The ultimate goal of mathematics is to eliminate any need for intelligent thought.

— Alfred North Whitehead

Genifer 4.1 theoretical notes

YKY (甄景贤)

July 19, 2015

Let me recap a bit, what I've been trying to do....

In 2013, Tomáš Mikolov et al in Google created **Word2Vec**, that became very influential. Basically it associates to each (English) word a vector in continuous space (that requires some form of approximation), and this approximation is based on the **Distributional Hypothesis**, which says, "words that are used and occur in the same contexts tend to purport similar meanings".

My purpose is not to explore how to calculate the distributive representation of words (or concepts), but to build up from this representation of primitive concepts, assuming that this task is completed, whether it uses Word2Vec or some other methods.

Our goal is to pass from **words** to **sentences**, and then to perform upon them **deduction** and **inductive learning**.

1 Deduction

In classical logic-based AI, a set of logical rules **acts on** logical facts to give rise to new facts. We can denote the set of facts as G, the **cognitive state** of the system. Thus the rules R act on G:

$$R : G \mapsto G$$
$$R : \mathbf{G} \to \mathbf{G}$$

Notation:

$$\begin{aligned} G &= \text{a single cognitive state} \\ \mathbf{G} &= \text{space of all cognitive states} \\ R : \mathbf{G} \to \mathbf{G} &= \text{rules operating on cognitive states} \end{aligned}$$

We can also call R the **single-step deduction operator**.

The goal of **inductive learning** is to learn the set of rules R.

So far, this is classical AI.

In the new formulation, sentences (or logical statements) are built up from words (or atomic concepts). Words can be placed in vector space via algorithms such as Word2Vec. Sentences can constructed from words via tensor products (there are multiple ways to do this, and we will explore this later), and such a sentence representation inherits the vector space structure of words. The KB[1] is a collection (set union) of sentences. All in all, the cognitive state G can be represented in vector space.

So, without change in notation, but now $G \in \mathbf{G}$ is a vector, $\mathbf{G}$ is a vector space, R is an *operator* acting on $\mathbf{G}$:

$$R : G \mapsto G$$
$$R : \mathbf{G} \to \mathbf{G}$$

1.1 Continuous relaxation

Without explicitly saying so, we have already *relaxed* R so that it need not correspond to a single logical deduction step. Look at this chain of deduction:

$$G_0 \xrightarrow{R} G_1 \xrightarrow{R} ... \xrightarrow{R} G_\infty$$

where G_∞ is the final cognitive state (we want to train R to give rise to an ideal final state G^*). We do not require R to correspond to the *discrete* steps of logical deduction. R can be "fractional" or even "infinitesimal", ie, it changes G by just a small (or infinitesimal) amount. This might be a good feature to have.

[1] or more precisely, the facts base, ie the cognitive state G

2 Integrating with reinforcement learning

Richard Sutton's view is that the **reinforcement learner (RL)** should be the *top-level* architecture of an AI, which I agree.[2]

So the logic sub-system would be a module inside the RL. Question is how to link the RL with the logic module. Sutton has also mentioned, that the challenge in reinforcement learning is how to give the RL an efficient **world model**, and I think that role is to be filled by the logic sub-system. Without a world model, the RL only has knowledge about state transitions, and the real world has an astronomical number of states. The logic sub-system *condenses* world knowledge into generalized logic statements.

The RL can be viewed as a black box that inputs $\{states, actions, rewards\}$ and outputs a $policy$. In order for RL to control the logic sub-system, we need to view the logic inference engine as a machine with controls. What are the internal **states** of this engine? What are the **actions** that transform the states? What are the rewards?

Say you have a boomerang, you learn to throw the boomerang so that it flies back to you beautifully. That is the job of the RL, to learn how to act in the real world. The logic sub-system can also be viewed as an object in the real world, which we could *manipulate*. But something is tricky here: the inference engine *itself* derives logical conclusions to advice the RL on decisions and actions.

[2] see my introductory blog article on reinforcement learning

On the highest level, the inference engine can only perform 2 actions: deduction and inductive learning. We need to *refine* these actions so the RL can "micro-control" the inference engine.

The rewards should be passed from the RL itself, as we have no other way to know what is desirable for the system.

Think of a chain of logical inference:

$$\Gamma \vdash \Delta \vdash \Lambda \ldots\ldots \vdash \text{"I decide to pee"}$$

and this should result in an act of urination. This means that we should recognize and allow certain logic statements to trigger real-world actions (such real-world actions should not be confused with the inference engine's internal actions, the latter I call **control actions**).

We can put data into the KB to cause the inference engine to perform specific reasoning tasks. For example, "I am in the middle of a busy street, I feel like urinating, should I pee?"

In my model, the cognitive state vector can also store *internal* or *intermediate* variables, similar to the registers of a Turing machine [3].

So the RL can alter G to control the inference engine's performance. The state of the inference engine is G, and the reward is

[3] That is the similarity between my model and Neural Turing Machines. The Turing machine's basic operations are *read*, *write* and *move tape*, which are very primitive. But my model can perform logical operations which are significantly more advanced.