To appear in: Touretzky, D. S., Elman, J. L., Sejnowski, T. J., and Hinton, G. E. (eds.) (1990) Proceedings of the 1990 Connectionist Models Summer School. San Mateo, CA: Morgan Kaufmann.

Cognitive Map Construction and Use: A Parallel Distributed Processing Approach

Ronald L. Chrisley*
New College
University of Oxford
Oxford OX1 3BN
United Kingdom

Abstract

The Connectionist Navigational Map (CNM) is a parallel distributed processing architecture for the learning and use of robot spatial maps. It is shown here how a robot can, using a recurrent network (the CNM *predictive map*), learn a model of its environment that allows it to predict what sensations it would have if it were to move in a particular way. It is shown how this predictive ability can be used (via the CNM *orienting system*) to enable the robot to determine its current location. This ability, in turn, can be used, when given a desired sensation, to generate sequences of goal states that provide a route to a place with the desired sensory properties. This sequence is given to the CNM's *inverse model*, which in turn generates a sequence of actions that effects the desired state transitions, thus providing a sort of "content-addressable" planning capability. Finally, the theoretical motivation behind this work is discussed.

1 INTRODUCTION

The Connectionist Navigational Map (CNM) is a system being developed to impart to an autonomous robot the ability to map its spatial environment and use this map to navigate in that environment. The primary theoretical motivation for constructing the system is to understand how a robot can make the the transition from pre-conceptual to conceptual representations of space.[1]

The CNM is being designed with a Heathkit Hero 2000 robot as the intended testbed. The features of the Hero 2000 that are relevant to this paper are that it is autonomous and mobile, its primary sense is sonar range-finding, and its spatial environment is a floor of research offices in a modern building. It is this experimental situation that is implicit in the following discussion and illustrating examples. For instance, the notion of "sensations" in the case of Hero can be taken to be a vector of n numbers, the first indicating, roughly, the distance to the closest surface directly ahead, the next number indicating the distance to the closest surface at a heading of $\frac{2\pi}{n}$ radians to the right of front, etc. Also, it is important to the proper functioning of the CNM that the sensations to which the robot attends are static, i.e., the sensation vector for a given location in the world is constant.

There are three main components of the CNM system:

- the forward model, or *predictive map*;
- the ability to determine to which place on the map a particular sensation might correspond, or *orienting system*[2];
- the ability to generate the actions that will reach a desired state, which is provided by the *inverse model*.

Each of these components will be discussed in turn.

2 THE PREDICTIVE MAP

One can view the process of robot map construction as the learning of a model of the environment, such that once the model is learned, the robot can use it to predict what sensations would occur if it moved in a

*Also affiliated with Xerox PARC Systems Sciences Lab, 3333 Coyote Hill Road, Palo Alto, CA 94304. Email: chrisley@csli.stanford.edu

[1] See section 6.

[2] The usage of "orienting system" here should not be confused with the usage of "orienting subsystem" in Carpenter and Grossberg's various papers on Adaptive Resonance Theory, e.g. (Carpenter and Grossberg '87). "Orienting" is used here in a way similar to the way one does when talking of orienting oneself using a field map and compass, or the north star.

particular ego-centrically specified manner (e.g "rotate $\frac{\pi}{4}$ radians to the right", "move forward 10 feet").[3]

Of course, this will require the agent to have some kind of representation of its current location, since, in general, the mapping from actions to sensations is dependent on where one is in the world. That is, the mapping *sensations* × *actions* ↦ *sensations* is one-to-many, since more than one place can have the same sensory properties. Thus, the spatial environment, and therefore a model of it, can be seen instead as a function from current state and current action to predicted sensations. The input consists of a state vector, corresponding to the current location l of the robot, and a vector representing the move m being made. The output of the network is a vector that is supposed to be equal to the sensation vector the robot would receive from its senses if it were actually at the place that is reached by making the move m at location l.

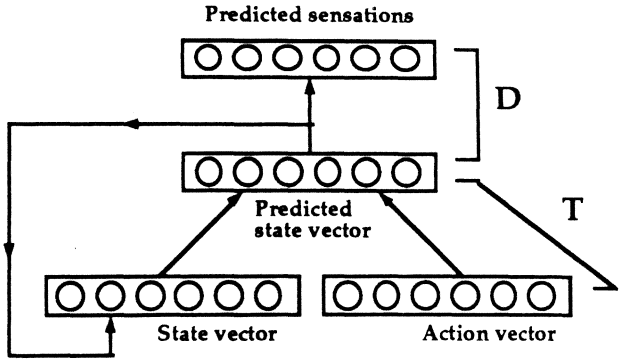

Figure 1: A PDP architecture for learning a predictive mapping (*states* × *actions* ↦ *sensations*) by composing a topological mapping T (*states* × *actions* ↦ *states*) with a descriptive mapping D (*states* ↦ *sensations*).

2.1 TOPOLOGICAL AND DESCRIPTIVE MAPPINGS

There is more structure to space than the mapping *states* × *actions* ↦ *sensations* indicates. Specifically, location and action determine a new location, which itself determines the sensations of the robot. Thus, it might be easier to learn a model if its structure reflects this regularity of the spatial environment. Consider the PDP architecture depicted in figure 1.

As said before, the input consists of a state vector and an action vector, the output is a vector that is supposed to be equal to the sensation vector the robot would receive from its senses if it were at the place that is reached by making the move at the location represented by the state vector. The network is a composition of two mappings: a T mapping and a D mapping (explained below). The recurrence between the hidden layer[4] and the state vector portion of the input layer allows this network to be a viable architecture for a predictive map. The activity pattern on the hidden units at time t is sent back to become the state representation in the input at time $t + 1$.[5] This imposes a relation on the patterns representing states, so that they become meaningful and useful. For example, if the robot considers moving forward and then turning right, it can use the map to predict what sensations it would have after those moves by calculating $D(T(T(l, \textbf{forward}), \textbf{right}))$, where l is a state vector corresponding to the robot's location before the moves, and D and T are the mappings indicated in figure 1. It is the recurrent connection in the network that allows this composition of the T mapping.

The mapping also becomes learnable, since all necessary inputs and outputs are provided or can be calculated.

Learning a model of the environment, then, can be decomposed into learning two mappings: a *topological* mapping T from states[6] and actions to states, and a *descriptive* mapping D from states to sensations.

[3]It should be made clear at the outset what is meant by a "map". "Map" is not taken to imply any kind of visual or imagistic representation. Rather, a map is any data structure that associates representations with actual locations, and with other representations that indicate the properties that that represented location has. Furthermore, the location representations are related to each other in a way that indicates what kind of motion would be required to get from one to the other. The point is that there isn't much weight being put on the word "map" itself, and if one prefers, one can just mentally replace it with "descriptive/topological data structure". The reason why this kind of data structure should be of any interest is brought out in section 2.1.

[4]In all discussions of network architectures in this paper, I suppress any mention of a hidden layer unless the activation patterns of that layer are used somewhere else in the network. For example, it is consistent with figure 1 (and is presumed by my discussion) that there may be a hidden layer between the input and output layers of the descriptive mapping, D.

[5]Such a use of recurrence has been suggested by (Jordan '86) and (Elman '88). A notable difference is that Jordan's network has recurrent links from the outputs, while Elman's network has recurrent links from the hidden units, as does the architecture in figure 1.

[6]I will hereafter use "state" instead of "location" not only because the network will have to represent orientations as well as locations, but because I want to allow activity patterns in the network to be interpreted in ways that do not involve the objective, absolute places that the word "location" connotes.

2.2 *A PRIORI* AND EMPIRICAL TOPOLOGIES

There are two approaches to having a mapping network learn a model of the environment. In one approach, the topological mapping is assumed to be known *a priori*, and the network only has to learn the descriptive mapping. The structure of space is known; all that remains to construct the map is to "fill it in."

In the second approach, no such *a priori* knowledge is assumed; the network must learn both mappings. Although there are many reasons why such a "tabula rasa" strategy would be more appropriate for the purposes of psychological modeling and philosophical understanding, one should not underestimate the advantages of an *a priori* topology for the purposes of engineering a working navigational system. The "empirical topology" that would result from learning both mappings would most likely fail to guarantee such essential properties as idempotency under the null action, invertibility, and composability, in the mappings it produces. For example, it would be possible for a network to learn a topology where the following are true:

$$T(x, \textbf{null}) \neq x;$$
$$T(x, a) = y \text{ but } T(y, a^{-1}) \neq x;$$
$$T(x, a) = y \text{ and } T(y, b) = z, \text{ but } T(x, b \circ a) \neq z;$$

where a^{-1} is the inverse of an action (moving back 2 feet as opposed to moving forward 2 feet), and $b \circ a$ is an action that is equivalent to performing a, then b.

Thus, PDP learning methods might produce topologies that can play an important role in explaining navigation behavior in animals, and in understanding concept acquisition,[7] but it is hard to see how they could produce a topological mapping that would be as accurate and general as a hand-crafted topology that had these three desirable properties, *inter alia*, built-in. This is not to say that such topologies could not be learned; but the development of such general topologies from experience alone, while theoretically of great import, and even potentially superior in terms of sheer performance, is not a prerequisite for a working system.

2.3 LEARNING THE TOPOLOGICAL AND DESCRIPTIVE MAPPINGS

The best strategy for learning the predictive map will depend on which of the two kinds of topologies, *a priori* and empirical, is being used. All the learning strategies considered in this paper assume the kind of learning situations available to back-propagation of error in multi-layer, feed-forward networks. Specifically, in learning a mapping $F : X \mapsto Y$, the network is presented with a number of samples, (x_i, y_i), where $x_i \in X$, $y_i \in Y$, and $F(x_i) = y_i$. This leaves much of the learning process unspecified (e.g., whether the samples are presented simultaneously, allowing epochal learning, or serially, requiring learning on a particular sample before the next sample can be examined), thus allowing for several distinct possible learning strategies. However, before discussing these different strategies, an observation needs to be made about the holistic nature of maps.

[7] An elaboration of such philosophical benefits can be found in section 6.

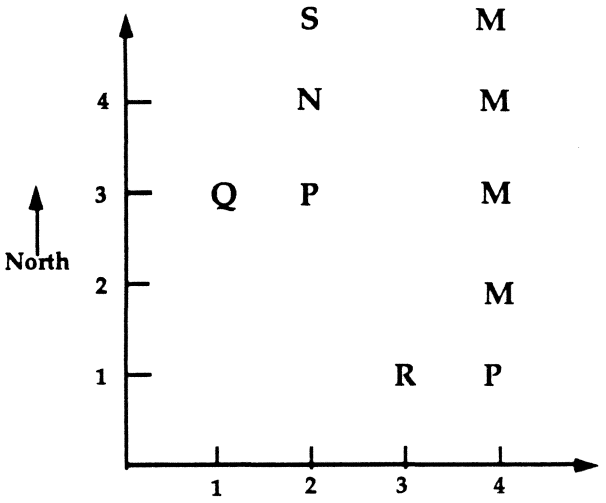

Figure 2: A schematic spatial environment used to illustrate the holistic nature of the *activity pattern* $\mapsto$ *world state* referential relation. Capital letters indicate the (idealized) sonar sensation at that location.

2.3.1 Maps are holistic

The place that a particular symbol on a conventional road map represents is not an intrinsic property of the symbol, but is determined by where the symbol is in some reference frame: how the symbol is spatially related to other symbols. Similarly, which world state a particular CNM activity pattern represents is not determined merely by the intrinsic properties of that pattern, or even by that pattern and the sensory properties assigned to it under the descriptive mapping; rather, a pattern's referent is also determined by its connections, under the topological mapping, to other activity patterns and their sensory associations given by the descriptive mapping.

For example, consider the spatial environment depicted in figure 2.

Note that in this toy world there are two qualitatively identical places, $(2, 3)$ and $(4, 1)$, each with the idealized sonar sensation P. Now suppose that there happen to be two activity patterns in the CNM, x_{10} and x_2, that, when input to the descriptive mapping, yield

P. That is:

$$D(x_{10}) = D(x_2) = P.$$

It is impossible to tell from this information alone which of the two places each pattern represents. Thus it would not be possible to determine what input pattern to give to the network in order for it to learn a mapping involving the place $(2,3)$.

But suppose that the topological mapping had the following properties:

$$T(x_{10}, \alpha) = x_3;$$
$$D(x_3) = Q;$$
$$T(x_2, \alpha) = x_4;$$
$$D(x_4) = R;$$

where α is an action that results in the robot moving to the west.

In this case, there would be an indication that x_{10} represents $(2,3)$ and x_2 represents $(4,1)$: the topological relations between patterns as defined by T helps determine the referents of the patterns themselves. But note that the four properties above do not conclusively determine the reference of x_{10} and x_2: the relation of x_{10} and x_2 to other states in the CNM might indicate just the opposite, with the above mappings being a local error. For example, we might have, for the north-bound action β:

$$T(x_i, \beta) = x_{i+1} \text{ for } 10 \leq i \leq 14;$$
$$D(x_i) = M \text{ for } 10 \leq i \leq 14;$$
$$T(x_2, \beta) = x_1;$$
$$D(x_1) = N;$$

which would indicate just the opposite: that x_2 represents $(2,3)$ and x_{10} represents $(4,1)$; the evidence for this conclusion would be even stronger if Q and R were very similar. Thus, not only do the referential properties of states depend on *both* the topological and descriptive mappings, but reference also depends on the *entire* mappings.

2.3.2 Empirical topologies: learning both mappings at once

As mentioned above, in the standard approach to getting a PDP network to learn a mapping, a database of input/output pairs for which the mapping holds is required. For the CNM predictive map, this would require a set of pairs where each pair comprises a state vector and an action vector as input, and a sensation vector as the desired output.

However, the spatial world does not provide an agent with explicit information as to which state it is in. A mapping network which does not assume an *a priori* correspondence between states of the network and states of the world will have to generate its own system of representation of world states. Thus, the point just made about the holism of maps is relevant in choosing a learning strategy. Since, as we saw, the representational relationship between an activity pattern and a world state depends on both the topological and the descriptive mappings, the pattern that corresponds to a particular world state will change as the two mappings in the network change. This means that the input/output mapping to be learned will be dynamic. The network's learning task will be a "moving target" situation[8]: the desired output for a given input will change as the weights of the network, and hence the mappings they realize, evolve.

Several standard approaches to collecting the input/output pairs for the learning process would be of little use then. For example one could not just select the samples on the basis of the world states one wants the network to learn about. It might be easy enough to get the desired output, but what should the state vector input be? One would have to come up with some way of determining, given a network and its current weights, what pattern represents a particular location. We have already seen that this procedure would be dependent on the state of the entire network, and that the topological evidence can be contradictory. It is very likely, then that the computational cost of the procedure would be prohibitively high. Furthermore, it would require some teacher to actually choose and collect this data.

An alternative strategy is to let the robot generate the samples itself by moving through the environment. That is, make the samples causally and temporally related to their predecessors and successors. The robot starts with an arbitrary state vector and makes some move. The desired output is the sensations it receives at the location reached by the move. Thus, the first training sample is created. Another action is chosen, and it, along with state vector on the hidden units, would become the input for the next sample, etc.

Note that this is not merely a matter of incorporating context. It is common, for example, in training networks for speech recognition or text-to-speech tasks to present to the network the samples as they are encountered in the environment, as this facilitates incorporating context effects. The network gets its input from a window on the input stream, and the window is moved down the input stream to provide another input. The actual order of the windowed input is irrelevant, however: the window could be placed anywhere on the stream to generate a useful input sample. But this *assumes* an input stream that exists independently of the network's states, whereas the strategy of using environment-determined inputs for the CNM is in order to generate such a stream. Nor is the importance of ordering related to superficially similar schemes that

[8] Both (Mikkulainen and Dyer '88) and (Chalmers '90) explicitly mention the "moving target" nature of their networks' learning situation. In both cases, the networks were successful in developing an appropriate representational system.

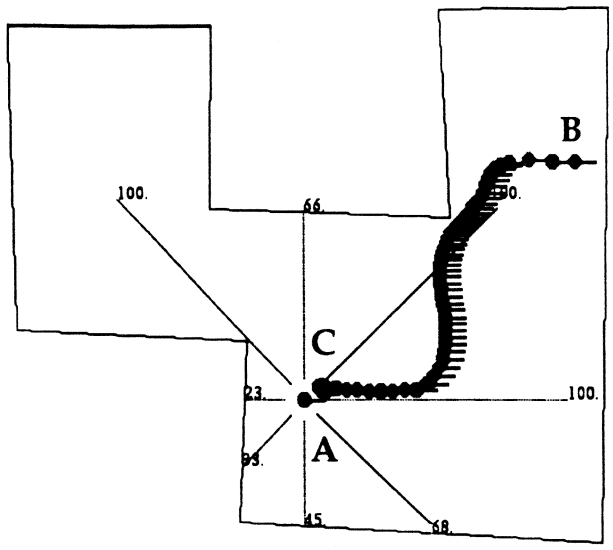

Figure 3: An illustration of the orienting system. A bird's-eye view of a two dimensional world is displayed. Rectilinear solid lines indicate walls. The robot is actually at location A, from whence it receives its sensory data (radial lines). Its state vector initially corresponds to location B. The state vector is iteratively modified so as to reduce the difference between expected and actual sensations. The trajectory of successive state modifications is shown, with the final fixed-point, C, being a rather good estimate of the actual location. (The orientation of the robot, which remained constant throughout all simulations described in this paper, is indicated by the horizontal bar extending from the robot's circular image.)

topological network is given a very good initial state, then perhaps one can have both a general and adaptive topology. Of course, simply allowing the weights of a "correct" topological network to be modified does not guarantee that it will successfully compensate for regular distortions in space, nor that it will optimally distribute its representational resources. Nor does it guarantee that the generality of the initial topological mapping can be maintained. Rather, these possibilities will have to be explored in future research.

3 THE ORIENTING SYSTEM

Two observations explain the need for an orienting system:

- Most means of using a map for navigation require the robot to know which map representation (a position on the paper in the case of a standard map; a state vector in the case of the CNM) corresponds to the current world state (location of the robot).
- Even with the best topological models and most accurate sensors, odometry-based location estimations drift away from the actual location quickly.

The orienting system, then, is a system that, given the robot's movement history, the predictive map, and recent sensory information, attempts to the find the activation pattern that corresponds to the current location.

3.1 STATE ESTIMATION: GRADIENT DESCENT IN ACTIVATION SPACE

Suppose the robot has constructed an acceptable, but not necessarily highly accurate, predictive map for a given region. But suppose, for some reason, the robot is in a different world state (p_{actual}) than the one that corresponds to the current state vector (x). Then it is no surprise that $D(x) \neq S(p_{actual})$, where S is the function from world states to the sensations they cause in the robot. Thus, there is error in the CNM: there is a difference between what is expected and what is observed.

Normally, error in a network is reduced or eliminated by modifying the weights of that network. But we are assuming that the map is more or less correct. *Ex hypothesi*, the source of error in this robot's CNM is its erroneous state estimation, not its predictive map as a whole. The way to reduce error becomes clear, then: use the error to modify the erroneous state vector, not the weights. Back-propagate the error signal from the output of the descriptive mapping network to the input, and change the activation patterns of the input units in proportion to the negative gradient of the error. Propagate this new input through the descriptive mapping again. If the output is still substantially different from what is observed, then back-propagate this error again, and so on.[11]

A state vector x_i is a fixed-point of this iterative process if $D(x_i) = S(p_{actual})$. That is, the network will, if the process converges,[12] find a state vector that is consistent with the current sensory data. This process is depicted in figure 3.

[11] The idea of using back-propagation to alter activations instead of weights has been considered before. (Williams '86) introduced the idea; (Mikkulainen and Dyer '88) used it to generate better internal representations; (Linden and Kindermann '89) and (Kindermann and Linden '90) provide good analyses of the procedure, and apply it to pattern completion problems, as well as other uses; (Chen, Belew, and Salomon '90) apply the idea to finding fixed points in iterative associative memories.

[12] A rigorous study of the convergence properties of the orienting system has not yet been performed, but informally tested cases generally converge, even with a poor predictive map.

improve learning times, accuracy, or convergence.

This iterative means of actually generating the training samples does not mean that learning has to be serial. One could, after collecting enough data, use an epochal learning strategy, in which all weight changes from each sample are calculated before any are made. But this would necessitate either a human assistant or prior navigation routines in order to generate actions for the robot while the data is being collected, since the CNM would not be able to be used for navigation until after the data for the epochal learning had been collected and some learning had taken place.

In a non-epochal system, the weights are changed after each sample presentation. (Jordan '90) has shown that it is possible in such cases to learn the inverse model at the same time as the forward model. In the case of CNM, this means that navigational abilities should be able to be learned at the same time as the predictive map. If so, the robot's navigational system could choose the actions to be taken, even at the very first stages of learning, as Jordan's systems do. Thus, no human teacher would be required; the system would be capable of generating all of its inputs and desired outputs.[9]

2.3.3 Learning with *a priori* topologies

There are two kinds of *a priori* topology: fixed and variable. In a fixed topology, the weights that implement the topological mapping are frozen[10], with learning in the CNM only changing the weights that implement the descriptive mapping; as said before, all that remains to construct the map is to "fill it in". A variable *a priori* topology, on the other hand, initializes the topological mapping to some ideal state, but allows subsequent modification and refinement on the basis of experience.

To learn a predictive map with a fixed *a priori* topology, one would initialize the robot's state vector to the one that corresponds, in the *a priori* topology, to the initial location of the robot. Then, the robot would chose a move to make. The topology is again consulted to yield the state vector that corresponds to the location that robot would occupy if it were to make that move. This vector is used as input to the descriptive mapping, which yields a predicted sensation as output. The robot makes the move, and uses the difference between what it actually observes and what it expected to observe as an error signal for weight change in the descriptive mapping network.

However, there are several reasons why an *a priori*, fixed topology might be unacceptable:

- **Error due to noise, failure, or unforeseeable events.** This isn't just a problem for fixed topologies: *any* means of predicting future states from the current one will be faced with the uncertainty and error that comes from being embedded in a complex physical world. This is addressed in the CNM by the orienting system, which refines the robot's estimate of its current location based on the coherence between the robot's current sensations and the sensory expectations provided by the predictive map. See section 3.
- **Systematic error.** General orienting and localization abilities are necessary, as just mentioned, but other, systematic sources of error could be handled with simpler methods in order to prevent an over-reliance on the orienting system, which could be computationally expensive or unreliable. Specifically, the CNM should be able to reflect the fact that the world *isn't* topologically uniform, but there are pockets of topological regularity in a heterogeneous space. For example, the topological mapping should be able to take into account that the kind of state transition that a move m effects in a room with thick carpeting is different from the transition effected by m in a tiled corridor, that floors can be sloped in some areas, etc. These deviations from a uniform topological mapping are systematic enough that they could be addressed directly by allowing slight modifications to the topological mapping, rather than relying exclusively on orienting strategies.
- **Limited resources.** A network has a finite quantity of representational resources, which it must distribute over space, which is infinitely dense. A fixed *a priori* topology will have a distribution of representational states over world states that will be independent of the qualitative character of those world states. For example, the amount of actual distance corresponding to the minimal effective difference between state vectors will most likely be constant in an *a priori* fixed topology. Even if it isn't constant, it certainly won't vary in a way related to the presence of obstacles, doors, walls, etc. in the environment. A more optimal use of the limited representational resources of a network would be to allow, e.g., a sparse representation of regions in which there are few or no obstacles, while regions around doors, or areas cluttered with permanent furniture are modeled in finer detail, with a smaller grain size. This requires an adaptive, variable topology.

The point is that the only major drawback of an empirical topology was that it seemed unlikely that a fully general topology could be learned through the weak method of error minimization. However, if the

[9] How the inverse model is to be learned and used for navigation is discussed in sections 4 and 5.

[10] Actually, in the case of a fixed topology, there seems to be no practical reason for having the topological mapping implemented in a PDP network at all.

3.2 SIMULATION DETAILS

To generate behavior like that illustrated in figure 3, a 2D world, with walls and a robot, was simulated. The sonar sensations of the robot at any given time were calculated by finding the distance to the nearest wall at each of the angles $\frac{2\pi k}{8}$; $k = 1, 2, \ldots, 8$ from the current orientation of the robot. Each component of the input vector was one of these distances, or 100, whichever was less (to reflect the range limitations of sonar).

A fixed *a priori* topology was used for the sake of simplicity and experimental control, so the robot only had to learn the descriptive mapping. Each sample in the training data consisted of an activation vector corresponding to a location, and the sensory input the robot had at that location. This data was collected by moving the robot around the environment, periodically storing the current location coordinates and the current sensations (distances at the eight angles). The coordinates were scaled to be within the range $0 \cdots 100$. Each scaled coordinate and (pre-scaled) distance was then translated into a four-unit activation vector using a representation scheme roughly based on one mentioned in (Hancock '88). A simplified description of the translation procedure is: if the value to be coded is $0, 33, 66$, or 100, then all the units have 0 activation except for the unit $1, 2, 3$ or 4, respectively, which has an activation of 1; if the value to be encoded is 0.25 of the way between 33 and 66 (i.e., 41.25), then unit 2 has an activation of 0.75 and unit 3 has an activation of 0.25, etc. These four-unit vectors were composed to make the appropriate input and output vectors (i.e., the input vector was composed of two four-unit vectors, one for each 2D coordinate, while the output vector had 32 units, 4 units for each of the 8 sensed distances). Roughly 200 samples were collected in this manner. The absolute orientation of the robot was kept constant.

The network was trained on this data, using Scott Fahlman's Quickprop algorithm (Fahlman '88), and his Lisp implementation of this algorithm. The network had 8 hidden units, and converged to within 1% of its final error within 500 trials.

After convergence, the behavior displayed in figure 3 was generated as follows. The robot was placed at the location A, and its current location coordinates were encoded into an input vector via the process described above. This input was forward-propagated through the network to produce an output vector, o. This was compared to the actual sensations the robot was receiving, s, to yield an error signal $\frac{1}{2}(s-o)^2$. This error was back-propagated in the normal manner (Rumelhart, Hinton, and Williams '86). The error signal at each input unit was normalized by dividing it by the number of weights from which the unit received an error signal. This error value was then multiplied by a scale factor of 0.2, and then added to the activation of each input unit. Activation values less than 0 or greater than 1 were truncated to 0 and 1, respectively. This process was iterated until a fixed-point was reached. Each change of input activation corresponds to a revision of the network's estimate of the robot's location, and is depicted in figure 3 by the chain-like succession of robot images.

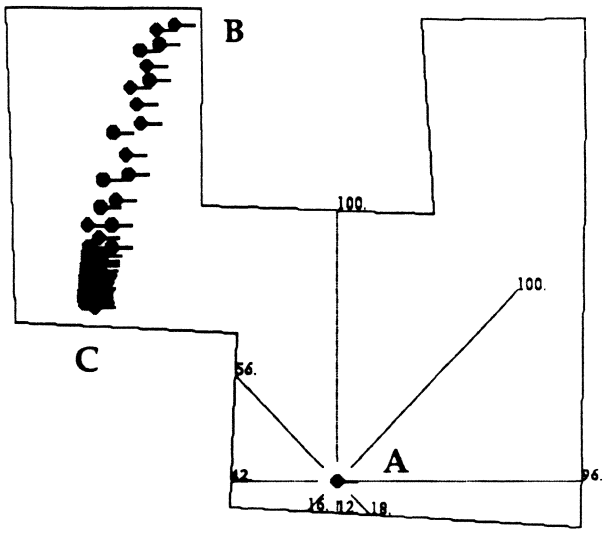

Figure 4: An illustration of how the simple orienting system is dependent on initial conditions. The system converges on the location (C), closest to the initial guess, that best matches the actual sensory data. The robot is roughly in the same actual location (A) as it was in figure 3, but since it has a different initial guess (B) as to where it is, the orienting system produces a different (and in some sense incorrect) estimate of the current location. This ambiguity can be overcome, if desired, by chaining constraints (see text).

3.3 OVERCOMING AMBIGUITY: CHAINING CONSTRAINTS

Of course, since any state vector x_i for which $D(x_i) = S(p_{actual})$ is a fixed-point of the orienting system, there is no guarantee that the state converged to corresponds to p_{actual}. Rather, the orienting system will (most likely) converge to the state vector, closest to the initial state, for which $D(x) = S(p_{actual})$. See figure 4, which was produced using the simulation just described for figure 3.

Note that even though this orientation scheme is susceptible to local minima, this does not mean that it can't be useful in navigation. For example, odometer drift is a constant problem, but it seldom happens that a robot's estimate is so wrong that the orienting system would converge to the wrong location. Thus, even the "nearest-neighbor" orienting system just described would be of use whenever the odometer drift

is bad enough to make corrections based on the predictive map to be of use (i.e., whenever the odometer drift is greater than the relatively high margin of error of the predictive map), but not so bad that the robot could drift into another sensory "well" between consultations of the orienting system.

Furthermore, the final state vector is not (necessarily) the closest in actual space, but in activation space. Thus, if the network uses a state vector space of a dimensionality higher than actual space, and it uses an empirical (or at least variable) topology, it might be able to construct the mapping between state vectors and world states to be such that minimizing distance in activation space would correspond to some interesting trajectory in real space. For example, the orienting system might converge to a state vector for which $D(x) = S(p_{actual})$, but that also has some higher-order similarity to the initial state, such as "being in a room with one exit", rather than being merely the closest in a raw spatial sense. Such possibilities are suggested by the analysis in (Servan-Schreiber, Cleeremans, and McClelland '89) of the internal representations of a network that learns a finite state grammar, and warrant further exploration.

These qualifications aside, it is nevertheless in general desirable to have an orientation system that will work even when the initial location estimate is very poor, and when the state vector topologies are of little use. In such a case, the orienting system can operate by *chaining constraints*.

To chain constraints for orienting, a cascade structure like the one in figure 5 is used. All D and T mappings are copies of the descriptive and topological mappings, previously learned. SV_1 is initialized to the initial state estimate, and the current sensations are stored at AS_1. All other values are as yet undetermined.

The basic orienting process occurs between SV_1 and AS_1: SV_1 is modified until PS_1 matches AS_1. Then a move is made, and the activation pattern for this move is imposed on M_1. SV_1 and M_1 are propagated to PSV_2, and SV_2 is initialized to PSV_2. The move at M_1 is made.

The sensations after the move are stored at AS_2, and the orienting process occurs between SV_2 and AS_2. If the first orienting process yielded an incorrect guess as to the location before the move, then there will most likely be a discrepancy between PSV_2 and SV_2. This is a source of error that can be back-propagated to the original SV_1, which is changed accordingly, and the entire process repeats from the beginning. Of course, one move may not be enough to disambiguate, or the initial guess might have been correct. In such a case, the process is extended to SV_3, and so on. In order to prevent memory overflow, older states, moves, and sensations can be forgotten.

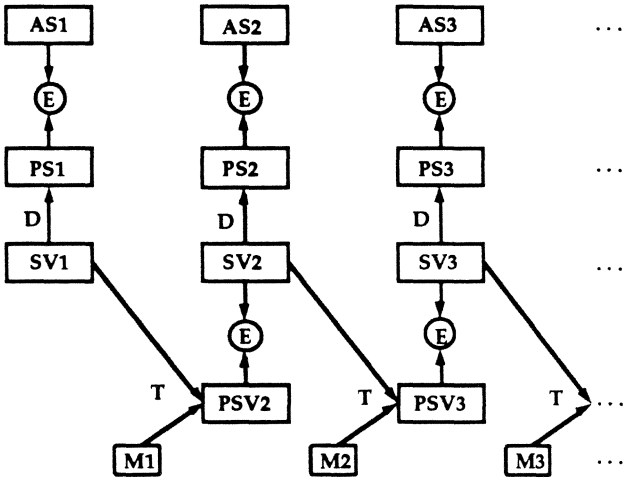

Figure 5: Using the method of chaining constraints in order to find the state vector that corresponds to the current world state. The modification to a state vector is determined not only by what error it yields under the descriptive mapping, but also by the error produced under the descriptive mapping by state vectors that follow and depend upon it. SV = state vectors; PSV = predicted state vectors; AS = actual sensations; PS = predicted sensations; M = moves; D = descriptive mappings; T = topological mappings; E = sources of error.

Simulations examining this more robust means of orienting are currently underway.

4 THE INVERSE MODEL

At least part of the task of navigation is this: given where I am and where I want to be, how do I get there? In the case of the CNM, this might correspond to the robot being given two state vectors, a and b, with a representing its current location and b representing its desired location, and the task being to come up with a move or set of moves that will take the robot from $R(a)$ to $R(b)$, where $R(x)$ is the world state to which x refers.

Of course, it would be very unlikely that this problem could be solved in general by a simple mechanism, for states that are arbitrarily distant from each other, separated by arbitrary obstacles. Rather, a simple mechanism can only provide moves that make transitions between relatively local states, and it must be up to some other system to find the sequence of states to be traversed. Such a system will be discussed in the next section; for now we will consider how the CNM could compute the function P : *states* $\times$ *states* $\mapsto$ *actions* for nearby states.

4.1 DERIVING THE INVERSE MODEL FROM THE PREDICTIVE MAP

There are several ways a robot could use the predictive map to compute the function P:

- *A priori* **solutions.** Any CNM with an *a priori* topological mapping might just as well have an *a priori* inverse topological mapping. For instance, if the *a priori* mapping used the Cartesian coordinate system to represent states, and had associated with each move a corresponding vector, $(\Delta x, \Delta y)$, that indicated the change of state that action produces, then it should be trivial to also have the inverse association, from the difference between the actual and desired state, to a move that makes that transition.[13]
- **Heuristic search.** The robot could input a into the map, and test, for various moves m, whether $T(a, m) = b$. If not, another m is tried. However, it is unclear what means of sampling the space of moves (other than obvious heuristics such as "If you have just tried move m, don't try m^{-1}") would make a feasible search strategy.
- **Gradient descent in activation space.** In a manner similar to the mechanism proposed for the orienting system, the inverse model could be computed by inputting the a and an initial move "guess" m to the mapping T. The error $T(a, m) - b$ is back-propagated to the input units, and the move vector is changed so as to effect a gradient descent reduction of error.[14]

Although this last idea seems very promising, and is currently under investigation,[15] the possibility of actually learning the inverse model directly (as opposed to being given it, or deriving it from the forward model) should be considered.

4.2 LEARNING THE INVERSE MODEL

(Jordan '90) discusses how a network might learn an inverse model similar to the kind we are considering. One major difference is that Jordan assumes that the state information is available, or in other words, that the sensory information also serves as the state information. Nevertheless, his analysis is appropriate as long as care is taken to remember these discrepancies.

One idea of how to learn an inverse model is to try to learn it directly, by training a network on input/output pairs of the form $(a, b)/m$. In Jordan's framework, the robot could generate these training samples by actually performing an action in the world and observing what states result. The initial and final state are used as the input, the action that led from one to the other is the desired output. But, as pointed out before, in the case of spatial mapping the state vector corresponding to a particular location is not directly observable, so the best a robot could do would be to generate the training samples by inputting a state a and a move m to the forward topological mapping (T), to produce $(a, T(a, m))/m$ as a training sample.

Jordan provided three reasons for rejecting this method:

- **One-to-many.** Since there is more than one way to move from one state to another, the training data for a direct inverse model is potentially one-to-many. Back-propagation handles one-to-many training situations by averaging the desired outputs, which in general will not be a meaningful solution. This might be able to be avoided by restricting the set of actions being considered in forming the inverse model. For example, one might want to restrict the possible state transitions learned to be those that can be achieved by a simple action: motion in a straight line for some small distance, or rotation clockwise or counterclockwise through less than π radians. This would make the *states* $\times$ *actions* $\mapsto$ *states* mapping one-to-one, so its inverse, the *states* $\times$ *states* $\mapsto$ *actions* mapping, would be one-to-one as well.
- **Not goal-directed.** Direct inverse modeling samples action space to generate its training samples, and, at least in the case of Jordan's applications, using this method does not guarantee that the network will learn the mapping for the "desired" inputs (state vectors) in which one is interested. (However, it is unclear whether the proper "goal" for the CNM inverse model is to learn about specific desired states or actions in general.)
- **No direct connection to the world.** Jordan warns about using the network's forward model as a means of generating training data. It is much better to use the world itself when one can. The problem is that the world does not directly provide the state information that is needed to generate the training data, so the network must rely on its own predictions of what state will result (but see below).

Jordan proposes instead a means of indirect inverse

[13] Of course, such a topology will have all the problems of inflexibility mentioned before; a variable, *a priori*, inverse topology might overcome some of these problems. The learning involved in such a topology will be similar to that in an empirical topology; see section 4.2.

[14] Although gradient descent (the third item) is, strictly speaking, a form of heuristic search (the second item), I mean to make a distinction here between heuristics on the (conceptual) level of *actions* (heuristic search) and those on the (non-conceptual) level of *activations* (gradient descent). For a discussion of the conceptual/non-conceptual distinction, see section 7 and (Cussins '90).

[15] This idea is also being investigated by others; see (Thrun, Möller, and Linden '90).

modeling, or *forward modeling*. The idea is to compose the inverse (T^{-1}) and forward (T) mappings,[16] and learn an identity mapping across this composition. A desired state vector b and current state vector a is input to T^{-1}, which yields an action vector m. In the cases that Jordan is considering the error $T(a,m) - b$ is ignored; rather, the action m is executed and the state vector b^* is obtained directly from the world via the senses. The error $b^* - b$ is back-propagated though the $T \circ T^{-1}$ composition, but only the weights in T^{-1} are modified.

In this manner, two of the above limitations of the direct inverse approach have been addressed. By composing the T^{-1} and T mappings, the inverse model is forced to converge to a *particular* inverse solution, not plagued by the one-to-many problem. Also, forward modeling is goal-directed in Jordan's sense, since the input to the system is a desired state (but again, it is unclear whether this is an important constraint for the CNM inverse model).

However, for the spatial map learning tasks relevant to the CNM, an unmodified forward modeling approach remains susceptible to the third criticism mentioned above: it is not directly connected to the world. There is no immediate way to observe what the actual world state is after making a move, so there is no b^* available. Thus, the network is forced to use its own prediction, b, as a training output, a method which Jordan warns us against. Modifications need to be made if the CNM is to use forward modeling to learn its inverse model.

One idea is to apply forward modeling to the composition $D \circ T \circ T^{-1}$. That is, the error that is back-propagated is $D(T(a,m)) - S(p_{actual})$, with S being the sensory function from world states to sensations and p_{actual} being the current location, as defined before in section 3.1. This allows the error to be calculated, but not at the expense of having the network generate its own desired outputs: the world actually provides the error signal.

Another way of addressing this problem is to use the orienting system described in section 3 to generate the training outputs for the composition $T \circ T^{-1}$. In this case, $\hat{b}$, a world-dependent estimate of b^* (as opposed to the completely model-dependent estimate, b) is indirectly inferred from the current sensations by running the orienting process on the current sensations, $S(p_{actual})$. The pattern b is input to D as an initial guess for $\hat{b}$. The error $D(\hat{b}) - S(p_{actual})$ is calculated and back-propagated to the input in order to make changes to $\hat{b}$ that result in a better estimate of b^*. This process will converge on a state vector that, according to the predictive map, represents the closest place that matches the current sensory data. This estimate, $\hat{b}$, is used as a training output (via the error term $\hat{b} - b$) for $T \circ T^{-1}$ with the input b. Thus, the network's model, $D \circ T$ is involved in the generation of the training output, but so is the actual world, in the form of $S(p_{actual})$.

[16] Although in some sense the inverse mapping is not a formal inverse of the forward mapping T, it will be denoted by T^{-1} here to emphasize the fact that $T \circ T^{-1}$ is an identity mapping.

5 NAVIGATION

The purpose of having an inverse model is that it can assist the robot in navigating its environment. As pointed out in the previous section, the inverse model can only be expected to generate actions for relatively close state transitions. In order to be of use, the inverse model must be given an appropriate sequence of states from some other system. This section looks at a few ways that such sequences could be generated and used.

5.1 STATE-BASED NAVIGATION

State-based navigation takes as input an ordered sequence of N state vectors s_i and generates from them an ordered sequence of moves m_i such that after executing the nth m, the robot will be in the world state $R(s_n)$ (or at least guarantees that after the execution of all of the m_i, the agent will be in s_N).

This type of navigation could be useful for following previously encountered routes. While defining a route, the robot can periodically store the current state vector, such that after reaching its destination, it will have stored a sequence of state vectors. Then, if it ever finds itself in one of those states later, and it wants to go to another one of those states, it can input the current and neighboring state into the inverse model, which will output an action to take it to that next state, and the process can be iterated until the destination is reached.

Of course, things don't always go as planned. The action performed, m, will not always bring the robot to the desired state, s_i. There are three basic strategies for dealing with this:

- The robot can plan the next step as normal, ignoring that it is not where it should be (this might be satisfactory for small errors).
- The robot can use the orienting system to find the current state and then plan to the next desired state, s_{i+1}.
- Or the robot can be more conservative, and assume that it must reach s_i before it can reach s_{i+1}. Therefore, it uses the orienting system to find the current state, but plans to the same desired state as before, s_i.

5.2 CONTENT ADDRESSABLE PLANNING: DESIRED SENSATIONS

The CNM can also be provided with a means of generating actions based on desired sensations, to result in what is effectively a limited sensation-based inverse model: when given the current state and desired sensation, an action is output that will lead to a (nearby) world state that has those sensory properties. This is done by using a (state-based) inverse model and the orienting system again. The input to the D mapping network is initialized to some guess b (obtained by calculating $T(a, m)$), and back-propagating the error $D(b) - d$, where d is a sensation vector representing the desired sensations, through the network to change b until it is a state vector that indicates the nearest place that has the desired sensory properties. At this point, $T^{-1}(a, b)$ should indicate a move that could be made to reach a place with the sensory signature d.

Another way to achieve a similar result is to use the orienting system in a novel way. Instead of finding the current location, given the current sensory data and the initial guess b, the orienting procedure can be used to find a route in state space from the current location to a place that meets a sensory description. Given some desired sensations d, the state vector representing the current location, a_0, is input to the descriptive mapping D, and the error $D(a) - d$ is used to change a_0 in a gradient descent fashion to a new vector, a_1. The two state vectors, a_0 and a_1 are then input into the inverse model to produce an action to effect the transition. The process is then repeated, with a_1 as the next input to D. If the scale factor (conventionally denoted by η) used in the back-propagation is small enough, $T^{-1}(a_i, a_{i+1})$ will in general be meaningful (i.e., it will denote a performable and correct move).

This process can be illustrated by re-interpreting figure 3. The sensations at A are desired. The current location is B, and the current state vector denotes B. The orienting process generates a sequence of states (the trail of states shown in the diagram), which can be input to the inverse model in a pair-wise manner to generate a sequence of actions that will reach location C. Again, things will not always go as planned, so the robot might want to periodically use the orienting system in its proper capacity to ensure that its current state vector continues to be an accurate representation of its location.

As it stands, this procedure is unacceptable, since there is no guarantee that the orienting system will not produce a trajectory that runs through a wall or other permanent obstacle. This can be addressed by adding terms to the error function that will penalize state vectors that represent world states close to walls, even if they are very similar to the desired perception. For example, in the cases being considered here, where the components of the sensation vectors indicate the distance from walls at various angles, the error for the ith output node could be defined as $(D(a)_i - d_i) + \frac{\alpha}{D(a)_i}$, since the reciprocal of $D(a)_i$ is an indication of how close one would be to a wall if one were in the location denoted by a. The constant α is used to scale the importance of the obstacle avoidance term: the higher α is, the further the robot will stay away from walls.

This procedure can be (loosely) called "content-addressable planning" since it generates a plan based upon a qualitative specification of a place, rather than a conventional address, a coordinate, of the place. As in state-based navigation, content-addressable planning can be used in conjunction with stored routes. The robot, while determining a route, can store various "scenes" or sensations along the way. At a later time, this route can be followed by inputting these stored sensations to the content-addressable planner, which will move the robot along the route. Depending on the nature of the robot's environment, this method might be more robust than state-based route following. The absolute state vector values stored might change their meaning between storage and use, due to a variable topology's compensation for a systematic change in the world, such as tread wear on the robot's wheels. Since the CNM assumes that a given location's sensory input to the robot is more or less constant, it might be best to remember a route based on these actual sensations, rather than coordinates in a varying topological code. Of course, even with a sensation-based route, errors will still creep in, but measures directly analogous to the three mentioned at the end of section 5.1 can be taken to help ensure that the destination is reached.

6 THEORETICAL MOTIVATION[17]

The primary theoretical motivation for constructing the CNM system is to understand how a robot can make the the transition from pre-conceptual to conceptual representations of space. The underlying hypothesis is that map construction and use is a paradigmatic case of concept formation, that the computational means underlying spatial concept mastery will be of a type similar to the means used for concept mastery in other domains. The preceding discussion and simulations were a start at an answer to the question: how might we use PDP to understand cognitive map construction and use? Now it is time to address the question: what could understanding cognitive map construction and use tell us about representation and the mind in general?

A complete account of cognition, natural or artificial, will have to be based on a theory of representation.

[17] Those readers of a less philosophical nature might want to skip this section and proceed directly to the conclusions in section 7.

Our theories will have to explicitly characterize the representations that play a role in cognitive systems as well as the content, or information, that is carried by these representations. Traditional (conceptual) ways of specifying content (such as in the ascription "The content of Leslie's belief is that this is a brown building") do so in such a way that requires that the agent to which the ascription is being made (Leslie) have the concepts used in the specification (e.g., **brown** and **building**). This means is inappropriate, then, for ascribing contents to simple systems that do not possess (many) concepts, and for many of our representational contents (such as perceptual and indexical ones) that are not conceptually mediated. We need a means (a non-conceptual means) of specifying contents that does not imply that the agent in question possesses the concepts used in the ascription. We also need a related account of the conditions under which it is appropriate to ascribe a particular content, so non-conceptually specified, to a particular physical system.

Consider the domain of spatial navigation, the ability to find one's way in the world. A plausible idea is that the first need, the need for a non-conceptual specification of the contents involved in navigation, can be answered by specifying contents in terms of an organism's abilities. For example, one might characterize the content of a particular computational state of a rat's brain in terms of the abilities it engenders, such as the ability to keep the end of the corridor in the center of its visual array. This would be a *non-conceptual* specification since, e.g., a rat doesn't need the concepts **corridor** or **visual array** in order to possess this ability or the contents which are specified in terms of it; and it would be a specification of *content*, since the rat certainly represents the world when it navigates: if the rat were exercising this ability in front of a life-size painting of a corridor, as opposed to an actual corridor, it would in some sense be wrong about the way it is taking the world to be, and being wrong about something is a good indication of the presence of representation.

Given this, one might think that the second need, the need for content ascription conditions, could be provided by any computational architecture that explained why a system possessed the abilities used in the content specifications. If, as in some cases seems likely, rats do not navigate by employing some concept of absolute location, but rather merely memorize the sequence of turns necessary to get from one place to another, then an account of how those abilities can be provided by a particular computational system might also provide the ascription conditions for the contents involved in rat navigation. But rats aren't the only animals that represent the world; we do, and oftentimes we do so conceptually, in terms of objective contents such as that carried by the concept **location**. Thus, there is a further requirement on our account of content, the requirement for an explanation of concept possession and acquisition: the transition from non-objective to objective ways of representing the world. This requirement imposes a constraint that restricts the class of explanatory computational architectures, a constraint that suggests that a clear member of the class would be a type of PDP network.

The reasoning behind this claim is as follows. Traditional, conceptual specifications of content are an attempt to characterize contents that are objective, contents that correspond to (could be specified in terms of) abilities that are perspective-independent, in that the abilities are appropriate in almost any context. An example of such an ability is using a conventional, paper-in-hand map: no matter where one is, or where one's destination, such a map indicates, at least roughly, in what direction one should go. But as we have seen, there are many contents that aren't objective or context-independent. These could be characterized in terms of abilities that are perspective-dependent, such as the ability to use a route-based map. Such an ability will only be of use if one is on the route in question, so unlike a conventional road map, its utility will depend on where one actually is. If we can provide a computational architecture that doesn't merely explain why a system has the abilities it does, but also explains why a system can move, via learning, from perspective-dependent abilities (such as route-based navigation) to perspective-independent abilities (such as full-blooded map-based navigation), then we can provide an explanation of how a non-conceptually characterized system can possess concepts. Analyzing a system non-conceptually allows us, unlike a purely conceptual analysis, to explain concept acquisition, since both the starting point (context-dependent contents) and ideal end (objective contents) of that process, as well as the interesting area in between, can be so analyzed.

We need, therefore, an architecture that renders explicit the non-conceptual contents in a system. PDP is a promising candidate for this because:

- PDP representations are of varying context-sensitivity and perspective-dependence,[18] thus making them more amenable to non-conceptual content specifications than are the representations in logical or symbolic architectures, which encourage a conceptualist, strictly objective interpretation.
- Conversely, conceptual specifications of content seem inappropriate for PDP representations, since it is typically difficult (hidden-unit analyses not withstanding) to isolate an aspect of a PDP network that corresponds to some objective feature of the task domain as registered by the theorist. In fact (to paraphrase Cussins), it may be that

[18] See, for example, Smolensky's "coffee" example in (Smolensky '88) and the discussion of it in (Cussins '90)

PDP needs non-conceptual analysis more than non-conceptual analysis needs PDP (Cussins '90, p. 431).

- PDP networks emphasize incremental learning, which is the kind of transformation on representations that would appear to be useful for a system to make the transition from perspective-dependence to perspective-independence, from context-sensitivity to objectivity, from mere representation to conceptual thought.
- Finally, the continuous mathematics and non-linearity of PDP networks provide for a flexibility in expressing relations and transitions between non-conceptual representations that would be lacking in a discrete, classical architecture that has been developed to express the relations between conceptual representations only.

These philosophical motivations depend heavily on the discussion in (Cussins '90).

7 CONCLUSIONS

It has been shown how a robot might, through interaction with its environment, acquire and maintain a map that allows it to predict what sensations it might have if it moved in a particular way.[19] Preliminary simulations indicate that the generalization properties of standard feed-forward networks trained with back-propagation are well suited to this spatial learning task, since a training set of only 200 points produced a network that could provide a reasonable estimate of the sensations to be found at an indefinite number of locations.

Work that remains to be done includes further simulations that vary orientation as well as location; simulations involving the chaining of constraints algorithm; simulations that test the relative feasibility of variable *a priori* vs. empirical topologies; and applications of all of these ideas to the actual Hero 2000 robot.

Acknowledgements

Special thanks to David Gurr, who played a major role in the birth of many of the ideas in this paper, to Mike Jordan, for valuable suggestions and encouragement, and to Dave Touretzky, for comments on a draft. Also thanks to Dave Chalmers, Adrian Cussins, Peter Dayan, Rainer Goebel, David Kirsh, John Woodfill, and Ramin Zabih for discussion and suggestions. This work was made possible by support from the Center for the Study of Linguistics and Information at Stanford University, the Systems Sciences Laboratory at Xerox PARC, and by grants from the San Francisco branch of the English-Speaking Union, the Chancellors of the UK Universities, and the Oxford University Overseas Student Support Scheme.

[19] (Mozer and Bachrach '89) present an alternative way of achieving this kind of functionality. However, their approach differs notably from the CNM: the action space is small and discrete (there is a weight matrix for each action); there is a prior, localist encoding of world states (one unit per state); and the back-propagation "unfolding in time" learning procedure (Rumelhart, Hinton, and Williams '86) is used.

References

Carpenter, G. A., and Grossberg, S. (1987) A massively parallel architecture for a self-organizing neural pattern recognition machine. In *Computer Vision, Graphics, Image Processing* **37**, pp. 54-116.

Chalmers, D. (1990) Syntactic transformations on distributed representations. To appear in *Connection Science.*

Chen, J. R., Belew, R. K., and Salomon, G. B. (1990) A connectionist network for color selection. In *The Proceedings of the International Joint Conference on Neural Networks, January 1990*, pp. 467-470. San Diego: IEEE.

Cussins, A. (1990) The connectionist construction of concepts. In M. Boden (editor), *The Philosophy of Artificial Intelligence*, pp. 368-440. Oxford: Oxford University Press.

Elman, J. (1989) Finding structure in time. Center for Research in Language technical report 8801. La Jolla: UCSD.

Fahlman, S. E. (1988) Faster-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski (editors) *The Proceedings of the 1988 Connectionist Models Summer School*, pp. 11-20. San Mateo: Morgan Kaufmann.

Hancock, P. J. B. (1988) Data representation in neural nets: an empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski (editors) *The Proceedings of the 1988 Connectionist Models Summer School*, pp. 11-20. San Mateo: Morgan Kaufmann.

Jordan, M. (1986) Serial order: a parallel distributed processing approach. Institute for Cognitive Science report 8604. La Jolla: UCSD.

Jordan, M. and Rumelhart, D. (1990) Forward models: supervised learning with a distal teacher. Submitted to *Cognitive Science.* Hillsdale, NJ: Lawrence Erlbaum.

Kindermann, J., and Linden, A. (1990) Inversion of neural networks by gradient descent. To appear in *Parallel Computing.*

Linden, A., and Kindermann, J. (1989) Inversion of multilayer nets. In *The Proceedings of the First International Joint Conference on Neural Networks.* San

Diego: IEEE.

Mozer, M. C., and Bachrach, J. (1990) Discovering the structure of a reactive environment by exploration. In D. Touretzky (editor) *Advances in Neural Information Processing Systems 2*, pp. 439-446. San Mateo: Morgan Kaufmann.

Mikkulainen, R., and Dyer, M.G. (1988) Encoding input/output representations in connectionist cognitive systems. In D. Touretzky, G. Hinton, and T. Sejnowski (editors) *The Proceedings of the 1988 Connectionist Models Summer School*, pp. 347-356. San Mateo: Morgan Kaufmann.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (editors) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pp. 318-362. Cambridge: MIT Press.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. (1989) Learning sequential structure in simple recurrent networks. In D. Touretzky (editor) *Advances in Neural Information Processing Systems I*, pp. 643-652. San Mateo: Morgan Kaufmann.

Smolensky, P. (1988) On the proper treatment of connectionism. In *Behavioral and Brain Sciences* **11**, pp. 1-74.

Thrun, S. B., Möller, K., and Linden, A. (1990) Planning with an adaptive world model. Internal report. German National Research Center for Computer Science, D-5205 St. Augustin, Postfach 1240, F.R.G.

Williams, R. (1986) Inverting a connectionist network mapping by back-propagation of error. In *The Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pp. 859-865. Hillsdale, NJ: Lawrence Erlbaum.