Introduction

It is perfectly possible to train a neural network with logical data. However, the learned logical system typically has a fixed structure, on the order of the size of the neural network, and rules must be learned at each position in the structure where they are to be used. The network is unable to re-apply abstract logical rules at multiple locations in the structure.

Some systems expressly designed for logical reasoning have been enhanced with probabilistic capabilities, giving them many of the benefits of neural systems (e.g. [2]), but at the top level they remain rigidly structured, missing neural advantages at the highest level.

Solving the reusable symbol problem defined here would solve several of the integration challenge problems of [1]. For example, logical statements (including ones with quantifiers) can be processed just as in formal logic, using axioms and axiom schemas as the "larger patterns" (discussed below) into which reusable symbols are placed, with the sequence of statements in a proof creating a coherent structure much like the tiling in the problem discussed below. This approach is not such a new idea (note the title and date of [4]), but so far it has not been successful, simply because the reusable symbol problem has not yet been solved.

In this paper, first I will examine in detail what symbols are and how they are useful, then I will briefly dispel the illusion that pointers are the key to the power of traditional computation (as compared with neural network approaches), and finally I will give a clear instance of the reusable symbol problem.

Symbols: a mechanism for encapsulation and reuse

The symbolic processing that arises in neural networks occurs among a fixed set of symbols with fixed relationships to each other. The fixed set of symbols is not so worrisome, as people also tend to be content with existing symbols when manipulating information. But the fixed relationships are a more of a problem. The reason we are happy to use the symbols A and B over and over again is because we can easily remap the relationships between them. If we are told that two A's must always be followed by a B, we can immediately understand and apply this new constraint on old symbols. Perhaps surprisingly, this is even easier to understand than using dedicated symbols for this constraint, e.g. two $\mathscr{S}$'s must always be followed by a $\mathscr{C}$.

This sort of symbol reuse matches with our experience of programming, but contrasts sharply with the kind of symbolic processing that appears in neural networks (even in modern network architectures such as graphical model based designs). In these networks, we typically do not have any notion of reuse of symbols.

We are so used to symbolic reasoning that it is worth reminding ourselves what kinds of computational primitives are implicit when one uses symbols. The most basic fact about a symbol is that it represents something. That is, there are two objects: the symbol, and the object it represents. The symbol could be a letter representing a mathematical variable, or a street sign representing a particular rule of the road, or a name representing a variable in some computer code, or a word representing an idea, or any of many other possibilities. The symbol itself is typically relatively small, while the represented object can be quite complicated. The symbol "stands for" the represented object, meaning that wherever the symbol appears, we understand that the represented object is essentially there (although this substitution may be hard to imagine in cases such as a variable name in code, or a particular bitmap representing a letter of the alphabet, or other cases where the represented object does not have any other practical form in which it could appear). This link from symbol to represented object is one-way: the represented object is not tied to the symbol, and could equally well be represented by any other available symbol.

Importantly, a symbol can appear an arbitrary number of times. Once we know how to recognize the symbol and what the symbol stands for, we are ready to use the symbol. Using the symbol means that the symbol can appear in some larger pattern which provides a context for this instantiation of the represented object. This larger pattern may be visual, or may be textual, or grammatical, or it may simply be a fixed relationship between a fixed number of objects. For example, the larger pattern could be "___ comes between ___ and ___", and this larger pattern might get filled with the symbols A, B, and C, with A representing noon, B representing morning, and C representing evening. The symbol carries the meaning of the represented object into the larger pattern, meaning that the represented object has some kind of structure, and the larger pattern indicates some kind of structure between the elements of the pattern, and the symbol represents the presence of the represented object's structure within the structure of the larger pattern. Typically there exist constraints on the represented object imposed either by the larger pattern itself or by the larger pattern in conjunction with other represented objects that also symbolically appear in the larger pattern. For example, in the above example, if A represents noon and B represents morning, then C is fairly strongly constrained to also represent a time, perhaps a time in the afternoon or evening.

In language or imagery we are used to reasoning along lines like "this can go here, that can go there," meaning that certain symbols can fit into the larger pattern in certain places, subject to the constraint that the represented object should fit well into the larger pattern. Even when putting the pieces together of anything conceptual or abstract, we appear to operate in terms of building up coherent larger patterns of represented objects. In this construction process, why do I say we are placing symbols? Why not skip the symbols and simply place subpatterns into larger patterns? One reason is that if we reuse an object, then in the result we know that the two instances are exactly the same object, without having to inspect the subpatterns (object instances) to see if all their details match. Another reason is that pattern parts have their own structure of subparts, each part of which again has its own structure of subparts, and so on, and it seems unreasonable to assume that this unbounded detail is copied into each instance of pattern use, or even fully present in a single use. Symbols break this cycle of substructure, allowing there to be parts whose details can be recalled only if needed. Symbols also obviate the need to copy larger structures, limiting copying to duplication of symbols.

Symbols encapsulate their represented object, allowing multiple uses of the object in larger patterns. This fundamental operation is generally lacking in neural network models of information processing.

Pointers and copying

One of the immediately noticeable differences between neural networks and computer programs is that programming languages have pointers. For example, every data structure is found using a pointer. Indeed, much of the symbolic manipulation carried out by computers is done by using pointers as symbols for the objects they point to. As with symbols, it is worth reminding ourselves how it is that pointers are useful.

In computer memory a pointer is typically not replaced by what it points to. Rather, the use of a pointer is to allow the CPU to find (copy into the ALU) parts of the pointed-to object. Once an object part has been copied to the ALU, data processing operations can be performed, perhaps calculating another pointer. Finally, if part of an object needs to change, that part is copied back into memory.

In the electronic hardware, data is represented by bits, which are in turn represented by voltage levels, which are capable of existing on (and importantly, traveling along) any set of wires. Indeed, we often think of information processing in terms of data moving around.

Pointers are simply what allow this copying to occur. The pointers themselves do not provide any great functionality – after all, neural networks don't have trouble finding information even without flexible pointers. Rather, it is the movability and copyability of the chosen data format that lies behind our current approach to electronic computing. (And it is our symbolic approach to reasoning that led us to choose such a data format in the first place, having the properties needed by symbols: copyability, and usability as the "address" of addressable memory, which is a one-way associative memory, just what is needed for symbols.)

After decades of neuroscience electrophysiology experiments, there is no evidence that the brain uses such copying operations (although it cannot be ruled out with certainty, and there is not universal agreement). Instead, in the brain, to the extent that we understand the signals we find, they appear to have meaning based on their location (which neuron), rather than based on a spatial or temporal pattern which would have the same meaning even if coming from a completely different set of cells. This makes it unclear how symbolic reasoning is performed in the brain.

The open problem

Here I propose a concrete canonical instance of the reusable symbol problem. Solving this instance would clearly illuminate how this problem could be solved in general.

The goal is to design a neural-style architecture (I will leave this undefined) which permits activations corresponding to solutions of a Wang tiling problem [4]. Wang tiles are square tiles (not to be rotated or flipped) with a color on each edge. A tiling must use tiles from the finite set of available reusable tiles to cover a grid. Two tiles may be used at adjacent locations in the tiling only if they have the same color on the edge where they meet.

The overall architecture of the neural network should consist of two parts: a workspace, and a set of allowed tiles. Each of these may be of a fixed size in a given network, although it should be clear how to expand the network to allow a larger workspace or a larger set of allowed tiles. The workspace should consist of some form of a grid of positions where tiles may go. The set of allowed tiles should be implemented in such a way so that if one wants to change the definition of a tile, then this change can be made in one place, within the "set of allowed tiles" portion of the architecture.

This problem is easily solved by a Markov random field if every location in the workspace knows about the set of possible tiles and their color constraints. The problem with this solution is that every location in the workspace must be independently trained to learn which tiles can go next to which other tiles. In other words, it violates the constraint that a tile definition should only appear in one place.

The challenge is to solve the problem using reusable symbols (whose implementation I also leave undefined). The workspace should be fillable with symbols which represent tiles from the set of allowed tiles, but this should only be stable when done in ways that satisfy the matching edge color constraints.

If this problem could be solved in a neurally plausible way, especially in a way that allows the set of allowed tiles to be learnable from example tilings, then this would represent a significant advance in our understanding of how neural systems can perform symbolic processing.

REFERENCES

[1] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler, 'The integration of connectionism and first-order knowledge representation and reasoning as a challenge for artificial intelligence', *Journal of Information*, **9**(1), (2006).

[2] Kathryn Laskey and da Costa Paulo, 'Of starships and klingons: Bayesian logic for the 23rd century', in *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pp. 346–353. AUAI Press, (2005).

[3] Christoph von der Malsburg, 'The correlation theory of brain function', Technical Report 81-2, Max Planck Institute for Biophysical Chemistry, (1981).

[4] Hao Wang, 'Proving theorems by pattern recognition II', *Bell System Technical Journal*, **40**, 1–42, (1961).