AGI architecture
white paper 2018-2

甄景贤 (King-Yin Yan)

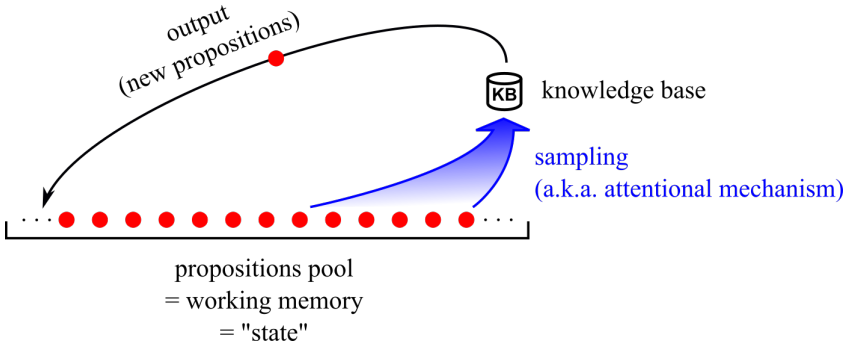
June 11, 2018

Abstract

结合逻辑 AI 和深度学习的最新尝试。重点是接受逻辑的命题结构 (propositional structure)，在这前提下，这是笔者能设计的最简单 architecture。或许其他读者可以改进它。

0 Basic architecture

首先了解一下经典 AI 的基本 architecture：

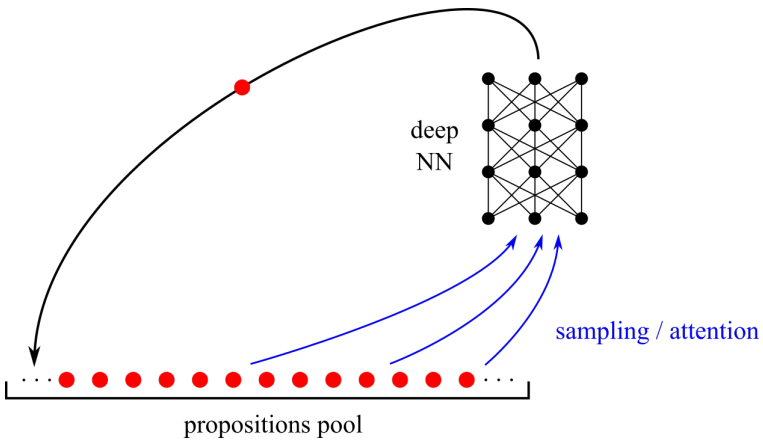


(1)

- working memory 是一些命题的集合
- KB 代表系统的所有**知识**的总和
- attention 就是现在深度学习中很火的概念，但在这架构下，它和 cognitive science 的「注意力」是完全一样的

关於这方面的理论可以在任何经典 AI 教科书找到。

我提出的 architecture 是用 deep neural network 代替 KB 的工作：



(2)

因为经典 AI 的「死穴」是 learning 太慢，它用的方法是 inductive logic learning，这领域的著名研究者包括 Stephen Muggleton, Luc de Raedt, Ivan Bratko, A. Srinivasan 等人（抱歉有很多遗漏），但基本上没有重大突破，导致经典 AI 的停滞。

神经网络为什么 powerful?

Deep learning 是现时最强的机器学习方法，理由是因为：它的输入和输出可以看成是 $\mathbb{R}^n \to \mathbb{R}^n$ 也可以看成是二元化了的 $\{0,1\}^n \to \{0,1\}^n$ 亦即是 n-dimension hypercube 的顶点（共 2^n 个）。神经网络里 **参数**的个数是：

$$L\, n^2$$

其中 L 是层数，假设每层都是 square matrix, fully connected. 但在输入和输出空间之间的函数的个数是：

$$2^n \to 2^n = (2^n)^{2^n} \tag{3}$$

这在计算机科学里基本上就是「无限」。换句话说，神经网络用相对地很少的参数，定义了一个函数家族，后者的数量是 **指数级**的。深度网络之所以能够这样，是因为随著层数的增加，整体函数的形状复杂性呈指数式上升（这复杂性可以用某种 topological degree 例如 Leray-Schauder degree 来描述）。深度学习是现时唯一有这种特性的学习机器。

有了问题的良好定义 (from classical AI)，也有了武器，那么问题是必然可以解决的，只差在有人要将二者结合起来。本文提出一个尝试。

逻辑的命题结构

在这 architecture 中，deep NN 的输入是一些命题 (encoded as vectors)，输出是一些新的命题（逻辑推导的结论）。

这 architecture 的重点是它应用了逻辑中的 **propositional structure**。在哲学逻辑中，**命题**的概念是非常基本的：命题就是能赋予真假值的东西。如果连命题结构也不接受，则整个经典逻辑也不用玩了。

命题结构的特点是 **可交互性** (commutativity)：

$$A \wedge B = B \wedge A \tag{4}$$

所以在 propositions pool 里，可以将命题任意排序，the order in which they are presented to the deep NN is unimportant. 我较早前提出了 symmetric neural network, 即 $F(a,b) = F(b,a)$ 的一种函数，但现在弃用这一方法，理由是因为 rule matching 是更重要的樽颈问题.....

1 Rule matching

「以 facts 找 rules」

在经典 AI 里有个很重要的问题：已知一些 facts，如何在 KB 中找到适合 apply 的 rules? 这个问题在 1970's 年代已有解答，即 Rete algorithm（拉丁文，*rete* 的意思是「网状」）。Rete 算法的重点是：「以 facts 找 rules，而不是用 rules 找 facts」，理由很简单：因为在 working memory 中 facts 的个数不算很大，但在 KB 中 rules 的个数是海量的。细节从略。

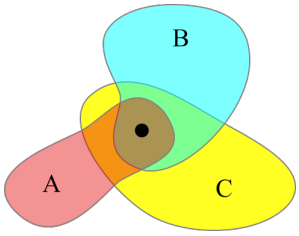
后来，Rete 被应用到 SOAR architecture，这是一个著名的 rules engine 兼 cognitive architecture，在 Carnegie-Mellon 大学发展的。但当然，它仍未能做到 strong AI，因为先前已经说过，logic AI 最严重的樽颈是 learning。

在我们的 architecture 里，learning 交给 deep NN 做，但 rule matching 仍然是一个很费时的动作。虽然说 recurrent neural network 有 "unreasonable effectiveness"，但既然 rule matching 这动作的结构已知，似乎不应该浪费资源让 RNN 「由零开始」学习它，否则学习太慢仍会失败。

我提出的解决方法，其实和 Rete 的思路是一样的：以 facts 找 rules。举例来说，有这样一条 rule：

$$A \wedge B \wedge C \to D \tag{5}$$

其中 A, B, C, D 是命题。解决办法是反过来将 $A \wedge B \wedge C$ 这个 conjunction 用一个 **点**表示，而命题 A, B, C 用一些 regions（区域）表示。如果 region A 包含 $\bullet$，表示 A 有份参与在这个 conjunction 里：

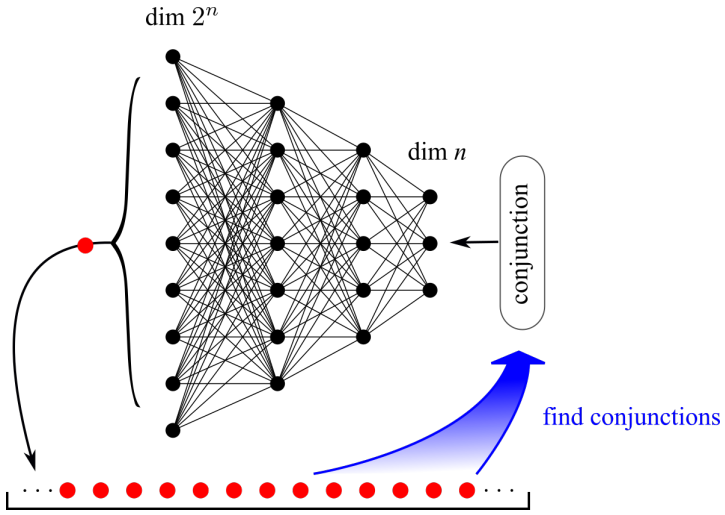


(6)

这样做的用意是：每个 proposition 是一个 region 也可以看成是一个 characteristic function，它输入某 **点**的位置，输出 true 或 false 表示这点属不属於该 region。换句话说它就像一个 "agent"，输入 conjunction，它回答它自己有没有参与这 conjunction，换句话说就是「以 fact 找 rule」。

具体做法是这样的：需要很多的点代表 conjunctions，不妨令 hypercube 的 2^n 个 vertices 做这些点。因为每个形如 (5) 的 rule 都有一个 conjunction，所以这些点的个数就是 KB 里 rules 的 **总数**。换句话说，deep NN 的作用是：已找到 rule，apply 那 rule。Rule matching 的工作分拆出来做，NN 只是学习和储存 KB 的 rules。

Architecture 变成这样（但仍有问题）：



(7)

NN 的输入端是 hypercube 的某个 vertex，这 vertex 代表某 conjunction。换句话说，输入是一个 binary vector of dimension n. 输出是一个 proposition，而根据我们的设定，proposition 就是 hypercube 的所有 vertices 的某 subset。换句话说，输出 $\in 2^{2^n}$。如果输出也是一支 binary vector，则其长度是 2^n。当 $n > O(10)$ 时，变得不切实际。举例来说，你或许会相信人类智能的 KB size $\approx 2^{100}$ 至 2^{1000} 或以上，但如果是 $2^{20} = 1048576$ 则似乎太少了。换句话说，NN 输出的 proposition representation 需要 **压缩**。

压缩的意思是：将长度为 2^n 的 vector，用较短的 vector 代表。这 encoding 可以是一个 $2^n \to \mathbf{2}$ 的近似函数（输入某个 vertex = conjunction，输出 yes / no）[1]，例如：

- 输出一串神经网络的 weights，这些 weights 再建立一个 **小神经网络** $F : 2^n \to \mathbf{2}$
  In other words, each proposition is a micro-neural network.
- 用 multi-variate discrete Fourier transform，truncate low-energy terms

这压缩的步骤很麻烦，第一种方法基本上就是「大 NN 输出小 NN」，其可行性未经证实。需要压缩的原因来自：proposition representation 的长度太长。解决的办法或许是将 proposition representation 的复杂性的一部分转移到 conjunction 的 representation 中去。但这样做有没有用？它只是将输出 representation 的复杂性转移到输入 representation 的复杂性。暂时想到这里。[2]

Stochastic local search for matching

馀下一个细节：根据以上的设计，给出一串命题，如何找成立的 conjunction(s)？算法的灵感来自 GSAT 和 WalkSAT，SAT 就是命题逻辑的 satisfiability，著名的 NP-complete 问题。SAT 和我们的 matching 问题非常相似，可能是同构的（?）

GSAT 的 G 是 greedy 的意思，WalkSAT 是指 random walk。实践证明，greedy + stochastic search 是对 SAT 问题非常有效的 heuristics.

如前所述，每个 proposition 是一个由 vertex → true / false 的 micro-function。可以将这输出固定在 true 位置，用 **反向传播**得出一些 satisfying vertex 的例子（不是唯一的）。用 list 记住这些 vertices.

然后或者可以用 genetic algorithm 进化这些 vertices，直到找到有 satisfied conjunction.

[1] 实际上，每个 conjunction 的子命题个数 k 不同，所以要使用 $2^n \to \mathbb{R}$ 或 $2^n \to [0,1]$ 的形式。而且，要避免同一个命题被 counted twice。

[2] 这个压缩的问题，在我提出的另一个 AGI architecture 里也遇到过，当时的目标是做到 Cartesian closed category 的 $X^X \cong X$. 方法是让神经网络的输出直接写入神经网络自己的 weights。但 weights 的个数显然 $\gg$ 于输出向量的维数，所以在那情况下亦需要压缩。我的感觉是，这压缩问题显示了 AGI 问题的樽颈所在。

2 Learning

学习算法很简单，基本上是 deep reinforcement learning，根据 Bellman update。训练时需要有智能的 teacher 或 training data 给予奖励或惩罚。这方面已经是一个有深入研究的课题，例如 UC Berkeley 公开了《深度强化学习》的网上课程。

Credit assignment 问题：奖与罚并不是做每一个逻辑推导即时获得的，它可能出现在一连串的 inference chain 之后，这 inference chain 其实就是强化学习里面的 **eligibility trace** 概念。换句话说，整条 **逻辑推导链**，同时获得奖励和惩罚（也就是将那些「有责任的」weights 增强 / 减弱）。All the weights in the NN can be re-normalized periodically.

A note about natural language

将 internal representation 转换成自然语言，只需额外的 logical inference，但在学习 internal representation 和知识的同时，要学习这 language generation 的能力可谓难上加难。因为奖与罚的时候并不知道是内在知识有问题还是语言的表达有问题。所以建议在开始训练时不要同时学习知识和语言表达。然而，人类在讲述自己没有信心的内容时，也会有「结结巴巴」的现象，这似乎表示，内在知识和语言表达的 inference chains 同时受奖与罚是「正常」的 ☺

3 Conclusion

目前最不满意的地方是 proposition representation 所需的压缩。因为命题是一些 micro-神经网络，改变一个 weight 的作用很微小，有 vanishing gradient 的忧虑。但笔者亦庆幸越来越看到 AGI 接近实践的地步。

Acknowledgements

将神经网络和 von Neumann architecture style 的程式混合，这种做法虽然缺乏美感但可能是很有效的。我在读 Andrew Ng *et al* 提出的 recursive neural network（有别於 recurrent neural network）时看到这种例子。