Implementing Spatial Relations in Neural Nets: Figure/Ground and Containment

Matthew Zeidenberg*
University of Wisconsin–Madison and Electrotechnical Laboratory, Japan

October 1991

1 Introduction

There is, no doubt, a division among mental functions. Some are learned, while some are innate. I argue that lower functions, that is, those that are perceptual primitives and are required for much higher level processing, are more likely to be innate than are higher functions, which are more likely to be learned (and therefore culturally affected).

In the field of object recognition, it is likely that recognizers for simple objects, such as edges or corners, are built in, and the recognition of more complex objects is learned, in terms of the innate recognition of more simple components. This is because these simple objects exist everyone in the world, and it would be wasteful for the organism to spend time and energy learning how to recognize them. It may be, also, that it is impossible to learn to recognize both simple objects and then more complex ones.

Some simple spatial relations among objects are, I would maintain, in the class of innate functions. That is, the species has learned to recognize certain spatial relations, so the individual need not. Foremost among these are the relations that roughly correspond to the English prepositions in, on, over, under, across, and between.

In order for spatial relations between objects to be computed, objects need to be recognized. That is, the visual scene has to be segmented into significant chunks before the spatial relations between those chunks can be computed. Also, the nature of the semantics of the spatial relations between between objects depends on the nature of those objects themselves. For instance, if something

*This research was partially carried out while the author was a participant in the Summer Institute in Japan program sponsored by the NSF and Japan's Science and Technology agency. I would like to thank my host at the Electrotechnical Laboratory, Kazuhisa Niki and my research advisor at UW-Madison, Leonard Uhr, for helpful advice and discussions.

is in a box, it can be sticking out, but if it is in a room, it cannot. Of course there is a difference between the natural language usage of these prepositions and any formal function that can be computed from a scene.

This paper discusses one inportant spatial relationship, that of containment (and the related concept of an object's figure and ground). I recognize that the concept of containment varies depending on the objects concerned.

Let us consider, however, the simplest form of containment: the case in which both objects are a connected collection of 1's in a binary array, they are both closed (that is, have no breaks in the their outer perimeter), and one object (A) is contained inside another (B) if every bit on A's perimeter (as well as every bit in A) is contained within the perimeter of B.

Another way of putting the last statement is that every bit in A is contained in the figure of B. The figure of B is simply all points that are within B, whether or not they are part of B or not. The ground of B is the complement of the figure.

Thus one way to compute whether A is contained in B is first to compute the figure of B. If all points in A are contained in figure(B), that is, if A itself is a subset of figure(B), then A is contained in B.

How then, are we to compute the figure and ground of an object?

2 Computing the Figure and Ground of an Object

Most approaches to computing the figure and ground of an object have been based on the idea of spreading activation. For instance, Kienker et al. [2] computed the figure/ground of a spiral by using a simulated annealing to get out of the local minima that a simple relaxation model would settle into. Spreading activation is modelled in a connectionist network by having nodes reinforce their neighbors, and (in the case of figure/ground) having the edge (perimeter) nodes reinforce nodes on one side of them and inhibit nodes on the other.

It seems likely that humans use some sort of spreading activation to compute the figure of an object. Yet simulated annealing is a very slow technique to compute anything, and it is not a very plausible algorithm from the point of view of human biology.

The standard algorithm (see, for example, Sedgewick [5]) for computing whether a point x is inside a closed figure A is the following: draw a line from x to any point completely outside A (such a point is easily found from the extremal points of A on any coordinate system.) If this line crosses the perimeter of A an odd number of times, then x is contained in A, otherwise not. This is because exits and entrances to the figure alternate as one moves out from x on this line.

This algorithm can be readily implemented in a neural network. One can use a parity network. Given two grids, the input grid containing the object, and the

figure grid that one wants to contain the computed figure, one can connect each node in the figure network to all the nodes above its corresponding node in the input network, and thereby compute the algorithm stated above. Various types of parity networks are possible. I have chosen a cascaded parity network, since it minimizes the number of nodes and connections needed. See the Appendix for details.

The direction chosen for the computation of the number of crossings is arbitrary. We can simply, as indicated above, choose the horizontal direction. If the object A has no gaps, this will pose no problem.

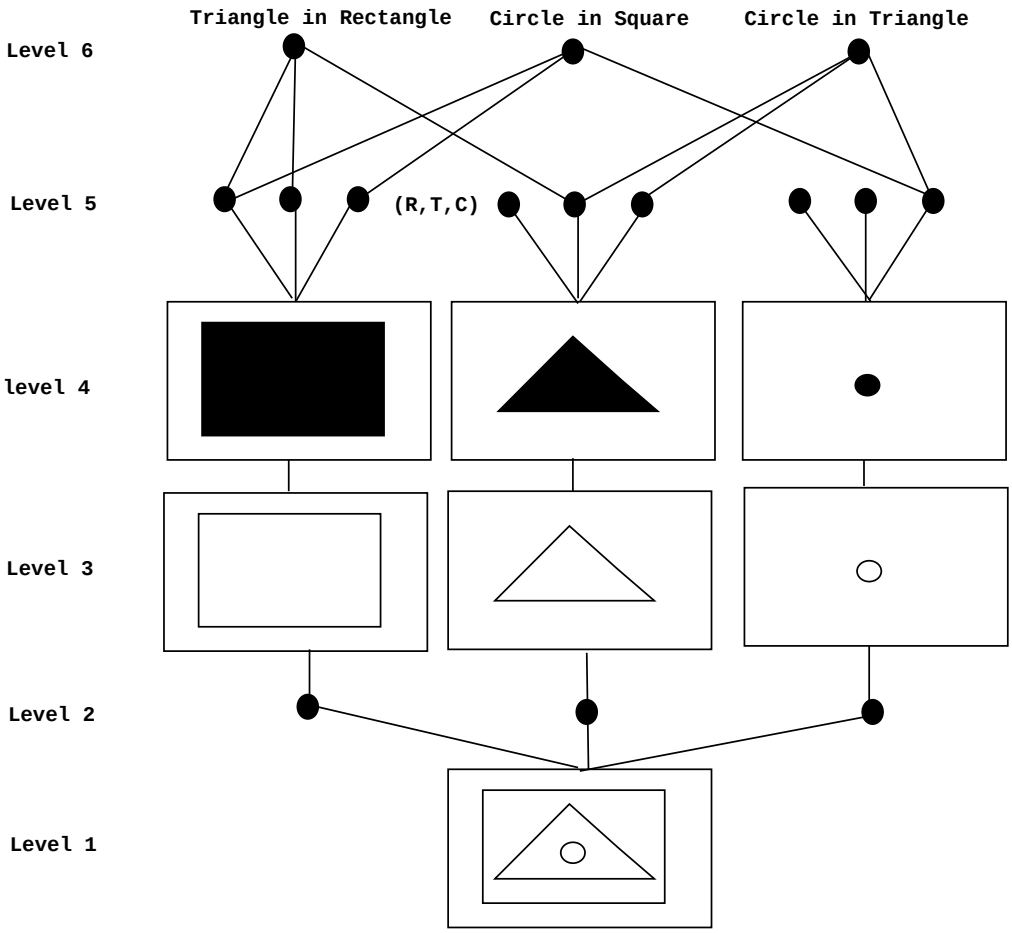
There are some slight problems with this algorithm. It does not work in the case in which the point in question lies along a tangent to the figure in the direction chosen. Also, it does not account for thick edges. One cannot simply pick a point on the grid and count how many points are on the raster line in the image, to the right of it, because two or more of these could be adjacent points in the same edge. In order to deal with this, we can perform edge thinning, which is a standard image processing algorithm, here implemented in a neural network.

In order to build a certain amount of redundancy into our network, we use two directions for the thinning process. and the subsequent filling in of the figure. For convenience, we use the two axis directions of the input grid, although we could in principle use other directions. We first compute from the original input a version of it thinned in the x direction (to the right), then we fill this version using the parity network as described above. We do the same for the y direction (towards the top of the grid). We now have two filled shapes, one for each direction. We compute an image consisting of those pixels which are on in both of these filled shapes or in the original input. Note that the logical functions "and" and "or" which are used to come this final image are readily computed in a neural network.

We perform thinning in the horizontal direction as follows: each pixel that has a neighbor on either the left or the right dies. This is repeated until the only pixels remaining have no horizontal neighbors. The same is done in the vertical direction.

Note that, in the horizontal case, this thinning has a tendency to disrupt edges that are horizontal, even if they are thin. Thus the computation of figure in the vertical direction (bounded by these horizontal edges) will be disrupted. The same holds for the converse situation. When we compute the "or" of the two figures, this disruption is repaired (i.e. "leakage" from the figure caused by the disruption of edges is not put into the final figure.)

Ths result of this is tht we compute, in parallel, the figure of each of the objects found in our original input object. The thinning and filling process goes on between between levels 3 and 4 in figure 1; details of it are left out (that is, the intermediate thinned and filled images).



3 Computing Containment from Figure/Ground

Once we have computed the figure of each object in the image in parallel, we are in a position to compute containment as we outlined above. Levels 5 and 6 in figure 1 in the network do this. Consider the three units at the left in figure 1. These units have receptive fields that detect the rectangle, triangle, and circle respectively. In this sense they are duplicates of the units on level 2. The difference is that they are only detecting these shapes within the figure of the square, if the square is present. The situation shown in figure 1 is that in which all three objects in the input are present, as shown on level one. If, instead, for instance, the square is not present in the input, then none of the nodes in levels 2-5 that detect it (in the left column in the figure) would be

activated. Thus, for instance, the second node from the left in level 5 detects the triangle in the figure of the square. Note that the fact that the nodes in level 5 are all dark *does not* mean that they are activated. In the situation shown in figure 1, if the 9 nodes in level 5 are numbered from 1 to 9, from left to right, then the activated nodes are 1, 2, 3, 5, 6, and 9. Note that nodes 1, 5, and 9, which detect the rectangle, triangle, and square in their respective figures, have the same function as the three nodes in level 2, since if an object is present in the original input, then its figure will always be constructed, and an object is always present in its own figure, by definition.

In order, for instance, for the circle to be present in the rectangle, the rectangle must be present, the circle must be found in the figure of the rectangle, and the circle must be present. These facts correspond to the activation of nodes 1, 3, and 9 in level 5. This is why these nodes output to node 2 in level 6, which detects this fact. Similarly, nodes 1 and 4 in level 6 each receive input from three nodes in level 5. Strictly speaking, there should be six nodes in level 6, but since three of them correspond to impossible containment relations (due to size ordering), such as the containment of the rectangle in the triangle, they have been left out, in the interest of simplicity. Nodes 4, 7, and 8 in level 6 are equally useless, since they detect impossible situations.

The number of nodes at level 5 is the square of the number of objects n that the system is capable of processing. Here there are 9 nodes at level five and 3 at level 2. Since, as we have noted, 3 of these nodes are unnecessary, because of size ordering, the actual number of nodes needed at level 5 is 6, or $n(n+1)/2$. This represents a combinatorial explosion in the number of nodes required, due to the local encoding employed by this system. In a real cognitive system, such as that of people, one would expect a capability to recognize thousands of objects, which would require millions of nodes at level 5 of our system.

The only answer to this criticism that I can see is the use of attention. It is a common rule in psychology that people can only attend to a small number of objects at a given time. This suggests the following modification of the system as described above.

Many parallel recognition channels observe the input scene, but at any time only a few of these channels are ones that are selected, which are the attended-to channels. For each of these channels, a symbol of what they are recognizing is output, as well as a the object itself in isolation (as in our system at level 3), and its scale, location, and orientation. The attended-to channels are "switched into" our system at levels 1 to 3. Our system would have a capacity of 5-10 channels. The output nodes at level 6 Would be interpreted in terms of the symbols that were output by the attended-to recognition channels. For instance, the first node at level 6 would be interpreted as in(triangle,rectangle) because the symbols for triangle and rectangle would be output by the first two channels that are switched-in.

Sandon [4] has studied the problem of implementing attention in connectionist networks. Also, Jacobs et al. [1] have studied modular neural networks

in which a form of switching, or gating, as they call it, similar to what I have described is performed.

4 Future Directions

This general approach, as I have outlined it, could be applied to the recognition of a variety of spatial relations, which is part of my ongoing research. Each relation (such as the relations associated with the prepositions around, across, between, etc.) has a different semantics, and requires a different type of network to compute it. Still, we can have the common goal of expressing the outputs of these various networks in terms of a propositional form such as in(triangle,square) or between(triangle,square,circle).

Once propositional representations of simple spatial relations, suitably temporally marked, are computed by local or distributed nodes in a neural network, it remains to do inference with them. Several authors (e.g. Pollack [3], Touretsky [6]) have shown that one can do production-system-like inference in a neural network, but have not shown any particular advantages of doing it this way. It is therefore not necessary to build inference systems explicitly in terms of neural networks, since one can always re-implement standard systems in a network. Thus one approach that one can take is to translate the output of the neural network into predicates, and then do reasoning using a standard logic- or production-system-based approach.

If we want to learn rules of the form:

$$go(A,B,C,t1,t2) \leftrightarrow at(A,B,t1) \wedge at(A,C,t2) \wedge path(A,B,C,t1,t2)$$

can we use network based learning to do it? There is a severe problem that is caused by the presence of time. That is, if one of the inputs to the system is a clock that ticks many times as an object moves across a grid, the system will detect $at(A,B,t1)$, $at(A,B,t+e)$, $at(A,B,t+2e)$ until the definition of at in terms of B's neighborhood, however that may be defined, is no longer satisfied. We again have the problem of attention, that is, of deciding which piece of information is salient. The presence of time creates an explosion of facts. Time is implicitly present in many statements about spatial relationships: for example, in the sentences "The ball rolled between the posts", or "The tiger ran around the tree."

It is clear that some understanding of the concept of a path in time is critical to an understanding of sentences of this type. Thus paths form a starting point for building systems that can comprehend time-based spatial relationships.

References

[1] Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. Task decomposition through competition in a modular connectionist architecture: The

what and where vision tasks. Technical Report 90-27, Department of Computer and Information Science, University of Massachusetts, Amherst, 1990.

[2] Paul K. Kienker, Terrence J. Sejnowski, Geoffrey E. Hinton, and Lee E. Schumacher. Separating figure from ground with a parallel network. *Perception*, 15:197–216, 1986.

[3] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, Nov 1990.

[4] Peter A. Sandon. Simulating visual attention. *Journal of Cognitive Neuroscience*, 2(3):213–231, 1990.

[5] Robert Sedgewick. *Algorithms*. Addison-Wesley, Reading, Mass., 1983.

[6] D.S. Touretzky. Boltzcons: Reconciling connectionism with the recursive structure of stacks and trees. *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pages 155–64, 1986.