I recall with amusement Eckert's reaction to the impending visit. He said that he could tell whether von Neumann was really a genius by his first question. If this was about the logical structure of the machine, he would believe in von Neumann, otherwise not. Of course this was von Neumann's first query.

— Goldstine
The Computer from Pascal to von Neumann [1972]

Genifer 4.2 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 upon 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 **deduction** and **inductive learning** over sentences.

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?

I don't have a solution yet, just a few points:

2.1 Logic as an external teacher

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

Think carefully, there is no "bad recursion" here. The inference engine can be regarded as a teacher. It suggests to the RL some actions. The RL can accept or deny to perform those actions. If it does accept, the RL observes how good is the outcome. Then it can *adjust* or *fine tune* the teacher accordingly. In the entire process above, the teacher is an *external* entity manipulated by the RL.

2.2 How to invoke the inference engine?

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?"

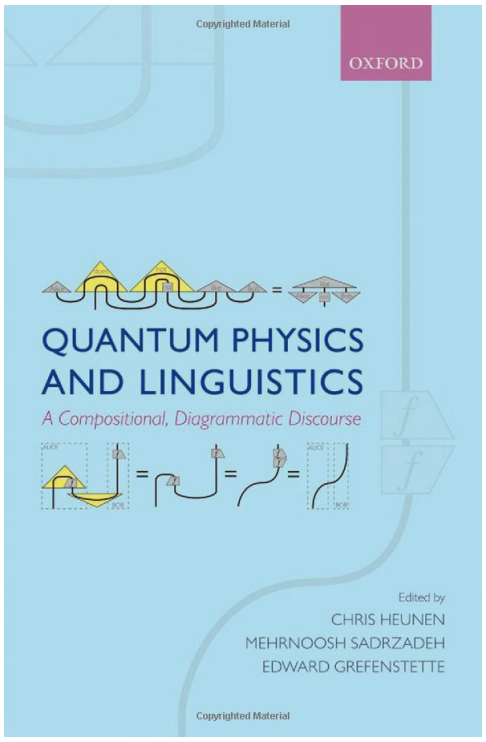
In my model, the cognitive state vector can also store *internal* or *intermediate* variables, similar to the registers of a Turing machine [3].

So the RL can alter G to control the inference engine's performance (these are the control actions from the RL's perspective). The state of the inference engine is G, and the rewards are observed from how well the logic engine performs as a teacher.

[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. Whereas my model can perform logical operations which are much more advanced.

3 Tensor product representations

- The most basic is the relation $a\,R\,b$ with 3 elements
- Smolensky's method, which can represent any recursive tree structure
- In a fairly new research direction, people realize that linguistic structure is identical to abstract algebraic structure, so attention is shifted to studying abstract categories. For example this book:



The most general categories are **monoidal categories**, also called **tensor categories**. The tensor product of vector spaces is an example in this category. Essentially we are using tensor products to represent sentences; Category theory provides a more general and abstract framework.

3.1 Neural Tensor Network

Andrew Ng is from Hong Kong, he taught a famous machine learning course in Coursera. Recently he left Stanford to join Baidu (the Chinese search engine).

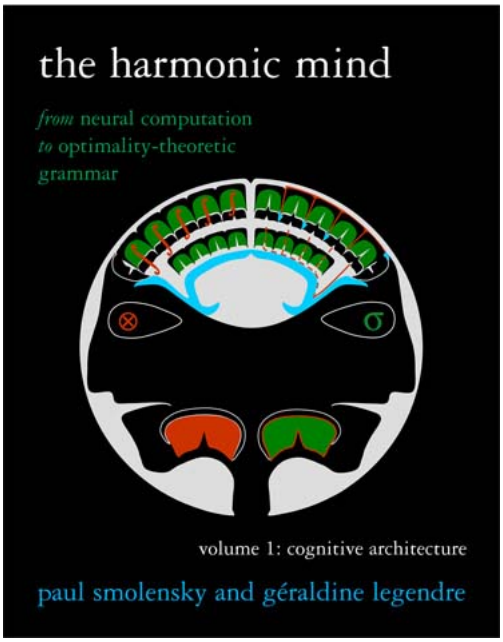
Socher, Chen, Manning, and Ng proposed the NTN model in 2013 [4]:

$$\top(a\,R\,b) = u^T \boxed{\int}(a^T\,U\,b + W\begin{pmatrix}a\\b\end{pmatrix} + c)$$

$\top(a\,R\,b)$ is the **strength** of the relation $a\,R\,b$.
U is a tensor, it takes two vectors a, b and gives a third vector $a^T\,U\,b$ (this obeys the bi-linear property of tensors).
W is a matrix, $W\binom{a}{b}$ is a layer of traditional neural network, $\boxed{\int}$ is a non-linear threshold function.
u^T is a single neuron, it's purpose is to add up the different components, giving a number.

3.2 Paul Smolensky

Paul Smolensky's book is *The Harmonic Mind* (2006) [3]. Volume 1 is AI theory, Volume 2 is linguistic theory.



He suggests to use tensors to represent relations; Generally speaking:

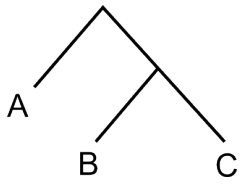
$$\begin{aligned} father(john, pete) &\Leftrightarrow father \otimes john \otimes pete \\ Var_1 : val_1, Var_2 : val_2 &\Leftrightarrow Var_1 \otimes val_1 + Var_2 \otimes val_2 \end{aligned}$$

The second line means: $variable_1$ takes value $value_1$, $variable_2$ takes value $value_2$.

He also introduced the notions of "roles" and "fillers": $variable_i$ is a role, $value_i$ is a filler. Using this method to handle variable binding, the advantage is that it can avoid the $A \cdot B = B \cdot A$ problem.

For example, in natural language we could have a sentence structure like this:

$s =$

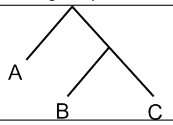


r_0 and r_1 are respectively the tree's left and right children's roles. Expressed as a tensor product:

$$\begin{aligned} s &= A \otimes r_0 + q \otimes r_1 \\ q &= B \otimes r_0 + C \otimes r_1 \\ \therefore\ s &= A \otimes r_0 + B \otimes r_0 \otimes r_1 + C \otimes r_1 \otimes r_1 \end{aligned}$$

The good thing about tensor products is that, r_0 and $r_0 \otimes r_1$ are different elements, they would not "clash". A drawback is that as the length of the tensor product grows, its dimension also grows very fast.

The following table summarizes Smolensky's approach:

	Example	Vector representation
Set element	$\{c_1, c_2\}$	$c_1 + c_2$
Filler-role binding	$AB = \{A\backslash r_1, B\backslash r_2\}$	$A \otimes r_0 + B \otimes r_1$
Recursive structure		$A \otimes r_0 + [BC] \otimes r_1$

As I have explained on my blog before, [4] tensors are the universal form of all bi-linear forms. If we have 2 vector spaces U and V,

$$\begin{aligned} T: &\quad U \otimes V \to W \\ t: &\quad u \otimes v \mapsto w \end{aligned}$$

and 2 sets of vectors in U, V that are linearly independent, $\{u_1, ..., u_m\}$ and $\{v_1, ..., v_n\}$, then their product $\{u_i \otimes v_i\}$ in W would still be linearly-independent. This is a *defining property* of tensor products, in other words:

$$\text{linear independent set} \otimes \text{linear independent set} \mapsto \text{linear independent set}$$

and this is precisely what we need, because under the "scalar $=$ logic truth value" interpretation, this is precisely the condition where propositions would not "clash".

Antony Browne & Ron Sun's earlier paper "Connectionist Inference Models" (2001) [1] talks more about various neural-symbolic integration schemes.

3.3 Coecke-Sadrzadeh-Clark

This representation is proposed by Coecke, Sadrzadeh, Clark in 2010 [2].

Firstly it is based on **categorial grammar**, which is a grammar with "left" and "right" multiplications. For example, in the sentence ``*John likes Mary*", the type of ``*likes*" is:

$$likes := (S \backslash NP)/NP$$

That means when the word ``*likes*" is multiplied by a noun phrase on the left, and another noun phrase on the right, we obtain a sentence S. This kind of grammar is pioneered by Lambek in 1958.

Translating to tensor products, ``*likes*" is represented as a vector:

$$\overrightarrow{likes} \in N \otimes S \otimes N$$

which makes sense, because this tensor product space contains as elements the multi-linear functions:

$$f : N \times N \to S.$$

[4] blog post

References

[1] Browne and Sun. Connectionist inference models. *Neural networks 14*, 2001.

[2] Coecke, Sadrzadeh, and Clark. Mathematical foundations for distributed compostional model of meaning. *Linguistic Analysis 36, 345-384*, 2010.

[3] Smolensky and Legendre. *The harmonic mind, vol 1: cognitive architecture*. MIT Press, 2006.

[4] Socher, Chen, Manning, and Ng. Reasoning with neural tensor networks for knowledge base completion. *NIPS*, 2013.