Grounding Predicate Symbols with Neural Networks

Richard Horvitz and Raj Bhatnagar

Abstract

We propose a cognitive architecture which integrates symbolic and connectionist representations. The symbolic representations, in the form of predicate logic, emerge from, and are grounded in, non-symbolic experience. This leads to multi-level representations, both symbolic and non-symbolic. This architecture is preliminary and is used as motivation for the methods described for grounding predicate logic representations in experience.

We first discuss a method of grounding descriptions in propositional logic and how that method might be extended to ground descriptions in propositional logic. This method fails to capture the idea of *object* so a more involved method is explored. We then discuss experimental results.

Introduction

Philosophers have pointed out the inadequacies of classical AI (Searle 1980) and of connectionist systems(Fodor & Pylyshyn 1988). In the first case, strong AI is allegedly refuted through the Chinese Room thought experiment, which suggests that computers embody only syntax but not semantics. A solution to this via symbol grounding has been proposed(Harnad 1990),(Harnad 2002). The symbols then are no longer meaningless tokens, but acquire meaning through experience. The debate still rages, and Searle is still convinced that he has refuted strong AI(Searle 2002). There are logical problems with the Fodor and Pylyshyn's argument, as noted by Chalmers(Chalmers 1990) but their criticisms are well noted. Typical connectionist systems, other than those contrived just to make the point(Plate 1995),(Smolensky 1990) do not exhibit systematicity, compositionality, and productivity of thought, which are features of symbolic reasoning.

As AI systems move toward more human-like activity, such as with flexible robotic systems, the need for grounding symbolic representations in experience has become clear(Jung & Zelinsky 2000). We go beyond this though and argue that such grounding is needed not only for communication but is essential for the system's own understanding of the world. In this paper we propose outlines of a cognitive architecture having these features:

- The symbolic representation of the world divides the non-symbolic world of experience into situations, objects and relationships.
- Neural networks are used to ground the meaning of symbols representing objects and relationships.
- The system can move its focus, evaluating what it senses with these grounding neural networks, filling out situations with objects and relationships it discovers.
- This search is guided by those situations that are currently active, which suggest the likelihood of particular objects and relationships occurring.
- Situations become active by the observation of objects and relationships that belong to them.
- Probes into the external world that successfully find known objects or relationships return a multi-level representation of that object, both symbolic and non-symbolic.
- Within a situation causal rules are learned (using neural networks) using all levels of these multi-level representations.

This model is not intended to be complete, but is given as a plausible structure illustrating the importance of the multi-level representations produced by the grounding networks discussed in this paper. First we will discuss some efforts at using neural networks to ground symbolic representations after which we will elaborate on the issues in the above list.

Neural networks and propositional logic

Modified back-propagation networks may be used to create descriptions in propositional logic. These networks are based on the fact that the various logical connectives may be represented algebraically. For example, $p \rightarrow q$ would be in algebraic form $1-p+pq$, with 1 representing true and 0 representing false. In these networks, the output neurons each have weights corresponding to the constants in the algebraic from of an expression in propositional logic. Suitable terms added to the error function cause these weights to converge to a valid logical formula as training proceeds. The layer below the output neurons is the *propositional* layer, where each neuron represents a propositional variable. This method is discussed at length in (Horvitz & Bhatnagar 2003).

These methods are of limited usefulness due to the weak representational power of propositional logic. In particular, it doesn't do anything to help the lack of systematicity attributed to neural networks.

Neural networks and predicate logic

These methods can be extended in a straight forward way to produce descriptions in predicate logic. In this scheme the propositional layer of the network now becomes the *predicate layer*. The layer below the predicate layer is the *object layer*, where groups of one or more neurons are taken to represent objects. A network of this type is illustrated in figure 1.

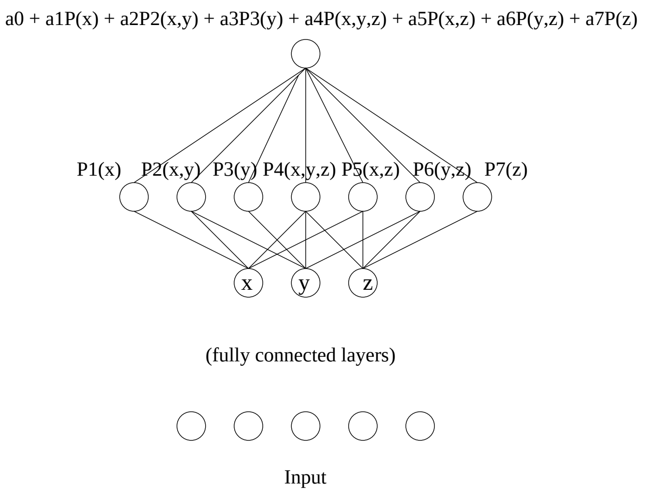
a0 + a1P(x) + a2P2(x,y) + a3P3(y) + a4P(x,y,z) + a5P(x,z) + a6P(y,z) + a7P(z)
P1(x)
P2(x,y)
P3(y)
P4(x,y,z)
P5(x,z)
P6(y,z)
P7(z)
x
y
z
(fully connected layers)
Input

Figure 1: Network for producing predicate logic descriptions

In this network the object layer and the predicate layers are not fully connected. Instead, each predicate neuron only sees input from the appropriate objects. As before, terms are added to the error function to encourage the outputs of the predicate neurons to take values close to 0 or 1. In this way, after training the weights connecting the output neurons to the predicate neurons will correspond to a logical connective, and the meanings of the predicate neurons and object neurons will be grounded in the weights below them.

This idea is flawed, however, as it does not sufficiently capture the concept of object. Some reasons that we recognize and name objects include

1. Objects exist independently of their perceptual background.
2. Objects persist in time. As our visual field changes (I speak of vision, but this applies to any of the senses), either through or own motion or changes in that which we are viewing, we can often still find that which we designate as an object.
3. Objects, perhaps in combination with other objects, have causal significance.
4. Objects recur in experience.

The plan described above for producing descriptions in predicate logic captures the fourth point here, as the system will identify as objects things that appear in multiple training examples. However, it fails on the the other points. In this work we are particularly focused on the first point. The above plan fails here because of its holistic approach to the input, where the neural network sees the entire scene at once. Back-propagation networks do not generalize well when images are scaled or rotated, let alone having major chunks moved around. The concept of object is closely tied up with the idea of focus. We can focus on an object and distinguish it from that which isn't the object. With this in mind we propose another way to ground the representation of objects and relationships using neural networks.

Ideally the system would work as follows. The system has three functional layers, each working with its neighbor or neighbors. Within each layer is a collection of neural networks. At the bottom layer the networks act as low level feature detectors. Local collections of features are then assembled into *candidate objects*. The second layer of networks is a collection of *object recognizer* networks. Each such network is sensitive to a particular type of object. It takes a candidate object as input, and if the object recognizer output is close to 1, then the candidate object is considered to be an object of that type. The third layer is a vector of *predicate networks*. For example, the vector might consist of networks representing the predicates $P_1(x), P_2(y), P_3(x,y), P_4(x,z)\cdots$. Each of these variables is typed, corresponding to a particular object recognizer network. The meanings of these predicates is formed during the training process, as are the logical connectives for the predicate formula.

The overall flow of the system is shown in figure 2.

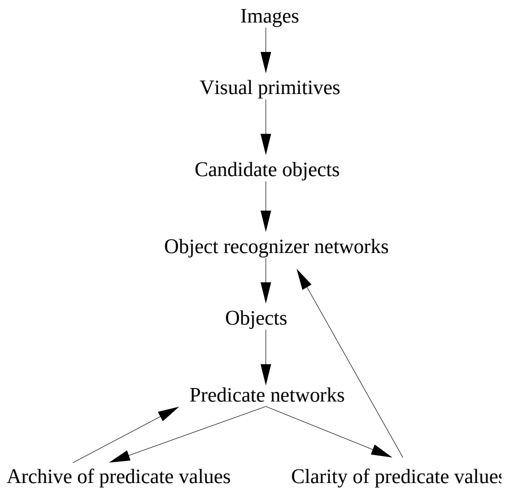
Images
Visual primitives
Candidate objects
Object recognizer networks
Objects
Predicate networks
Archive of predicate values
Clarity of predicate values

Figure 2: System for producing predicate logic descriptions

This figure represents the system as it exists, with some simplification from the ideal method described. The primary simplification is the preprocessing of images into collections of *visual primitives*, rather than having online search for objects directly from the image. Candidate objects are then constructed from groups of one or more visual primitive. This pool of candidates is built once when the main learning program starts. Each image is processed once per training epoch. Each time an image is processed the candidate objects associated with it are evaluated by the object recognizer network.

The system takes the set of objects that have been recognized by the object recognizer networks (those that returned the highest 'objectness' values) and enumerates all possible permutations of variable bindings in the predicate expression where the object types are correct. There are two possible methods to pick the best of the variable binding. One method is based on the *clarity* of the predicate vector. This is defined as:

$$clarity = \sum_i (1 - (P_i(1 - P_i)))/n$$

where P_i is the output if the i^{th} predicate network. This is a measure of how close to 0 or 1 all of the predicate networks are. If all of the networks could output exactly 0 or 1 then the clarity would take its maximum value of 1.

Alternatively the variable binding which produces the smallest error may be chosen. The system maintains an archive of the outputs of the predicate networks for all of the training examples over the previous training epoch. If the boolean pattern produced for a training sample is different from the boolean patterns for all of the archived examples having no class in common with the current example (e.g., for the current experiments, have no character in the image which is in the current image), then there isn't any error. Based on this the error is defined as:

$$E = \sum_i \prod_j (1 - (P_j - Q_{ij})^2)$$

where i indexes over all members of the archive having no class in common with the current example, j indexes over the predicate networks, P_j is the output of the j^{th} predicate network for the current example, and Q_{ij} is the output of the j^{th} predicate network for the i^{th} example from the archive. If one predicate in the current example differs by exactly one from that predicate in the archive (i.e. for one example it is true and for the other it is false) then the boolean patterns are different and there is no error contribution from that archive example.

The system can save the variable bindings that produced the clearest result, or the one that produced the smallest error, or it can save multiple variable bindings, so long the clarity of error is beyond some threshold value.

For an ideally trained system, the error value would approach 0 and the clarity value would approach 1 and these two methods would then be equivalent. However, early in the training this is not the case. For example, each predicate network could output the constant value of 1, and the clarity would be maximized, but so would the error.

There is no correct response for any of these networks. Instead they are trained as a kind of self organizing map. The system computes the derivatives $\frac{\partial E}{\partial P_i}$ for each predicate network, and can adjust the weights by the usual gradient descent method.

After the binding of objects which produces the clearest result is found, the predicate networks are trained, after which the object recognizer networks are trained. The clarity of the results of the predicate networks using objects approved by an object recognizer network is used to train that object recognizer network. The idea is that if the candidate objects approved by the object recognizers are very good quality, then the predicate networks will produce clear results. Thus the clarity of a predicate network is a measure of the 'objectness' of its arguments.

Since the clearest evaluation from the vector of predicate networks may involve more than one object of the same type (i.e., approved by the same object recognizer) the object recognizers are trained in batch mode, adjusting their weights once after being reevaluated for the relevant candidate objects.

There is no explicit representation of the logical connectives in the derived logical expression characterizing the various classes, but after the system is trained, the actual predicate logic formulas may be reconstructed from the values in the archive of results over the last epoch.

There is a possible issue in the treating of negation. For example, if the form of the predicate expression is $P_1(x), P_2(y), P_3(x, y)$ and for a particular example we have $P(x)$ is false, what does that tell us? Isn't there always some x such that $P(x)$ is false? If at the same time $P_3(x, y)$ is true, then it isn't an issue, but what happens if that predicate is also false? On the other hand, for any $P(x)$ another predicate $Q(x) = \neg Q(x)$ may be defined, in which case the issue does not seem to occur. Is one of these predicates correct in some sense? The solution is that we should not expect any of the difficult cases to ever occur.

In the simple experiments we are currently working on, we are using black and white images of text characters in various fonts as training data. When using line segments as visual primitives, a point is selected after which the longest line through that point is found having the same color as that point. We end up with two types of primitive, white black, each characterized by its length and angle. After a large collection of these primitives is found, the set is pruned with the object of spacing them uniformly over the image.

In using rectangular patches as primitives, a point is selected, after which the largest rectangle (at any angle) of approximately the same color as that point which contains the point is found. This method has several advantages. First, no distinction is made between black and white primitives and thus the method is more easily extensible to color images. Second, the data going to the object recognizer networks is the original image data.

Experiments

We generated a set of training data consisting of characters in six different fonts (using the letters A through H), each of a random font size between point sizes of 42 and 102. In addition they were rotated at a random angle, between 0 and 360 degrees. Another set of figures was constructed in the same way from a seventh font, to be used as a test set. Examples are seen in figure 3.

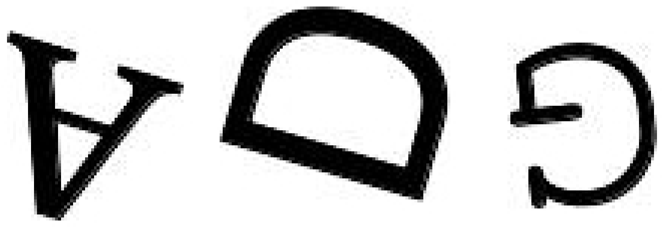

Figure 3: Some examples from the training set.

We have experimented with two types of visual primitives. The first of these is a *line* primitive. These are suitable only for black and white images as the primitives themselves are either black or white. Each primitive is found by selecting a point in the image and finding the longest line through that point of the same color as the point. One problem with this method is that it is difficult to find a good balance between adequately covering the image with primitives and not having too much duplication. Pruning algorithms which seek to space the primitives as far apart from each other were of some help, but the problem was not eliminated.

The second type of visual primitive used is the *patch* primitive. Each patch is an approximation of the largest rectangle around a point which is of approximately the same color as that point. The search procedure, which is somewhat time consuming, considers angles at small increments for each point. For each angle the system attempts to extend the rectangle in each of four directions by one row of pixels, going around and around until no more extensions are possible. The system then saves the largest such patch found. The large number of patch primitives found is reduced first by rejecting any that are too small in either dimension (with an arbitrary size threshold), and removing any patches which are contained in other patches. This method is more readily extensible to non black and white images than are line primitives. In addition, when using the patch primitives, the various neural networks in the system see the actual pixel data (interpolated as necessary into a fixed input size) rather than the converted form of data in the line primitives.

The efforts with the line primitives have been abandoned in favor of the superior patch primitives.

Local clusters of line primitives were assembled into candidate objects. This could also be done with the patch primitives, but for simplicity's sake each patch by itself was taken as a candidate object. The information seen by the neural networks consists of a ten by ten array of interpolated pixel data, along with the aspect ratio of the patch. No information about the absolute size or orientation of the patch is included. For predicates with two or more variables, information is also included about the relative orientation of the patches and the relative areas of the patches. Again, there is no information about absolute size or orientation. The result is that recognition of objects is independent of scaling or rotation. This is a key difference between this method and the simpler method mentioned earlier of using neural networks to create descriptions in predicate logic.

Experience has shown that saving (for the archive) the variable binding(s) giving the lowest errors works much better than saving the binding(s) giving the clearest results.

It is currently necessary to specify the form of the predicate vector. Experiments were run with various forms. In one case, five variables were defined, each of a different type (there are five different object recognizer networks). It is possible for an object to be accepted by more than one object recognizer network and thus be of more than one type. Ten predicates were defined as:

$$P_0(v,w,x),\ P_1(v,w,y),\ P_2(v,w,z),\ P_3(v,x,y),\ P_4(v,x,z),$$
$$P_5(v,y,z),\ P_6(w,x,y),\ P_7(w,x,z),\ P_8(w,y,z),\ P_9(x,y,z)$$

The system was trained until the mean error over an epoch was less than or equal to .001. The system was not making any classification errors on the training data. The test set was then evaluated. The results in this case were only moderately good, with two errors in eight cases. This seems to be a case of over-training, as earlier when the training error was only down to .0087, there were no test set errors (the test set was evaluated periodically). In any case, ten predicates were really too many and led to complex characterizations of the data. For example, the character 'A' was defined by

$$P_2(v,w,z) \wedge ((P_8(w,y,z) \wedge P_9(x,y,z) \wedge \neg P_3(v,x,y)) \vee$$
$$(P_7(w,x,z) \wedge \neg P_5(v,y,z) \wedge \neg P_9(x,y,z) \wedge \neg P_8(w,y,z))$$

the letter 'F' was described by the formula:

$$P_3(v,x,y) \wedge P_7(w,x,z) \wedge \neg P_9(x,y,z) \wedge (\neg P_6(w,x,y) \vee$$
$$\neg P_5(v,y,z)$$

Figure 4 shows the objects the system found used to recognize the characters illustrated earlier. Each rectangle represents a patch primitive, not drawn as the actual pixel pattern, but only as a rectangle of the mean color of the patch. In the first two cases the patches bring the character to mind, in the case of the 'G' it is more of a stretch, but plausible considering the task is only to distinguish it from the other letters between 'A' and 'H'.

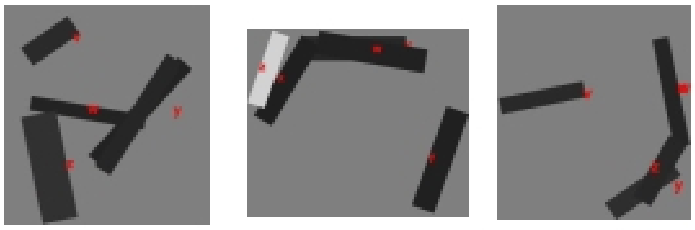

Figure 4: Objects used to recognize characters.

Open questions and next steps

There are several parameters in the system which have been assigned arbitrary values. These include the learning rates for the object recognizer networks and the predicate networks, a suitable error goal which will optimize generalization, the number and type of variables, the predicate form, the threshold of error or clarity for archiving variable bindings, and the number of objects that each object recognizer should retrieve. All of these issues require significant exploration.

In addition, the system needs to be applied to more interesting input. This will first include images containing multiple characters and then more complex color images. This will require either modifications of the visual primitive extraction method, or the addition of another layer of networks which will search for primitives.

Cognitive architecture

Symbols are not only useful for communication but are needed for understanding the world. Symbols are the lines we use to segment our representational world. If our only symbols were the tokens we use in language, then the grounding process would simply involve mapping these symbols to the world of experience, but when we take symbols as having much greater usefulness it is necessary for the system to create and ground its own symbols.

As indicated in the previous section, we view symbols as emerging from neural processing of non-symbolic information. In these networks, the output of the top two layers of neurons are taken as symbolic representations. The system moves the focus of its eyes (or whatever sensors are at its disposal), searching for recognizable objects and relationships. In this way, the neural networks are used as probes into the external world. If the object recognizer networks return values close to 1, indicating they have found an object, then a multi-level representation of that object is added to the activation of any relevant situations. The multi-level representation includes all values currently on any node of the neural network, from the non-symbolic (and specific) representation on the input terminals through the symbolic (and general) representation at the predicate neurons.

By situation we mean a collection of objects and relationships among those objects that commonly occur together, and have some causal significance. Within each situation exist another collection of neural networks, which learn the temporal/causal relationships between these multi-level representations. These networks take as input multi-level representations, both of cause and effect, and a time difference, and return a probability. Networks that work like the predicate grounding networks may also be used, either to ground new objects and relationships as causes or effects. In each case, weight decay should be promoted for the lower level levels of these networks, with the aim of achieving symbolic understanding.

Examples of situations might be ’driving a car down the street’ or ’reading email’. As the objects and goals in each are different, by this kind of separation problems are kept to a manageable size.

The activation of a situation based on the occurrence of objects or relationships belonging to it may also be represented as a predicate. Thus a situation arising may be a fact noted in other situations. If the system is planning within an active situation, these situations represented as predicates may be considered as subtasks.

Some situations may work in a mode which is virtually ungrounded, almost completely at a symbolic level, such as in doing mathematics.

Memories of specific events may also be at multiple levels, from the non-symbolic to the symbolic. The structure outlined here gives the opportunity to explore the interplay of these levels. For example, in humans, producing a symbolic description of a memory will sometimes block the non-symbolic memory, sometimes resulting in poorer performance (Melcher & Schooler 1996). Why this might occur could be investigated in the setting suggested here.

Obviously this presentation is lacking in detail and leaves many issues unexplored. We are hoping that it is the plausible beginning of a cognitive architecture, and that as such it illustrates the importance of multi-level representations, and the methods of grounding objects and relationships presented here.

References

Chalmers, D. 1990. Why fodor and pylyshyn were wrong: The simplest refutation. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* 340–347.

Fodor, J., and Pylyshyn, Z. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition* 28:3–71.

Harnad, S. 1990. The symbol grounding problem. *Physica D* 42:335–346.

Harnad, S. 2002. *Views into the Chinese Room*. Oxford University Press. 294–307.

Horvitz, R., and Bhatnagar, R. 2003. Neural symbol grounding. *Proceedings of the 14th Midwest Artificial Intelligence and Cognitive Science Conference*.

Jung, D., and Zelinsky, A. 2000. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots* 8:269–292.

Melcher, J. M., and Schooler, J. W. 1996. The misrememberance of wines past: Verbal and perceptual expertise differentially mediate verbal overshadowing of taste memory. *Journal of Memory and Language* 35(2):231–245.

Plate, T. 1995. Holographic reduced representations. *IEEE Transactions on Neural Networks* 6(3):623–641.

Searle, J. 1980. Minds, brains, and programs. *The Behavioral and Brain Sciences* 3.

Searle, J. 2002. *Views into the Chinese Room*. Oxford University Press. 51–69.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence* 46(1-2):159–216.