Structuralism

Linas Vepstas*

Draft of 20 June 2025; incomplete.

*BrainyBlaze

Abstract

A summary review of a structuralist, agent approach to machine consciousness.

Preface / Apologia

I will attempt to write as simply as possible. This means to some of the below might seem simplistic, obvious and well–known. If so, then its a mis–perception: attempting to turn the ideas below into working software is a half–finished project, with many difficult technical issues. These are accompanied by even harder questions of system architecture, design choice, and, at the most base level, the structural representation of what superficially seem like simple, easy obvious ideas.

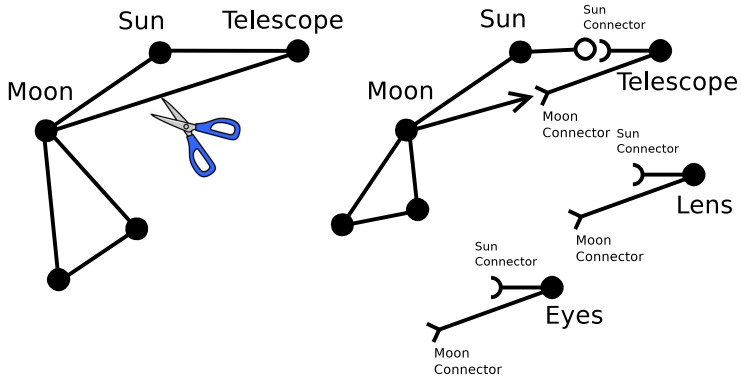
Anyone familiar with philosophy or sociology will immediately recognize a broad range of familiar concepts: the signifier and signifcand, for example. Anyone familiar with mathematics will recognize a different set of familiar topics, such as monoidal categories and fragments of linear logic (the internal logic of dagger–symmetric tensor categories). These resemblances are not accidental; this text is being written in such a way as to evoke those ideas, and you the reader should think of them. There is much to be said and the author cannot say it all, so it is up to the reader to elaborate.

XXX This is an unfinished draft. It's being sent out in its current form, despite not being clean and finished, so as to get it out quickly. This part covers well–established results and is straight–forward. A part two, covering more speculative ideas ("it's agents all the way down") is to be written; since what is here is already, the commentary on sensori–motor agents will be part two.

XXX This has not been proof–read and there will be errors and mistakes. I'm trying to get this out fast.

1 Introduction

It appears that the world is made of things and the relationship between things. These can be represented as graphs, with objects as vertices, and relations as edges. Graphs are decomposable into parts; a proper decomposition preserves the relationships by marking edges, so that the disassembled parts can be reassembled appropriately. The marked cut edges imply that the graph parts resemble jigsaw puzzle pieces:



The task of intelligence, of an intelligent agent, is to discover these relationships. The agent can do so by means of perception; at the lowest levels these are concrete: eyes, ears, etc. but can also be abstract. For example, knowing some facts about geography and politics may still leave one ignorant of the relationship between the two, and may require a significant intellectual effort to understand this relationship. This is again an act of perception: the pattern of relationships between politics and geography has to be perceived, although the organ of perception is no longer as simple as "eyes and ears", but is also abstract.

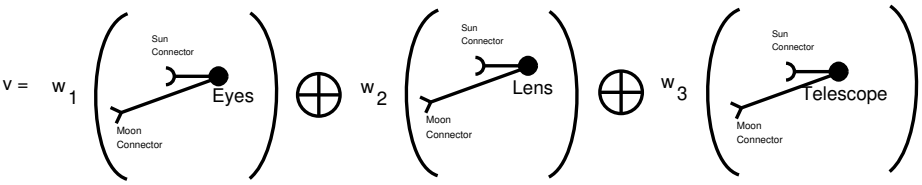
An intelligent agent perceives and models the external universe, and constructs a provisional model of "what is outside". This model conventionally called "the world model", is necessarily incomplete, hypothetical, and riddled with errors. There may be more than "one" model that is active at a time, as one needs to entertain possibilities: "it could be this or it could be that". This raises a minor philosophical question: is this "one" world–model, or "two"? After all, hypothetical relationships can be represented in a variety of formal logics; the "two" possibilities can be collapsed down to one grand–total.

The following is split into several sections.

- The first section reviews the concept of vector representations and their relationship to graphical models.
- The second section reviews

2 Vector representations

The multiple possibilities have a natural representation as a vector; such a vector representation make natural contact with the conventional frameworks of artificial neural nets.



The above is meant to be a diagrammatic representation of a a vctor with three basis elements (eyes, lens, telescope) with three (real number) weights w_1, w_2, w_3. The above is drawn diagrammatically; other common notations used in the literature are

$$\begin{aligned} v &= w_1 |\psi_1\rangle \oplus w_2 |\psi_2\rangle \oplus w_3 |\psi_3\rangle \\ &= w_1 |\psi_1\rangle + w_2 |\psi_2\rangle + w_3 |\psi_3\rangle \\ &= w_1 \hat{e}_1 + w_2 \hat{e}_2 + w_3 \hat{e}_3 \\ &= w_1 e_1 + w_2 e_2 + w_3 e_3 \\ &= [w_1, w_2, w_3] \\ &= X_1 p\left(X | x = X_1\right) + X_2 p\left(X | x = X_2\right) + X_3 p\left(X | x = X_3\right) \end{aligned}$$

All of these mean exactly the same thing; the notation varies according to the style and preference of the authors. The last line is meant to be typical of notation used in Bayesian texts, where one talks of conditional probabilities; the first line is typical of texts on quantum (in which case the weights are complex numbers) and sometimes seen in tensor categories and category–theoretic texts. The $\hat{e}_k$ are simply unit basis vectors, in the conventional linear algebra sense.

For the present purposes, these differences are immaterial; yes, of course, quantum is not the same as Bayesianism, but for now this does not matter. The distinctions will be drawn later; for now, the emphasis is that graphical elements, such as jigsaws, can be formally combined to form vectors, and that distinct graphs are the "basis vecttors" of the vector space.

Computationally, the graphical elements depicted in the diagram should be stored somewhere, for example, in a graph database; and alongside each there should be a floating–point value. It can be stored with the graph, or outside the graph, as a distinct vector; this is a software engineering issue and not a fundamental issue.

What do the vector weights "mean"? These are perhaps context–dependent. If one is visiting an astronomical observatory, it might make sense to set $(w_1, w_2, w_3) = (0.05, 0.05, 0.9)$ but if one is taking a walk in the park, then perhaps $(w_1, w_2, w_3) = (0.9, 0.1, 0)$ with w_2 non–zero because one wears eye–glasses. But the "actual

meaning" of this vector is somewhat immaterial; the weights will ultimately be determined by whatever algorithm one uses, and whatever "meaning" that algorithm assigns to weights. The algorithm might be some deep–learning neural net transformer; it might be some Bayesian system; it might be a frequentist counting algorithm.

Later on, we will make the conceptual jump that one may work with many different algorithms "at the same time", and that one can (meaningfully) arrange these into a vector. The weights (w_1, w_2, w_3) might indicate how many CPU cycles should be devoted to algorithms e_1, e_2 and e_3 but they might also mean how often these algorithms "get the right answer". The perceptual framing at this more abstract level is "how can I perceive which algorithm is the best one?" The intelligent agent might be an artificial scientist, running experiments to determine which of the scientific hypothesis e_1, e_2 and e_3 are correct.

The implied software design principle here is that the software system must be capable of representing "objects" like e_1, e_2 and e_3 in some abstract way, but also working with vectors and matrices and tensors in a natural way: this is the "neuro–symbolic" idea: the e_1, e_2, e_3 are symbolic, the w_1, w_2, w_3 are neuro.

The implied context of this text is that the appropriate software system for this work is the AtomSpace. This claim is founded on the author having spent more than a decade to realize the above (and more, some of which touched on below) in a practical, efficient, usable and useful way. The design is imperfect; new insights are gained regularly. Of course, other software frameworks are possible; the engineering task is to avoid common pitfalls, design oversights and, of course, re–inventing the wheel. The AtomSpace is not the be–all, end–all, and it might not be at all suitable for whatever you have in mind. In such a case, use what you need to use. For myself, speaking personally, it is the best that I've got, at the moment. The actual stress of daily use does mean that the process is evolutionary, and that revisions and enhancements are ongoing. Perhaps at a lower rate than before, perhaps limited to narrower sub–modules and subsystems. The theory drives the practice drives the code and the driving also happens in the reverse.

2.1 Frequentist vector representations

In the narrow context of deep–learning neural nets, it is widely understood the vector embeddings work extremely well, taking the form of modern LLM's and a huge variety of applications of transformers to a broad range of topics. What is not well known is that there are other ways of getting at least some of these results.

The goal of this section is to show how to obtain these results, and to provide a very short review of linguistics. Now, of course, present–day LLM's far exceed the abilities of present–day traditional linguistics; and the latter have fallen into disfavor threatening obscurity. The ideas developed immediately below is that there is another path, this path is explicitly symbolic, and that this path, to the extent that it has been pursued, actually reproduces at least some neural–net results.

The original prototype system was WORD2VEC and the prototypical example is the vector relation King–Man+Woman=Queen. Here, the word–tokens for the words "King", "Queen", "Man" and "Woman" were given vector embeddings, and the system, a multi–layer recurrent neural net (RNN) was trained on a large body of text, using gradient–descent methods. After training, the resulting vectors have several interesting properties: they exhibit "associative memory", in that nearby vectors appear to be related, and, much more surprisingly, vector subtraction, i.e. subtracting the vector for "Man" from the vector for "King" seems to give a vector representation for "ruler" that is gender–neutral, in that "Queen–Woman" is another vector that is close to "King–Man". Here, "close" means that the cosine product (the vector product) is approximately 1.0. Again, this is all well–known.

What is not well–known is that the above results can be obtained in at least one very different way: by frequentist counting. Below follows a short sketch of how this is done. The focus and lesson here is that the algorithm is not only very different from that of RNN's, but is also explicitly graphical (symbolic) in nature. This provides the leverage needed to make the jump from neuro to symbolic (structural) and back.

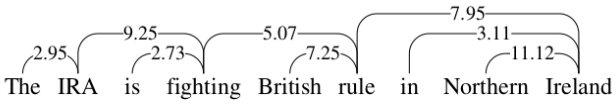
The algorithm described here was provided by Deniz Yuret in his PhD thesis in the late 1990's. A variety of related formulations were explored by a number of authors (including Dekang Lin and Pedro Domingos) over dozens of publications (if not more). None appear to have made the full conceptual leap being described here, although they all danced very close. I don't entirely understand why; this is perhaps the intermittent nature of scientific discovery.

The Yuret algorithm starts with a text training corpus, and simply *counts* nearby word–pairs. Nearby simply means "within a small context window", traditionally six words wide. This count is written as $N(w_a, w_b)$ for words w_a, w_b (no relation to the weights w in the previous section.) Yuret defines the mutual information (MI) associated with a word–pair as

$$MI(w_a, w_b) = \log_2 \frac{N(w_a, w_b)\, N(*, *)}{N(w_a, *)\, N(*, w_b)}$$

where $*$is "wild-card", so that $N(w_a, *) = \sum_u N(w_a, u)$ summed over all right–words u. Yuret observes that words that "go together" have a high MI. For example, he finds that the MI for "Northern Ireland" to be very high; this has to do with the presence of the word–pair "Northern Ireland" in many newspaper stories of the era.

This allows him to propose a parsing algorithm that gives results rather close to those proposed by linguists, in which a word such as "Ireland" is called a "noun", while "Northern" is taken as an adjective, modifying "Ireland". The resulting parses are of the form of a dependency grammar, *a la* Lucien Tesnière, and more generally along the many rich elaborations of dependency grammars over many decades, hundreds of books and far more papers. An example parse, taken from Yuret's thesis, is shown below. The numbers above the word pairs are the numerical MI value obtained from counting.



The parse is obtained by considering all possible trees joining the words, and selecting the tree for which the total MI is the largest possible. Basic familiarity with linguistics shows that this parse is exactly correct: all of the dependencies are indicated exactly as they should be. This is remarkable, because it shows that the toil and trouble that linguists have gone through to assemble lexis (dictionaries) by hand, indicating subject–verb–noun relations, adjective–noun relations, determiner–noun relations, prepositions, etc. can be fully automated simply by counting and noting the frequency. This is a remarkable achievement. This result is explored by many authors under the rubric of "maximal spanning tree parsing" (MST parsing) and is tied to more general principles of "maximum entropy principles" (MaxEnt).

There are two things that Yuret does not do, and apparently no one else of that era, exploring variations on this theme, seem to explore or discover. One is that there is a vector embedding, and that this embedding reproduces exactly the famous WORD2VEC result of King–Man+Woman=Queen. What's odd is that, chronologically, this result could have been discovered earlier, but wasn't. The second important step is that more complex structures can be recursively discovered.

First, lets look at the vector embedding. It is nearly trivial, once you see it. The vector to be written is

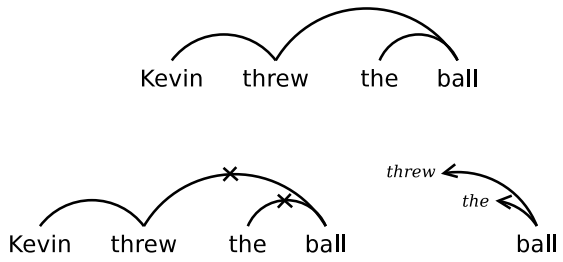
$$|King\rangle = \sum_a c_a \, |w_a\rangle$$

where the sum extends over all words in the vocabulary. Each word is associated with a basis vector, written here as $|w_a\rangle$. This is in the sense of the diagram up top, except that the graph is a trivial graph: it is just a single vertex, having the word as the vertex label. The c_a are just floating–point numbers. To belabor the point, the sum over the vocabulary can be written as

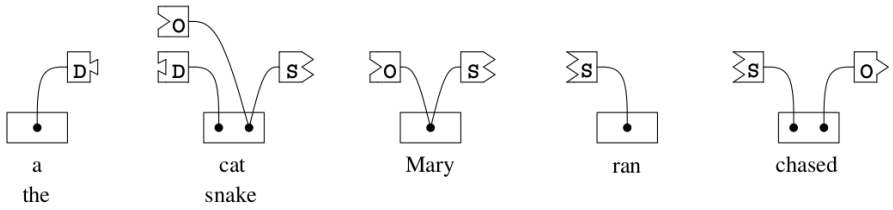
$$|King\rangle = c_{dog} \, |dog\rangle + c_{tall} \, |tall\rangle + c_{other} \, |other\rangle + \cdots$$

and the Ansatz is that the constants c_a can be taken to be $c_a = MI\,(w_a, King)$. This gives a bona–fide vector, and, as a vector one can take dot products, and compute the cosine angles between them, and, when one takes care to normalize the vectors to unit length, one recovers the prototypical vector embedding result King–Man+Woman=Queen. This holds entirely without any appeal to RNN's or any of the other mechanics of deep learning. Nor is there any immediate need or appeals to free energy, Boltzmann distributions, Markov properties, integrated information theory, Tononi's Phi or any other advanced conceptual frameworks. It can be noted (should be noted) that $MI\,(w_a, w_b)$ is a matrix (a 2–tensor) and that is is kind–of "Markovian" in a certain sense, although just how is requires a much fuller theory.

The second "obvious" extension to the maximum entropy principle that no one seems to make is that it can be extended to obtain the full dependency lexis that is the centerpiece of traditional linguistics. The basic idea is best illustrated with a very simple diagram:



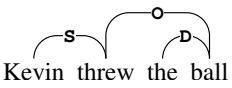
The top row of the diagram show a parse obtained via Yuret's frequentist maximal MI parsing algorithm. The lower row merely applies a pair of scissors to cut the graph into jigsaws, and labels the cut ends so that they can be reconstructed. This disgram should be compared to the figure below, taken from the 1991 paper describing Link Grammar (LG):



This figure explicitly shows the jigsaw connectors in a jigsaw–shape form. The point of Link Grammar is that if one abides by the rule that matching connectors must be mated according to the obvious matching rules, the generated sentences are always grammatical. The connectors are labelled with the letters D, S, O, standing for Determiner, Subject and Object. The intransitive verb "ran" must take a subject (which is a noun); the transitive verb "chased" takes a subject and an object (both nouns).

As a theory of grammar, Link Grammar "works": it has fully developed and complete dictionaries for English, Russian and Thai. It has a demo dictionaries for German, Arabic and Persian and proof–of–concept dictionaries for Vietnamese, Turkish, Lithuanian and Indonesian. It works.

A few quick remarks about notation are in order. The formal Link Grammar parse for the prior example would be



The lexical entries, in conventional LG notation are

```
ball:   D- & O-;
Kevin:  S+;
the:    D+;
threw:  S- & O+;
```

The capital letters, together with a +/- sign, are called "connectors"; a pair of connectors form a "link". The +/- sign indicates a left–right directionality: for example, the `S+` connector can only connect to the right; it must mate with an `S-` connector to form an `S` link. A valid parse exists if and only if all available connectors are paired up. The construction `S+ & O-` is called a "disjunct"; the name has a historical basis that is of no particular concern here. During parsing, all connectors in a disjunct must be satisfied.

Each lexical entry is a word–disjunct pair; they are of the general form

```
word: A- & B+ & C+ & ...;
```

This is the notation used in the original LG papers. It turns out that this is also an active topic of research in quantum computing; Bob Coecke gives a quantum algorithm that can parse LG in linear time. The notation used in those papers would write

$$|word\rangle = \langle A| \otimes |B\rangle \otimes |C\rangle \otimes \cdots$$

Please note that although the notation is quite different, the intended interpretation is exactly the same. This is not an accident: it turns out that algebraically, the system of jigsaw connectors forms a semi–commutative monoid; the internal logic of this category embeds a fragment of linear logic; this fragment is the same as that for history and trace monoids used to describe communicating computing networks, and the mutex locks needed to serialize access to data. But linear logic is the internal logic of dagger–compact categories, with are the categories that describe Hilbert spaces and quantum mechanics more generally. It is for this reason that one can discover quantum computing algorithms that perform linguistic parsing: the logic of jigsaw assembly is not just a subset of quantum, but is also the correct language for describing communicating computer processes and the message–passing between them. Informally, you can think of a jigsaw piece as a computer program, and the jigsaw connectors as the ports through which it can communicate.

This last paragraph consisted of a waterfall of jargon and buzzwords. The intent of this paragraph is to make the statement that the jigsaw paradigm is fundamental: it has algebraic properties that are exactly what is needed to describe a vast array of phenomena. The use of jigsaws for linguistics is noted as early as the 1960's; the utility for computing isn't noted till decades later, and the connection to quantum is not noted until the idea of string diagrams and the Mike Stay / John Baez "Rosetta Stone" is written. Perhaps this sounds somehow deep and magical; actually, its not. It is simply the observation that if you take a pair of scissors to a graph, and cut up the edges, but preserve the labels, you get jigsaws. Since, in a sense. "the universe is just a collection of

objects and the relations between them", it is not a surprise that "everything is describable with jigsaws".

A short note about algorithms. The diagram above yielded a jigsaw

```
ball: the- & threw-;
```

whereas the desired LG representation would be

```
ball: D- & O-;
```

How can one get from individual words, to connector categories? Easy: clustering. On can again perform frequentist counting, obtain MI , compute cosine products, and create clusters from the resulting "associative memory". This has been done in the OpenCog Learn project, circa 2016–2020, and it works fine. The tooling for all this exists for the AtomSpace; however, it is now old, and there is good reason to redesign it from the ground up, as the next sections will review.

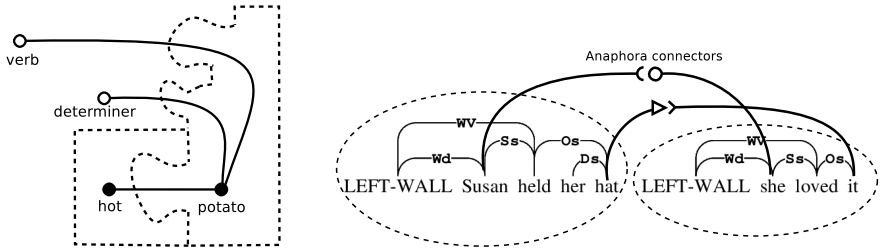
BTW, once again, one has embedding vectors: one can write (for example)

$$|King\rangle = c_1 \, |the\rangle \otimes |threw\rangle + c_2 \, |ruled\rangle \otimes |Poland\rangle + \cdots$$

and after computing the constants $c_1, c_2, \cdots$ appropriately, one will find, once again, that King–Man+Woman=Queen holds as a vector relation.

2.2 Compositionality and recursion

The jigsaw paradigm allows the structural ideas to be re–applied at higher layers of inference. This is most easily illustrated with a pair of diagrams.



The point being illustrated here is that smaller partial assemblages still present themselves as structures that can have relations between one–another, with open connectors that can be joined together. As before, the relations are explicitly identified: the anaphore "she" explicitly refers to "Susan", occurring earlier in the text. In principle, the same techniques applied at earlier stages to discover relationships (the pair–wise relationships between words, and the construction of a grammar) can be used again to discern structural relations at this higher, more abstract level. In principle, this can continue onwards, indefinitely, working at ever–higher abstraction layers. The above figures are still relatively concrete: "she" explicitly refers to someone. At the multi–paragraph or page

level, the structural relationships may be at the abstraction level of a social encounter, and at the multi–page level, perhaps some romance or perhaps some adventure or murder mystery. It should be possible to recursively discern a web of explicit structural relationships between increasingly abstract elements.

"It should be possible", of course, carries a lot of hope. Code for working with word–pairs and grammatical structures is fully complete, debugged and works great. There's a docker container that puts all these software pieces together, you turn it on and go. Initial steps were taken to work the code base to be able to perform these higher levels of recursive abstraction. A raft of issues then presented themselves, and so it is appropriate to halt for a critique at this point.

2.3 Present status and future directions

The theories and ideas presented above are old and well–established. The claims made, specifically about the vector representations, the behavior of the vector and matrix embeddings as a form of associative memory, and the ability to extract grammar, this has all been explored and demonstrated, and it works. I've personally written up an extensive body of papers detailing both the theory and presenting the results, analyzing them and putting them in context. There is a mature body of code implemented in the AtomSpace to perform this counting, compute the MI, compute vector products, perform the high–dimensional clustering to obtain categories. All this is attached to a generative system that can generate new sentences conforming to this grammar. Under it all, as a foundational layer, the AtomSpace can save and restore the graphs, weights and vectors to and from disk, and ship them around over the network, expose them on a network server. It works, but a critique is in order.

Some points, from narrow to broad:

- The core foundation provided by the AtomSpace is solid. It stores graphs and vectors just fine.

- The computation of MI and assorted products (cosine, Hamming, etc.) is implemented as a library. It is matrix–oriented: vector elements v_i and matrix elements M_{ij} are exposed; these use graphs as basis elements (as explained above) and the graph representation allows extremely sparse vectors and matrices (after all, "Kings" are mostly not related to fire–hydrants, stars and Oolong tea. Relationships are necessarily sparse.)

- The library is just that: a library. It really should be represented as Atomese, that is, as a graph itself. That is, instead of saying that $dot\,(v, w)$ is some function call to a subroutine called "dot", passing two vector arguments v and w, it should instead itself be represented as a graph, as a jigsaw, so that v and w are input jigsaw connectors, connecting to vectors, and "dot" is an output connector, connecting to anything that wants a real number as input.

- The goal of representing the algorithmic elements (dot products, Hamming distances, etc.) as jigsaws is so that meta–algorithms can themselves connect, change and re-arrange what is connected to what. That is, although MI, the mutual information, is a good way to measure affinity, there are in fact a large number of similar measures, some of which might be more accurate, and others which might be faster. The idea is that these should not be explored by human software engineers vibe–coding with Claude, but by meta–algorithms, perhaps genetic algorithms, assembling these (jigsaw) pieces.

- The above idea of implementing algorithms as processing pipelines is partly implemented; mostly implemented at the base layer. The generative parts, where pipeline elements can be reconnected by meta–algorithms, this has not been implemented. Both MOSES and AS-MOSES serve as a proof of concept of this idea, but need a fundamental redesign to be generally applicable. A later part of this paper will expand on this, but there is a mire of engineering and design issues to be encountered here.

All of the above–described results are obtained by counting. None of the current counting algos run on GPU's, nor is it obvious how to port them to GPU's. This is an open, unresolved question. The primary strength of the LLM & transformer algos is that they are extremely parallelizable and can be implemented on GPU's very efficiently. It is not at all clear how to do this for counting.

The tease is even worse, in some ways: although vectors and vector embeddings appear, and there is more than just superficial resemblance to RNN's, the precise details of the correspondence remain shrouded in mist. Perhaps a strong correspondence can be made between the above tensor framework, and RNN's, and perhaps not. I have personally attempted to find such a correspondence, but haphazardly, with only a half–hearted effort, as my attention is focused elsewhere. Still, having such a correspondence would be a breakthrough, as it would build a bridge between LLM's that currently reign supreme, and more traditional symbolic computing.

2.4 Perception, action and agency

This will be the topic of the next major section, but the motivation for it appears at this point, where one attempts to jump from the lower to the higher abstraction layers.

Conventional training involves the force–feeding of a blob of text through an algorithmic system. For he Yuret pair–counting pipeline, it is just that: counting word pairs. Upon completion, the MI can be computed. After the MI has been computed, a second pass of the textual data can be made, this time performing Maximum Spanning Tree (MST) parsing (or Maximum Planar Graph (MPG) parsing). This time, the disjuncts are counted, and upon completion, the MI can be computed, and the resulting vectors placed into the lexis (specifically, the Link Grammar (LG) lexis) where it can be used to generate text (random grammatical sentences). Anaphora resolution would require a third pass. Each

pass gets progressively longer, uses more CPU time and is a bit more fragile. The more abstract things get, the vaguer and messier the results, the less certain they seem to be, the less accurate. Perhaps a larger corpus solves all? Perhaps a longer training time? These issues sound exactly like the kinds of issues faced by LLM's, transformers, and almost all other machine learning systems. The simple answer is to just throw more data at it, more CPU, crunch longer and harder. But there is a better way.

One desire is to use feedback from higher layers to correct or amplify results from lower layers. The pipeline approach does not allow this. Another desire is to continue reading more text, feeding and expanding the lower layers, even as processing continues at the higher layers. Again, pipelineing does not allow this. Incremental ingestion of data also means that things like MI must also be updated incrementally; the pipeline computes it in one giant batch job (that takes hours to run, depending on the dataset size).Incremental ingestion also suggests that partial results should be saved to disk every so often. For several reasons: The system may crash or lose electric power; one wants to save at least some results. A bug might render later data into garbage; one would like to be able to rewind to earlier forms, without having to restart from scratch. Finally, the system seems to work best if the training done at each stage is on the same dataset as the earlier stages: the grammar induction should be done on the same texts as for which word–pair counting was done.

All this put together suggests that there should be some sensori–motor system put into place. The "sensory" part of this can be as simple as reading all of the files in a directory. The "motor" part of this is as simple as changing directories. Finally, there is a very minimalist "world model": it is very small and simple: it consists of filenames (filepaths, or URL's), plus a collection of bit–flags: has pair counting been run on this file, yet? Has MI been done yet, for this pair data? What about grammar induction? Anaphora or higher level counting? And maybe more: when is the last time this file was touched? Perhaps this file is contains similar contents to another file? Perhaps even its an exact duplicate (because we're crawling over web–site mirrors, or backup copies, or other reasons for duplicated files?)

These three parts put together suggests that there is (should be) an agent: this agent perceives files, remembers basic facts about files, and makes decisions to change directories, or to revisit files for a second or third scan.

Once one has the idea of a file system agent firmly rooted in mind, it is easy to imagine more general agents: perhaps chatbot agents that can hop onto chat channels, or twitter/bluesky/facebook agents that can work with social media posts, "sensing" (reading) and "moving" (going to different channels, or following different users), and remembering some basic status info about the interactions so far.

This naturally leads to the next major question: what is the appropriate design for an agent, in general? What is a generic description of sensory capabilities, and movement capabilities? Hand in hand with these questions comes the realization that such agents cannot be hand–coded, and must not be engineer–designed. This in turn implies that the agents must be coded in Atomese

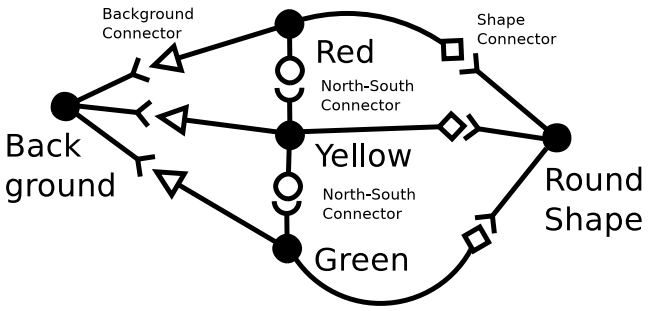
themselves, represented as graphs, as dataflow arrangements of jigsaw pieces, with the specific agents being generated according to a grammar of what parts of the agent pipeline can be connected to what (e.g. vector outputs must be connected to vector inputs; things that expect strings must not be connected to things that expect floats, and so on: there is a connectivity grammar, of what can be attached to what.

That is, agents should be auto–generated, explored and measured for quality, accuracy, speed and effectiveness; mutated, assembled and recombined, according to the the alloweed grammatical rules. This opens a new vista of design and representation questions. This is just a first taste, a later section will go into more detail. BTW, portions of such an agent system have already been prototyped in Atomese.

But first, lets get back to some baser, more grounded topics, to give a better idea of what other kinds of processing might be encountered.

2.5 Vision

The above uses natural language and grammar as the base example; but the concept generalizes. This is most easily communicated in a few pictures.

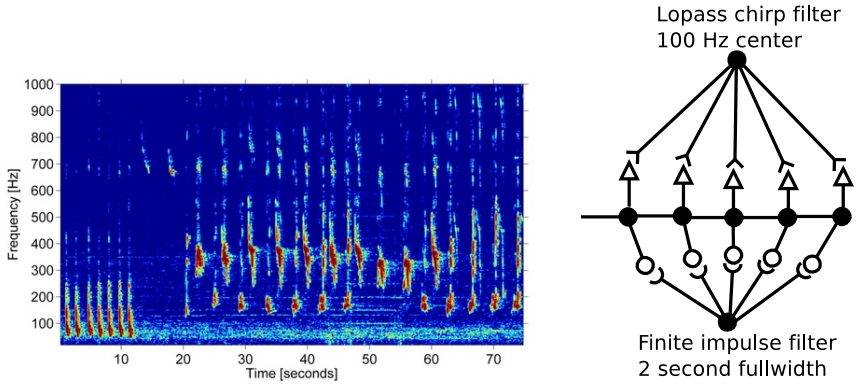


Shapes have a structural grammar. The connectors can specify location, color, shape, texture. It's not about pixels, its about element groupings into recognizable, recurring arrangements.

It is also explicitly symbolic. Each jigsaw piece identifies a specific element of the picture. It is identified in a form that we humans are already comfortable in working with: the round thing in middle, that's the yellow light. As a jigsaw, and its relationship with it's surroundings, it's accessible to reasoning, to logic. This is very different from the situation with neural nets, where you have collections of weights to deal with, and not the objects, the symbolic representations of the objects themselves.

2.6 Sound

The diagram on the left is a sonogram recording of a whale song; on the right is a jigsaw representation of an audio filter sequence that is capable of recognizing the form and structure of that whale–song.



The goal of this illustration is to show that audio elements can also be decomposed into elements of a symbol nature. Roughly speaking, the diagram on the right can be thought of as a representation of an electronic circuit, an electronic wiring diagram, a way of hooking together digital signal processing (DSP) functions capable of triggering on the whale song. The claim is that the jigsaw paradigm has legs: its a generic concept applicable in broad domains.

2.7 Critique redux

The vision and sound processing diagrams above present several new challenges. Starting at the narrowest:

- The library that implements the MI, dot–product, Hamming–distance, etc. functions is only partly pipelined up. It is not GPU–ready. Optimization, data flow, performance is a concern.
- The collection of trees must be equi-dstributed on a manifold, aka MaxEnt aka free energy, etc.

3 Sensorimotor aka "Part Two"

Outline, to be written and expanded on.

- Sensory perception in any domain, which can be arbitrarily abstract, e.g. the perception of movie plots, the perception of basketball defense strategies, the perception of geopolitical relations, the perception of abstract algebraic mathematical objeccts.

- Movement, as a choice of where attention should be focused, of moving from one topic to the next. Attention as a form of movement through the abstract sensory space.

- Agency as a individual having the ability perceive and move, and having a finite amount of internal state, the "world model", of the exterior world. The exterior world is unbounded, thus, the limited finite model of the agent is necessarily incomplete.

- Example: a text agent. It can "perceive" file contents, it can "move" by changing directories (changing URL's, "crawling the web") and has a finite state: e.g. URL/file file:/usr/include/some/where has been read, split, tokenized and counted at the first level, but second stage has not been done; size of tile is 4321 bytes, and file contents is gziped utf8 text. This is the agents "world model" (internal model) of what the "external world" actually looks like.

- Agents are composed of sub–agents, and it is recursively agents "all the way down", e.g. humans have a kidney, liver, heart, muscles that function cooperatively. The kidney is built of cells, organized into a social structure suitable for renal function. Each cell is built of mitochondria, centrioles, endoplasmic reticulum, etc. that function as individual agents operating in a cooperative level. Each of these are built of biomolecults, some kilodalton in weight, operating according to the principles of n-wave interactions, e.g. certain shapes of certain molcules fit, like a key in lock, due to not just the 3D shape, but the spectral vibrational electromagnetic properties. I.e. the molecules "perceive" shape, and the molecules "move" to conform, and they have a "world model" indicating a past path (including hysteresis) to get to the present shape. In short, its agents all the way down.

- Its also agents all the way up: humans organize into social groups, corporations, institutions, political groupings, etc. which also exhibit agential properties.

- There is a basic mathematical description of agency in a category–theoretic framework, developed by Greg Merideth; its a half–dozen algebraic structures and a dozen relations that capture the idea of sensing, movement, attentional focus, and having a world mode. As always, anything expressible algebraically is always "easy" to turn into functional computer software, simply because the algebraic preciseness means that it is clear how to write matching code.

- Any algebraic expression is directly representable as "jigsaw pieces"; the category–theoretic algebraic structure of agency is just a collection of a dozen or so jigsaw pieces that fit together. The various topics, techniques and generative algos touched on in part one can be deployed in this domain as well. This is how we get to "its agents all the way down", but also that

"the structures have specific shapes and attachment affinities of what can connect to what.

- The Per Bak model of a sand–pile at the self–organized critical point seems to be the appropriate inspiration for "what life is". Recall the sand–pile model. Grains of sand are dropped (at one location) until a hill develops. As the hill grows, the slop of the sides reaches an angle, called the "critical angle". At this angle, avalanches occur at all scales: many small avalanches, some medium–sized ones, and rarely, large avalanches. As sand-grains continue to be dropped, the slope remains the same: this is the so-called "self–organized criticality": the slope stays near the critical point, despite avalanches, despite additions of new grains of sand.

- The biological inspiration is that the grains of sand are replaced by jigsaw pieces, with either fit, or don't, and build up into structures, but eventually reach a critical point of being unable to sustain further growth at that scale, and must fracture into smaller consistent functional units. These functional units are the "agents"; the agents form the functional units at the next–higher layer of abstraction.

- A conventional, traditional conception of AI/AGI is, for example, a system capable of "abstract reasoning". In formal logic, one has collections of axioms, and theorem provers that can recombine axioms to arrive at proofs: inferences, deductions, abductions, etc. The axioms and the inferences rules are both examples of jigsaw–pieces; the assembled proof is a collection of jigsaws assembled according to the affinity rules for the jigsaws (i.e. one can only use some particular inference rule with only certain inputs, to create other outputs.)

- The traditional issue of using formal logic/theorem proving as a basis for AI/AGI is that human thinking is not axiomatic, and does not conform to the formal systems of e.g. Aristotelian logic (or of predicate logic, or of any of dozens of other kinds of logic): human "common sense" is not axiomatizable.

- The proposal being made in this text is that "common sense" can be arrived as an agglomeration of events, relations and correlations, decomposed into jigsaw pieces as explained in part one, and rearranged into functional agglomerations that work "most of the time", in conventional settings with conventional sensory inputs and tasks.

- The collection of "rough and ready" assemblages self–organizes to a critical state, where each new observation either conforms to a pattern, or forces a re–structuring of "what is known" (if you are open–minded) or flat–out rejection (if you are closed–minded) or unwarranted incorporation into your belief system (if you are gullible) or flat–out rejection (because the new evidence is "just plain wrong") I'm using human psychological terms in this last sentence, but the intent is that this is happening at the agential

level: all agents, at all size scales, are accumulating new sensory inputs, treating them as jigsaws, which either incorporate into the existing structure, or don't. This is an update of the "world model", and there may be movement or changes in attentional focus, as a result.

- Examples of avalanches, at the human level, are nervous breakdowns (in the classic 19th century psychological sense: you've built your entire world around a belief in your husband/wife/partner, who has betrayed you.) Non–painful, medium–sized avalanches include the epiphany, the "a–ha" moment, where the last bit of evidence falls into place, re–arranging into a brand new understanding. Major avalanches include the religious transcendental experience: the ineffable experience,, Moses and the burning bush, inefffable precsiely because the restructuring is so profound that there is no way to put it into words. Moses starts as an ordinary man, and returns with the deep and fundamental realization that the foundations of moral, ethical behavior can be structured on Ten Commandments. Where did these come from? Inexplicable; he spoke to God. In the structuralist vantage? It was an major avalanche, restructuring the fundamental relationships between "what is true" into a new and stable form (the Ten Commandments) that can support a weight that the earlier form could not (a random hodge–podge of life lessons in community behaviors, misdemeanors, crimes, wholesome upright behavior and heroic deeds.)

- Examples of avalanches at the superhuman level include financial crises and war, as well as e.g. the industrial revolution. Avalanches at the subhuman biological level include assorted medical conditions, both positive (e.g. growth) and negative. Again "its agents all the way down", and "its self–organized criticality all the way down".

- The structuralist, algebraic approach of working with jigsaws appears to be a good way to capture the concept of structure and relationships between "things", explaining how/why "life is living", while also being sufficiently formal that it is convertable into functional digital computer software.

- Disclaimers: large parts of this section are hypothetical and speculative. Actual computer implementations my very rapidly encounter combinatorial explosions and scaling limits, preventing anything beyond toy models from being workable. Still, there's hope: deep–learning neural nets also had scaling problems and combinatorial explosion problems, until we threw a few tens of millions of GPU's at it. The raw compute power tipped the balance. Well, that, and the discovery of certain algos, such as back–propagation.

The End. Above is an outline. It should be cleaned up.

References