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)

June 7, 2015

1 New insights

最近在香港和朋友 Dr 陈启良和 Dr 譚志斌 (?) 谈过之后：

1.1 命题空间的 dimension

在知识空间中，如果 v_1, v_2, v_3 是三个互不相关的命题，那么它们的线性相关 $a_1 v_1 + a_2 v_2 = a_3 v_3$ 是不容许的，否则会出现谬误。结论是：要么知识空间的 dimension 等於命题的总数（那会很大，可能无限大），要么放弃 a_i 是真假值的做法。

人的生字 (vocabulary) 的个数大约在 3,000 至 10,000's 之间。暂时假设我们可以用 dimensionality reduction 把原始概念的个数缩小到 3000，那么概念空间 C 的维数就是 3000。

如果只考虑像 $a\,R\,b$ 那样的关系，则命题空间是 $C \times C \times C$，维数是 $3000^3 = 9G$。这似乎是当代计算机勉强可以处理的。

1.2 「连续性」

第二个问题是「连续」指的是什么。例如命题 $P_1 =$「小明爱小娟」，$P_2 =$「小强爱踢足球」，那么由 P_1 变化到 P_2 必然会有 discrete jump，那是无可置疑的。除非我们重新定义命题是像 $k_1 P_1 + k_2 P_2$ 那样的东西，才可以有「连续变化」；这一点需要再加以精确化。

我发觉「连续空间」不是一个严谨的术语，因为在 discrete 空间中也可以定义"continuous" 这概念，例如在 computer science 的 functional programming 里有 domain theory、denotational semantics 等，常常谈到离散空间中的连续函数。而且「可微分」的概念也有一些 generalizations，例如 Fréchet derivatives（在泛函空间中）。

2 Neural Tensor Network

Andrew Ng 是香港人，Coursera 的 machine learning 教授，他在 Stanford，最近加入了百度。他和合作者提出了 NTN 模型：

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

$\top(a\,R\,b)$ 表示 $a\,R\,b$ 这个关系的**强度**。
U 是一个 tensor，取两个向量 a, b 给出第三个向量 $a^T\,U\,b$（它服从张量的双线性性质）。
W 是一个矩阵，$W\binom{a}{b}$ 是一层传统的神经网络，$\boxed{\int}$ 是一个非线性函数。
u^T 是一粒神经元，作用是把各个分量加起来，得出一个数。

3 Paul Smolensky's tensor representation

Paul Smolensky 的书是《The harmonic mind》(2006)，第一卷是数学理论，第二卷是语言学理论。
他提出用张量来表示关系：

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

第二句的意思是，variable 1 的值是 value 1，variable 2 的值是 value 2。
我在博客上解释过，tensor 是所有 bi-linear forms 的 universal form。这个表示法的意义是……？

4 Metric embedding

最近有一本新书：[Mikhail Ostrovskii 2013] *Metric embeddings: bi-Lipschitz and coarse embeddings into Banach spaces*，它似乎正正讲述了我们将 logic 嵌入到「连续空间」的问题。

What is bi-Lipschitz condition? A map $f : X \to Y$ is called a *C-bi-Lipschitz embedding* if there exists $r > 0$ such that

$$\forall u, v \in X, \quad r\, d(u, v) \leq d(f(u), f(v)) \leq rC\, d(u, v)$$

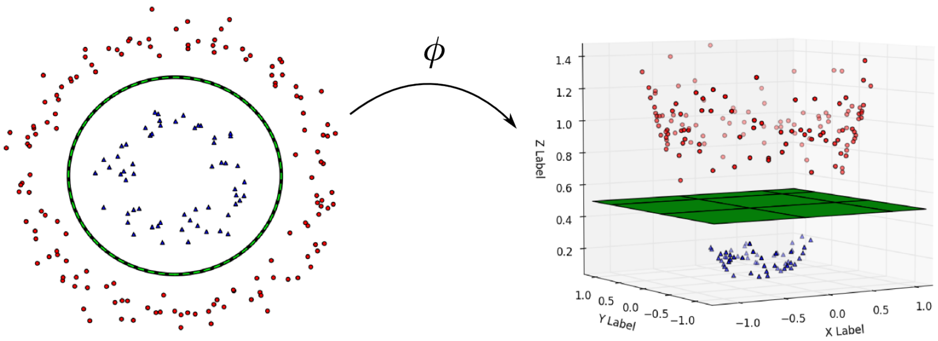
for some $C < \infty$. The smallest constant C for which there exists $r > 0$ such that the above is satisfied is called the *distortion* of the map f. 这似乎定义了「远的保持远、近的保持近」的意思。

5 SVMs and the kernel trick

印象中 **SVM** (support vector machine) 似乎可以将非凸的问题转化成凸优化的问题；可不可以将这个想法应用到我们的问题上呢？

SVM 的算法，首先用 kernel trick 将 data points 转换到高维空间，然后在高维空间中进行线性的 hyperplane 分割。

所谓 **kernel trick** 是指：给定一个 inner product，它暗含了一个到高维空间的转换 ϕ （但这个转换不需要真的计出来）。



而，在高维空间中用线性分割，那 error term 是一些正交距离，所以是 **quadratic form**，所以这问题是**凸优化**。

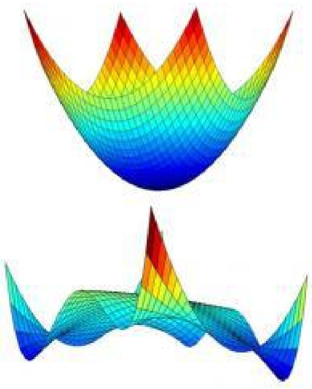
我们的问题是性质不同的，但两者似乎有些相似...

6 Old stuff

For readers curious about what I'm doing in Genifer 4, it is really not a big deal, basically I am trying to transfer logic to a **continuous setting**, so that I can apply iterative approximation techniques, such as gradient descent.

In the past year, after a Herculean effort, I barely succeeded in making logic continuous (and even this part is fraught with problems), but at least the problem of inductive learning seem to be transformed into one of mathematical continuous optimization.

And then I discover, the most efficient optimization methods are based on convex technologies. As in the following figure, the fitness landscape above is convex, the one below is non-convex: [1]



If we don't assume convexity, we have to fall back on techniques like genetic algorithms, branch-and-bound, etc. Those techniques I have already known for a long time in computer science, so I disappointedly felt my efforts have been in vain ☹

The latter algorithms could still be fast, but they are *heuristics* which means there are no speed gaurantees. Because the complexity of inductive learning is extremely high, do we dare to try the heuristics? For instance, would you risk investing $10,000's to bet on Genifer 3.0?

Mathematics is truly wonderful, it progresses constantly, but sadly it progresses very slowly. Right now I see the frontier is to extend convex optimization, but I

[1] from Charles H Martin's blog.

have not seen any great breakthroughs (though I am just a novice and have not surveyed the entire field.) If there is indeed such a breakthrough, it would really be ground-breaking, but that is not something within the capacity of ordinary folks, not even a lot of professional mathematicians.

Right now what we could do is probably:

- find a problem that is non-convex but which could still be solved efficiently, then transform our problem to that problem.
- change the representation of the original problem; this requires innovative thinking. Or even turn the optimization problem into an equation solving problem, etc.
- then see if the fitness landscape might have some better characteristics; This requires experimentation, plotting those error surfaces.
- give up convexity, we could still use other global optimization techniques: genetic algorithms, branch-and-bound, convex-concave / DC programming, back-propagation for neural networks, simulated annealing, etc. As I said, some of these techniques do not need continuity, we can go back to Genifer 3.

In the remaining article, I will talk about my attempt at making logic continuous, which might still find a use some day (but then again could also be a dead end).

7 Making logic continuous

Basically I try to establish this correspondence：

KB (knowledge base)	$\Leftrightarrow$	vector space
logic rules	$\Leftrightarrow$	non-linear operators
learning logic rules	$\Leftrightarrow$	searching for operators in functional space

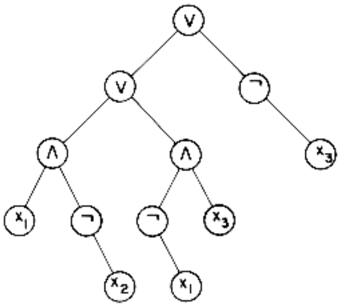
Even after the transition to continuous there would still be the non-convexity problem, we don't bother now. Just making it continuous is hard enough, because the structure of classical logic is very complicated.

Many theories of logic are expressed in mathematical symbols, but they actually do not have strong connections with other existing branches of mathematics, so they are no more than a pile of symbols. To be truly useful, we have to build *bridges* to other branches of mathematics. I find that it is easiest to do via (abstract) algebra, as the algebraic formulation allows to easily see similar patterns in other areas.

8 What is logic?

Logic can be divided into **propositional logic** and **predicate logic**, the former is isomorphic to Boolean algebra and is easier to understand and mathematically simpler; The latter is difficult to model mathematically, for example it requires using the *cylindric algebra* invented by Tarski, that is little known outside of the logic circle. I am trying an alternative route based on **combinatory logic** and **relation algebra**. [2]

Firstly, it is obvious that logic formulas can be expressed as tree structures, and trees in turn can be easily expressed in sum-product algebraic form. For example (this is even a binary tree):



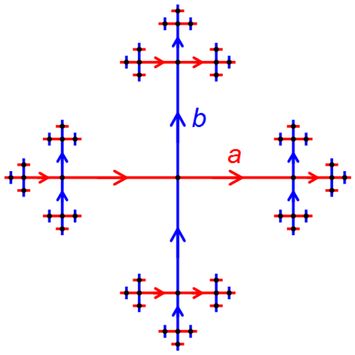
However there is difficulty in expressing this in vector space. Vectors can be added but there is no obviously good candidate for vector multiplication. For example, if two unrelated formulas such as john·loves·mary and paul·loves·playing·basketball end up "crashing" into each other in **conceptual space**, or being in proximity without any real reason, would be troublesome for the system.

[2] Combinatory logic changes the predicate logic form $R(a, b)$ to the algebraic form $R\,a\,b$ or $a\,R\,b$, for example $loves(john, mary)$ is changed to $john \cdot loves \cdot mary$. Relation algebra is a special form of combinatory logic.

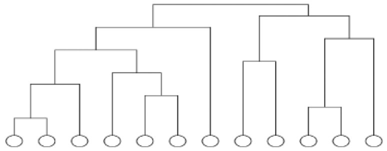
9 Cayley graph

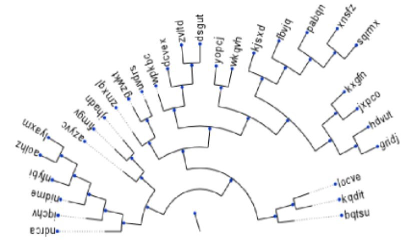
Earlier I tried an idea using the Cayley graph, but I found that it has many difficulties.

The idea of Cayley graph is very simple, for example the following diagram is the free group generated by 2 elements $\{a, b\}$, the products are linked by edges:



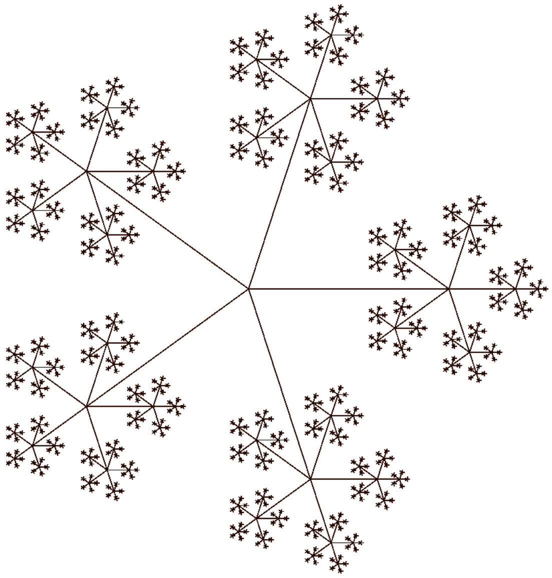
There is also the notion of **ontology** in AI, for example: cats $\subseteq$ animals. Ontology can be expressed as a hierarchical cluster, which can be shaped as a circle or ball:





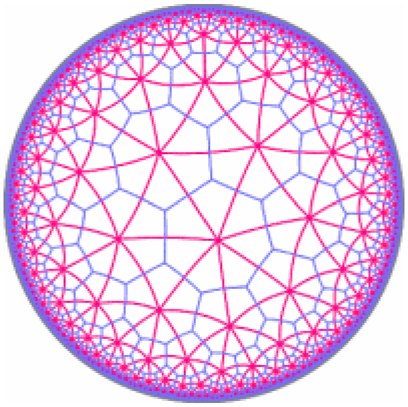
I attempted to combine the ideas of Cayley graph and ontology: In the Cayley graph each node is a **concept** (eg love or football), and every concept can find its location in the ontology ball. But the ontology ball itself is hierachically structured, and it seems difficult if I try to "embed" it into Cayley graph nodes.

Moreover, as the Cayley graph's number of branching increases, it looks more and more "unnatural"; for example this is the case for just $n = 5$:



The reason is that Cayley graphs are actually fractal structures! I have wished to achieve the "continuous" effect, but in the Cayley graph jumping from node to node is **discontinous**.

Nevertheless, Cayley graph has another advantage, that it can be embedded into the hyperbolic disc, which possesses the metric of **hyperbolic geometry** (as seen in many of M C Escher's artworks, ie, the metric shrinks as we move closer to the edge of the disc):



In hyperbolic geometry straight lines turns into arcs, and in artificial neural networks the discriminant function also contains the linear form $Y = \boxed{\int}(\sum W_i X_i)$, so perhaps we can create a kind of "hyperbolic neuron"? Unfortunately due to the discontinuity issue, this idea is abandoned for now.

10 Distributive vector representation (DVR)

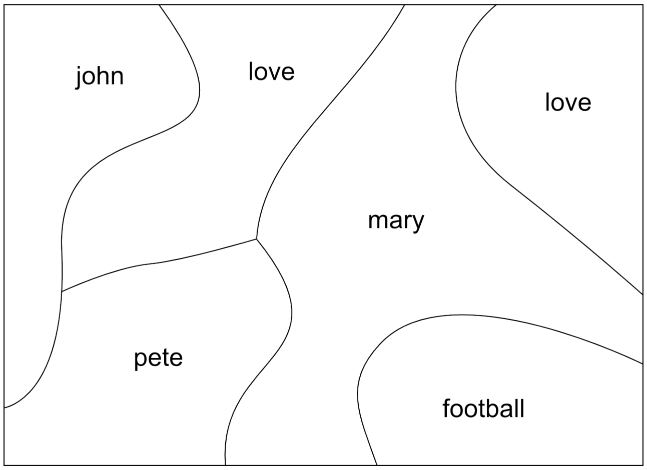
This is a better approach, and seems to solve the **continuity** issue.

Under DVR, knowledge is represented by a very long vector:

$$K = \mathbf{v} = (v_1, v_2, ..., v_n)$$

as is quite common in neural networks. This means that each **concept** is not represented by a single neuron ($=$ a single v_i), but rather by an entire layer of neurons ($= \mathbf{v}$). Under this representation, a concept is one or more **regions** in space, and the regions can be disjoint.

For example, when the dimension of the vector space is $n = 2$, we can have an example like this:



Vectors can be **added**, where addition means **superposition**, which notion came also from neural networks. For example, the two formulas: john·loves·mary and pete·loves·football can be superimposed on K, meaning that both sentences are simultaneously true. As long as concepts do not ”crash” into each other, this is feasible.

But there is still another problem: when a formula is repeated, its truth value would not be doubled (as when I repeat ”john loves mary” twice); In other words, $a \wedge a = a$, but in vector addition this will be doubled, unless we use vector space over a special **field** where $1 + 1 = 1$ or something[3]. This is a detail to be treated later.

For now we skip the issue of **multiplication**. In §7 we discuss how to handle products.

11 Deduction in logic

The most basic operation in logic is to **deduce** a new formula from the KB (**knowledge base**). The “KB” terminology came from classical AI, we denote it simply as K in the new formulation.

The basic operation of deduction is to *apply rules to facts*. Deduction can be represented by the operator T:

$$T : K \mapsto K.$$

But T is **non-linear**.

T corresponds to one logical rule. A classic example is the definition of “granddaddy”:
    granddaddy(X,Z) $\leftarrow$ daddy(X,Y) $\wedge$ daddy(Y,Z)

To perform this deduction step, we need to perform **pattern matching** first, and then the rule can be **applied** to matched facts.

So we have this correspondence:

deduction	$\Leftrightarrow$	apply operators
pattern matching	$\Leftrightarrow$	filter: $K \to K$

[3] One may use the "tropical" semi-ring, or min-plus algebra.

For example, assume we know the following facts:

daddy(john, pete)
daddy(pete, paul)

and we need to find the substitution { john/X, pete/Y, paul/Z }, in order to apply the above rule. Finding this substitution is called pattern matching; the classic **unification algorithm** accomplishes that.

That is the reason why T is **non-linear**: For various K's, the action of T is usually 0, unless pattern matching succeeds, then T's action would be non-zero. (Of course, we would make T smooth, so the "$= 0$" above would be replaced with "close to 0".)

In an AI system there would be many rules $T_1, T_2, ..., T_n$. Applying T_i on the KB, superimposing them together, that would be a **single step** of deduction. But because T_i is non-linear, the summation and T_i do not **commute**.

A single step of deduction is:

$$K' = \sum T_i(K)$$

and the **full logical consequence** of the KB is:

$$\begin{aligned} K^\infty &= \text{the above step repeated infinitely} \\ &= (\sum T_i)^\infty(K). \end{aligned}$$

12 Learning

The learning algorithm is the bottleneck of AI. If we have a good inductive learner, other details are just details (say we can solve with **reinforcement learning**).

The special thing about logic-based learning is that it has a **doubly iterative** structure that consists of **deduction** and **induction**. Induction needs to call deduction, but deduction is itself a process of high complexity.

In other words, we first use deduction to get K^∞ (this requires iteration of the operators):

$$K^\infty = (\sum T_i)^\infty(K^0).$$

And then we compare the result K^∞ with the ideal K^*, get the error, and our objective is to find a set of T_i's to minimize the error. The T_i's live in a space of non-linear operators.

Logically what we aim at is:

$$K^0 \cup \{T_i\} \models E$$

where E is the new examples or "new experience" that needs to be explained. The idea answer is $K^* = K^0 \cup E$.

The error $\mathcal{E}$ is the difference between the deduced K^∞ and the ideal K^*:

$$\mathcal{E} = K^\infty - K^* = (\sum T_i)^\infty K^0 - (K^0 + E).$$

If the T_i's are differentiable, the gradient $\partial \mathcal{E} / \partial T_i$ exists, and we can use gradient descent.

Compare this with ordinary optimization techniques, such as the Newton-Raphson, which is just the iteration of an operator:

$$x^\infty = T^\infty x^0$$

and x^∞ would be the required answer.

Compare with the traditional back-propagation for neural networks. The back-prop algorithm is:

(a single neuron)	$y_j = \boxed{\int}(\sum W_{ij} x_i)$
(multi-layer structure)	output $= y_0 \circ y_1 \circ ... \circ y_n$(input)
(update rule)	$\mathbf{W}' = \mathbf{W} + \alpha \, \partial \mathcal{E} / \partial \mathbf{W}$

In comparison:

- In back-prop the operators $y_0 \circ y_1 \circ ... \circ y_n$ come from the many layers of neurons, so back-prop can be regarded as the propagation of errors in **space**.
- Our iteration occurs in **time**, but that seems to make no essential difference.

My current diagnosis of the problem is: The space of K may be too large (though its dimension may not be so), because it contains all logic formulas, and if there is no approximation, this problem is exactly the same as the original logic learning problem, and there would not be any improvement, it would at best be a relaxation of the original. We should exploit function approximation to achieve some **generalization**.

But logic has its own generalization structure. The purpose of generalization is **compression**; we could as well say that learning is **compression**. Logic-based learning itself is a compression scheme of very high complexity (which is why it's so slow). The $A \subseteq B$ subsumption relation establishes a hierarchical classification structure, this classification amounts to *taking the logarithm of the size of the knoweldge base*, just like we do in **binary search**. However, due to the doubly exponential complexity of logic learning, perhaps this trick alone is not enough?

Stephen Muggleton, a leading researcher of inductive logic learning, wrote in a 2002 paper on applying **generic algorithms** to logic learning:

> *"The usual way for evaluating a hypothesis in first-order concept learning systems is to repeatedly call a theorem prover (eg Prolog interpreter) on training examples to find out positive and negative coverage of the hypothesis. This step is known to be a complex and time-consuming task in first-order concept learning. In the case of genetic-based systems this situation is even worse, because we need to evaluate a population of hypothesis in each generation. This problem is another important difficulty when applying GAs in first-order concept learning."*

13 Product structure

I'm still unsure how to represent products, such as john·loves·mary.

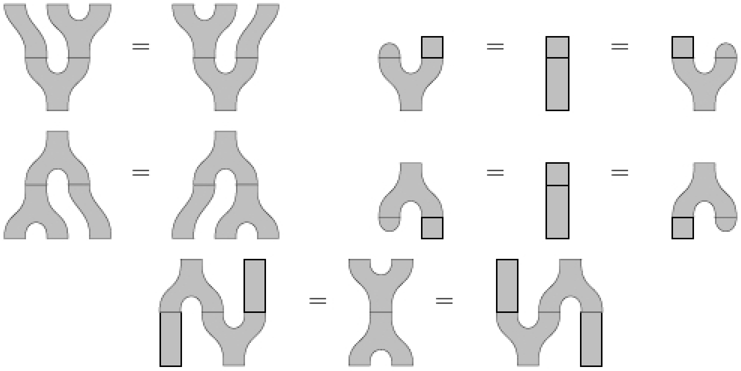
One proposal is from Geoffrey Hinton: Each concept is a **matrix**; If the concepts a and b are related by $a\,R\,b$, then their representing matrices would obey $AR = B$, and the positions of such matrices in matrix space have to be learned. This method seems a bit troublesome, as we need to perform the matrix multiplications to obtain which elements follow which.

A second proposal uses the **tensor product**, but the tensor product space has high dimension that grows with the product's length. Paul Smolensky's book *The Harmonic Mind* (2006) volume 1 describes his distributive tensor product representation (which I will explain when I have time).

A third proposal is from the collection of papers *Quantum Physics and Linguistics* (2013). There they use **monoidal categories**, a kind of categories that have products as well as compositions. Its definition has:

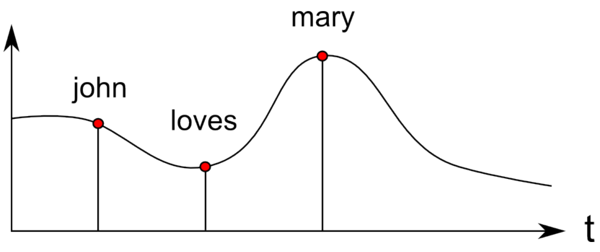
- objects: $X, Y, \ldots$
- morphisms: $f : X \to Y$
- composition: $f \circ g : X \to Z$
- grouping objects into: $X \otimes Y$
- a special object for the "empty system": I
- parallel composition: for $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$
  $f_1 \otimes f_2 : X_1 \otimes X_2 \to Y_1 \otimes Y_2$

Its algebraic operations can be represented by certain **braid** diagrams:



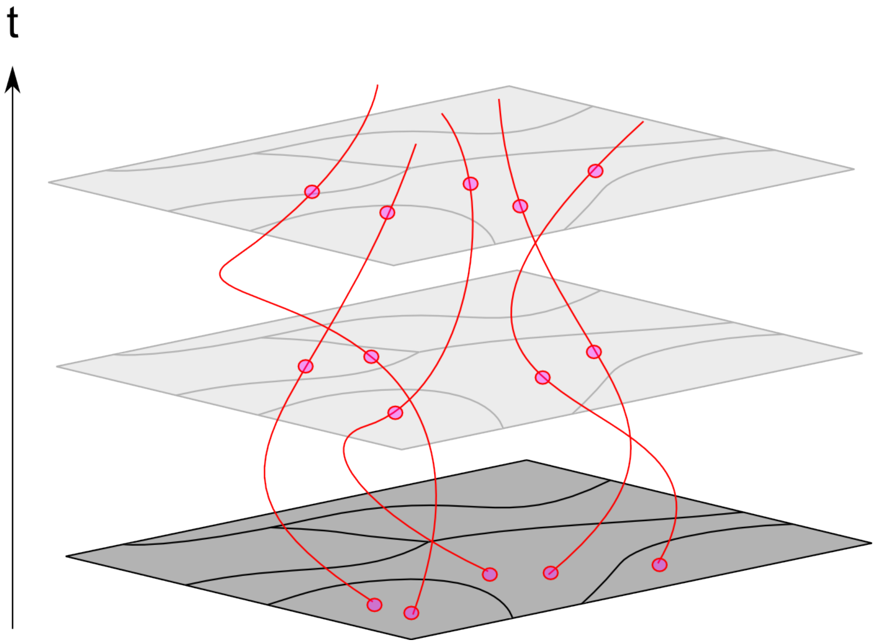
This seems to be a very general structure (including tensor products?) A main theme of the book is that these categories are capable of covering all syntactic structures in natural langauge. I don't have the time to study it in depth yet.

Another proposal is my invention. The idea is: we can treat the following function as a continous **time series**:



A product is a discrete sequence, we can make it continuous as a **curve** parametrized by t.

These curves in the space of $K \times t$ may look like this:



The curves can be represented by splines, that are **polynomials**. Then perhaps we could apply tools in **algebraic geometry** (a highly sophisticated branch of modern mathematics). A polynomial can be represented by its coefficients, ie, a list of numbers, which is very convenient. And the map between polynomials are also polynomials, forming a **dual space**, that reminds of the situation in logic, where formulas can apply to other formulas.

14 Formal power series

A power series has the form:

$$\sum k_i a^i$$

but if the a_i's are not ordinary numbers, the summation cannot be performed with ordinary addition, so we call it a **formal series**.

Abstractly speaking, let A be a set of **atomic concepts**, such as

$$\{\,\text{love}, \text{hate}, \text{men}, \text{women}, ...\}$$

A^* is the set of all sentences that can be combined from these concepts. Multiply these sentences with some coefficients, for example:

0.8 john·loves·mary $+$
0.3 mary·loves·john $+$
0.9 mary·loves·pete $+$
...

The coefficients $k_i \in K$ and K can be an arbitrary **semi-ring**. We can regard K as the semi-ring of **logical truth values**.

Denote all formal series like the above as $K\langle\langle A\rangle\rangle$.

Formal series are closely related to **finite state machines** (FSMs): FSMs can recognize **formal languages**. A formal language L can be any subset of A^*. If L contains a sentence, give it a coefficient 1, otherwise coefficient 0. So we obtain a formal series representing that formal language.

Not all formal languages can be recognized by FSMs.

If recognizable, then for each letter (our "atomic concept") in this language, we can construct a $M_{n\times n}$ matrix, whose M_{pq} entry depends on whether there is a state transition from p to q (1 if yes, 0 if no).

The multiplication of these matrices preserves the monoid multiplication rule, for example: $a \cdot b \cdot c \times d \cdot e \cdot f = a \cdot b \cdot c \cdot d \cdot e \cdot f$.

In **representation theory**, we say this is a **matrix representation** of the monoid.

According to representation theory, multiplying a ring K by a monoid A gives us a K-module, so $K\langle\langle A\rangle\rangle$ can be regarded as a (left) K-module.

Existence of the module implies existence of a representation. (The famous Jewish woman mathematician Emmy Noether pioneered the use of modules to study representations.)

And we have just seen, FSM recognizable ⇒ can build matrix representation.

So, ”FSM-recognizable” is equivalent to ”matrix representation exists”, and this in turn is equivalent to ”existence of a certain finitely generated K-submodule, that contains $S \in K\langle\langle A\rangle\rangle$ (where S is the formal series of the formal language).

Part of the above is from *Encyclopedia of mathematics: Noncommutative rational series with applications* (2011 Cambridge Univ Press).

facts (propositions)	⇔	monoid words
KB	⇔	formal series
rules	⇔	non-linear operators acting on formal series

15 Memory registers

Lastly, I have to mention this: The structure of classical logic is just too complex, our troubles are still not finished...

In the Genifer 3 white paper, I explained that we need to introduce a logic with “memory registers”. In other words, we need to add some logical “actions”, that allow to read and write those registers.

memory register	⇔	vector space $K \times t$
actions	⇔	some meta-operations (?) over registers

This is just too much trouble, perhaps I'll stop here and update next time, zàijiàn!