problem to a smaller problem; In other words, it is a **recursive relation** over time.

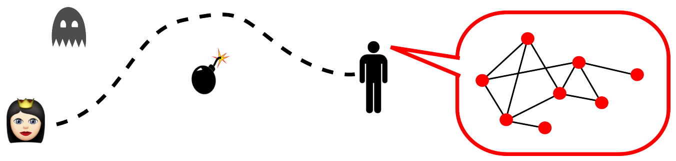
In AI reinforcement learning there is an oft-employed trick known as Q-learning. Q value is just a variation of U value; there is a U value for each state, and Q is the **decomposition** of U by all the actions available to that state. In other words, Q is the utility of doing action $\boldsymbol{u}$ in state $\boldsymbol{x}$. The relation between Q and U is:

$$U(\boldsymbol{x}) = \max_{\boldsymbol{u}} Q(\boldsymbol{x}, \boldsymbol{u}) \tag{15}$$

The advantage of Q is the ease of learning. We just need to learn the value of actions under each state. This is so-called "**model free learning**".

2.1 Internal vs external 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:



(16)

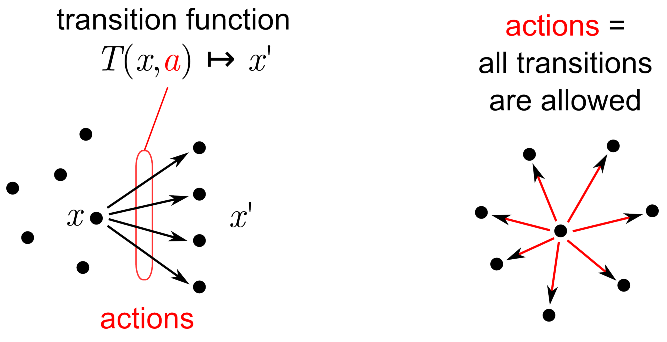
The usefulness of RL (ie, the Bellman update technique) makes me wonder if it'd be advantageous to invert the perspective to consider the internal mental landscape instead. Then, an internal state would have higher utility when it has been visited by many successful "thinking" trajectories.

2.2 Actions = cognitive state-transitions = "thinking"

In our system there are 2 things that need to be learned:

1. The transition function $\boldsymbol{F} : \boldsymbol{x} \mapsto \boldsymbol{x}'$. $\boldsymbol{F}$ represents the **knowledge** that constrains thinking. In other words, the learning of $\boldsymbol{F}$ is the learning of "static" knowledge.
2. Find the optimal trajectory of the state $\boldsymbol{x}$. This corresponds to optimal "thinking" under the constraints of static knowledge.

In traditional reinforcement learning (left view), the system chooses an action $\boldsymbol{a}$, and the transition function $\boldsymbol{F}$ gives the probability of reaching each state $\boldsymbol{x}_i$ given action $\boldsymbol{a}$. In our model (right view), all possible cognitive states are potentially **reachable** from any other state, and therefore the action a coincides with the next state x'.



(17)

2.3 Prior art: other cognitive architectures

The minimalist architecture based on reinforcement learning has been proposed by Itimar Ariel from Israel, in 2012 [2], and I also independently proposed in 2016 (precursor of this paper). The prestigious researcher of signal processing, Simon Haykin, recently also used the "RL + memory" design, cf. his 2012 book *Cognitive dynamic systems* [3]. Vladimir Anashin in the 1990's also proposed this kind of cognitive architecture [1]. There may exist more precedents, eg: [4].

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 restricted to an RNN, where the RNN's **state** is analogous to the UTM's **tape**, and the optimal weights (program) are found via Bellman optimality.

3 Logical structure

The transition function F appearing in (3) is "free" without further restrictions. The learning of F may be slow without further **induction bias**, *cf* the "no free lunch" theorem. But we know that the transition function is similar to the logic consequence or entailment operator $\vdash$. So we would like to impose the structure of deductive logic on F.

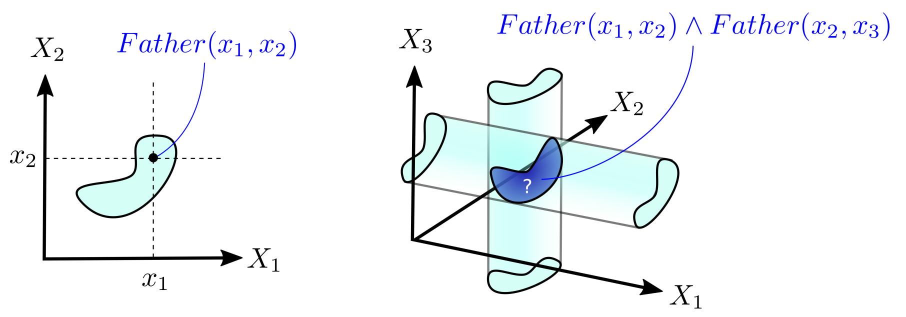
By logic structure we mean that F would act like a knowledge base containing a large number of logic **rules**, as in the setting of classical logic-based AI.

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

$$\forall X\ \forall Y\ \forall Z.\quad \text{father}(X, Y) \wedge \text{father}(Y, Z) \Rightarrow \text{grandfather}(X, Z) \tag{18}$$

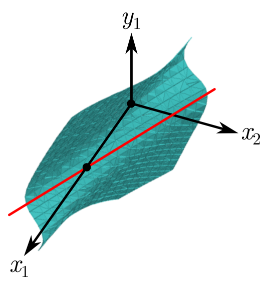
where the red lines show what I call "linkages" between different appearances of the same variables.

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. That cylindrical shapes can arise from quantification is illustrated below:



(19)

And "linkages" cause the graph of the $\vdash$ map to pass through diagonals as follows:



(20)

We are trying to use neural networks to approximate such functions (*ie*, these geometric shapes). Since NNs are universal function approximators, they can in principle achieve that. 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; And we know that deep learning is fairly successful at the latter task.

So what exactly is the logic structure?

Recall that inside our RL model:

- state = mental state = set of logic propositions
- environment = state space = mental space
- actions = logic rules

Basically, an action = a logic rule is of the form:

$$\mathsf{xxx} \wedge \mathsf{xxx} \wedge .... \Rightarrow \mathsf{xxx} \tag{21}$$

where xxx denotes a logic **proposition**.

Each proposition is a composition of 3 atomic concepts (think of these as word vectors as in Word2Vec):

$$\text{proposition} = \mathsf{xxx} = \mathsf{x} \cdot \mathsf{x} \cdot \mathsf{x}. \tag{22}$$

Each $\mathsf{x} \in \mathbb{R}^n$, where n is the dimension needed to represent a single word-vector (or atomic concept).

An **action** is the conclusion of a rule, *ie*, the right-hand side of (21).

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:

$$\mathsf{xxx}_1 \wedge \mathsf{xxx}_2 \wedge ....\mathsf{xxx}_m. \tag{23}$$

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

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

$$p(\mathsf{xxx}). \tag{24}$$

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.

3.1 Implementation issues

All the “knowledge” of the agent is contained in the Q-learning function:

$$\begin{aligned} Q : \mathbb{X} \times \mathbb{A} \to [0,1] \in \mathbb{R} \\ (x,a) \mapsto P(a) \end{aligned} \tag{25}$$

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

The function space of Q is equivalent to:

$$\mathbb{X} \to \mathbb{R}(\mathbb{A}) = \mathbb{R}^{\mathbb{A}} \tag{26}$$

which is very large. For example, if $\mathbb{A}$ has finitely 10 discrete actions, $\mathbb{R}(\mathbb{A})$ would be $\mathbb{R}^{10}$. It would be much worse if $\mathbb{A}$ is continuously-valued, but there exists a number of techniques to deal with continuous actions in the RL literature.

For our purpose, I think the **Gaussian kernel** (*ie*, radial basis function) method is very suitable. The true probability distribution is approximated by:

$$P(\boldsymbol{a}) \approx \hat{P}(\boldsymbol{a}) = \frac{1}{Nh} \sum_{i=1}^{N} \Phi\left(\frac{\boldsymbol{a} - \boldsymbol{a}_i}{h} \right) \tag{27}$$

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

For each state x, our NN should output a probabilistic *choice* of k actions. So we only need to maintain k “peaks” given by Gaussian kernels. Each peak is determined by its mean (a vector) and variance (a scalar). We fix the variance globally as the parameter h. So our NN, ie, Q-function, would be of the form:

$$Q : \mathbb{X} \to \mathbb{A}^k \tag{28}$$

where each $a \in \mathbb{A}$ is of the form xxx and is of size $\mathbb{R}^{3n}$, as explained above. Thus our NN is of the from:

$$\begin{aligned} Q : \mathbb{X} \to \mathbb{A}^k \\ = (\mathbb{R}^{3n})^m \to (\mathbb{R}^{3n})^k \\ = \mathbb{R}^{3mn} \to \mathbb{R}^{3kn} \end{aligned} \tag{29}$$

4 Algorithm

The algorithm is exactly the same as the standard Q-learning algorithm:

```
Initialize all $Q(x,a)$ arbitrarily
For all episodes
    Initialize $x$
```

```
Repeat
    Choose $a$ using policy derived from $Q$, eg $\epsilon$-greedy
    Take action $a$, observe $R$ and $x'$
    Update $Q(x,a)$:
        $Q(x,a) \overset{+}{=} \eta \left[R + \gamma \max_{a'} Q(x',a') - Q(x,a)\right]$
    $x \leftarrow x'$
Until $x$ is terminal state
```

5 Future directions

- **Memory:** In this minimal architecture there is no episodic memory, this will be dealt with in a later paper.

Appendix: connections with Hamiltonian control and quantum mechanics

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 $\boldsymbol{x}$

Simply put, **utility** is the integral of instantaneous **rewards** over time:

$$\boxed{\text{utility } U} = \int \boxed{\text{reward } R}\, dt \tag{30}$$

In **control-theoretic** parlance, it is usually defined the **cost functional**:

$$\boxed{\text{cost } \ J} = \int L dt + \varPhi(\boldsymbol{x}_\perp) \tag{31}$$

where L is the **running cost**, ie, the cost of making each step; $\varPhi$ is the **terminal cost**, ie, the value when the terminal state $\boldsymbol{x}_\perp$ is reached.

In **analytical mechanics** L is known as the **Lagrangian**, and the time-integral of L is called the **action**:

$$\boxed{\text{action } \ S} = \int L dt \tag{32}$$

Hamilton's **principle of least action** says that S always takes the **stationary value**, ie, the S value is extremal compared with neighboring trajectories.

The **Hamiltonian** is defined as $H = L + \frac{\partial J^*}{\partial \boldsymbol{x}} \boldsymbol{f}$, which came from the method of **Lagrange multipliers**. For details please refer to my *Control theory tutorial*[**?**].

All these are the same thing, so there is this correspondence:

Reinforcement learning	**Control theory**	**Analytical mechanics**
utility or value U	cost J	action S
instantaneous reward R	running cost	Lagrangian L
action a	control u	(external force?)

(33)

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].

This correspondence of these 3 theories is explained in detail in Daniel Liberzon's book [5]. While this correspondence has very interesting philosophical implications, it may not be very useful in practice: the traditional AI system is discrete-time; converting it to continuous-time seems to increase the computational burden. The recent advent of **symplectic integrators** [] are known to produce better numerical solutions that retain qualitative features of the exact solution, eg. quasi-periodicity.

The equation of motion for the continuous-time case is the famed **Hamilton-Jacobi-Bellman equation**:

$$\boxed{\text{Hamilton-Jacobi-Bellman}} \qquad 0 = \frac{\partial U^*}{\partial t} + \min_u H. \tag{34}$$

With the substitution $\Psi = e^{iU/\hbar}$ into the HJB equation, one can obtain the **Schrödinger equation**:

$$\boxed{\text{Schrödinger}} \qquad i\hbar \frac{\partial \Psi}{\partial t} = \hat{H}\Psi \tag{35}$$

which means techiques in quantum mechanics can, in principle, be applied to solve our AGI problem.

References

1. Vladimir Anashin and Andrei Khrennikov. *Applied algebraic dynamics.* de Gruyter, 2009.
2. Itamar Arel. *Deep reinforcement learning as Foundations for Artificial Intelligence*, chapter 6, pages 89–102. Atlantis Press, 2012.
3. Simon Haykin. *Cognitive dynamic systems.* Cambridge Univ Press, 2012.
4. Vladimir Ivancevic and Tijana Ivancevic. *Geometrical dynamics of complex systems: a unified modeling approach to physics, control, biomechanics, neurodynamics and psycho-socio-economical dynamics.* Springer, 2006.
5. Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction.* Princeton Univ Press, 2012.
6. King Yin Yan. Reinforcement learning tutorial `https://drive.google.com/file/d/0{B}x3_S9{S}Exak-X29uazI1YnhoNFU/view`.