Additive Modular Learning in Preemptrons

Gregory M. Saunders, John F. Kolen, Peter J. Angeline, & Jordan B. Pollack
Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210
saunders@cis.ohio-state.edu
pollack@cis.ohio-state.edu

Abstract

Cognitive scientists, AI researchers in particular, have long-recognized the enormous benefits of modularity (e.g., Simon, 1969), as well as the need for self-organization (Samuel, 1967) in creating artifacts whose complexity approaches that of human intelligence. And yet these two goals seem almost incompatible, since truly modular systems are usually designed, and systems that truly learn are inherently nonmodular and produce only simple behaviors. Our paper seeks to remedy this shortcoming by developing a new architecture of Additive Adaptive Modules which we instantiate as Addam, a modular agent whose behavioral repertoire evolves as the complexity of the environment is increased.[1]

Introduction

One of the major conundrums of machine learning research, of both the symbolic and neural varieties, is how to produce systems which demonstrate complex cognitive behaviors starting from simple kernels. Simple learning systems, such as feed-forward networks end up with simple behaviors, so are really only theoretical signposts; complex learning systems which start with a large initial software investment, such as explanation-based learning (DeJong and Mooney, 1986; Mitchell, et al., 1986), beg the question of origin. Placing a simple system in a complex environment can work if the environment is non-threatening (Elman, 1988), but often the cost of engineering the environment is greater than that of engineering a working system.

This trade-off between complexity of *specification* and complexity of *environment* has been playing itself out in recent tensions in connectionism between simple systems which do not scale well versus complex (modular) systems whose origins are "not phylogenetically plausible". The current swing to automatic modularization is a response to this tension, but suffers from a lack of distributed control. For example, both Jacobs, Jordan, and Barto (1990) and Nowlan & Hinton (1991) rely on a centralized gating network to select the proper expert module. In the former, the network is given a "task bit" as part of its input, so that the proper expert module is effectively preselected by the input vector. Similarly, the latter permits different inputs to the expert and gating networks, simplifying the modularization process. Furthermore, neither architecture exploits the fact that the outputs of the gating network are continuous; instead, the interactions between modules are encouraged to be binary so that module i has no appreciable influence on the output when module j is active. In fact, Nowlan & Hinton's work, following Jacobs, et al. (1991) on which it is based, explicitly trains away these interactions.

An alternative approach to modularity is found in the design of autonomous robots, a historically nontrivial control task. Brooks (1986, 1991) offers a task-based subsumptive architecture which has achieved some impressive results. However, since machine learning is not up to the task of evolving these systems, engineers of artificial animals have embedded themselves in the design loop as the learning algorithm, and thus all components of the system, as well as their interactions, must be carefully crafted by the engineer (see, e.g., Connell, 1990).

Research aimed at replacing the engineer in these systems is at an early stage. For example, Maes (1991) proposes an Agent Network Architecture which allows a modular agent to learn to satisfy goals such as "relieve thirst"; however, she presumes detailed high-level modules (such as "pick-up-cup" and "bring-mouth-to-cup"), and her system learns only the connections between these modules. An earlier work that does not presume such an a priori modularization (Maes & Brooks, 1990) allows a six-legged robot to learn to walk, but there are no real modules in the final system. Beer and Gallagher (1991) attack this same problem of robotic mobility, but in a different way. They use a genetic algorithm (GA) that produces a robot which walks well (in simulation), yet they engineered the precise modularity of their system. Lin (1991) similarly presumed a detailed modularization and proceeded to learn each piece.

Thus, the researchers in artificial animals fall into

1. This research has been partially supported by ONR grants N00014-89-J-1200 and N00014-92-J-1195.

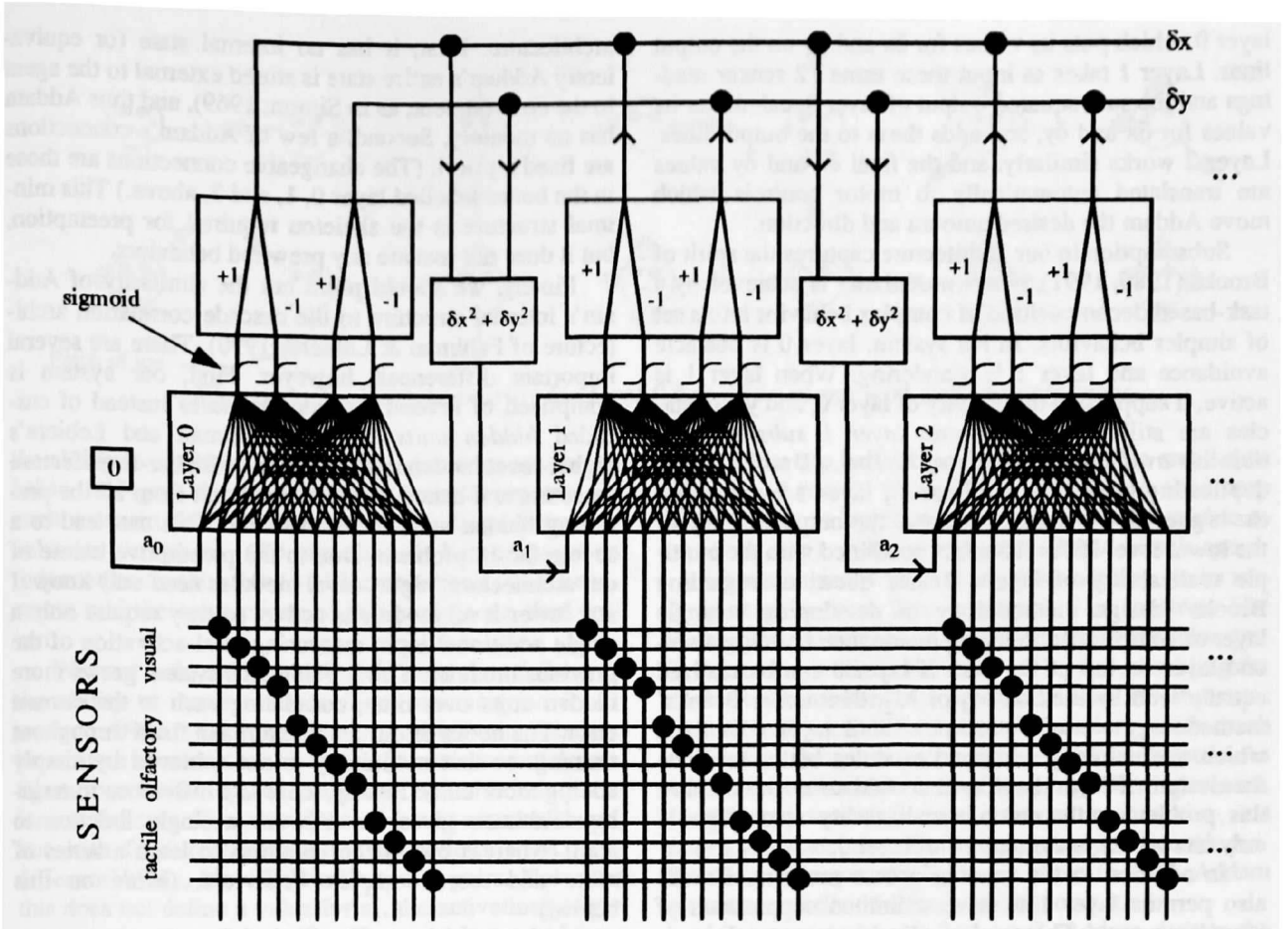

***Figure 1:** Addam's internal architecture.*

the same pitfalls as others who ignore the conundrum of machine learning: either their systems are too complicated to learn, their learning algorithms to simple to scale, or their modularization is arbitrarily indexed to the task.

In this paper, we present a novel approach to modularization, inspired by the work of Brooks, but tempered by the requirements of modular learning. As will be discussed below, our connectionist version of subsumption replaces Brooks' finite state automata (FSAs) with feedforward networks and additional circuitry, combined so that each module in our hierarchy respects the historical prerogatives of those below it, and only asserts its own control when confident.

Additive Adaptive Modules

Our control architecture consists of a set of Additive Adaptive Modules, instantiated as *Addam*, an agent which lives in a world of ice, food, and blocks. To survive in this world, Addam possesses 3 sets of 4 (noisy) sensors distributed in the 4 canonical quadrants of the plane. The first set of sensors is tactile, the second olfactory, and the third visual (implemented as sonar that passes through transparent objects). Unlike other attempts at learning that focus on a single behavior such as walking (Maes & Brooks or Beer & Gallagher, discussed above), we chose to focus on the subsumptive interaction of several behaviors, and hence Addam's actuators are a level of abstraction above leg controllers (similar to Brooks, 1986). Thus Addam is moved by simply specifying δx and δy.

Internally, Addam consists of a set of dynamical systems (instantiated as feedforward connectionist networks) connected as shown above in Figure 1. This architecture is actually quite simple. The 12 input lines are from Addam's sensors; the 2 output lines are fed into actuators which perform the desired movement (δx, δy). Note that we desire $\delta x, \delta y \in (-1, 1)$ so that Addam may move in the positive or negative direction. To keep the outputs in this range, we first tried using the hyperbolic tangent activation function (output range -1 to 1), but this was inadequate because it did not permit 0 as a stable output. We then switched to sigmoids (output range 0 to 1), necessitating the boxes with the fixed -1,+1 connections below. Thus the four outputs of each "Layer i" box represent $+\delta x$, $-\delta x$, $+\delta y$, and $-\delta y$, respectively. This system allows both positive and negative movement, as well as 0 as a stable output for any "Layer i".

Addam's movements are controlled by this system as follows. First, the 12 sensors are sampled and fed into

layer 0, which puts its values for δx and δy on the output lines. Layer 1 takes as input these same 12 sensor readings and the sum squared output of layer 0, calculates its values for δx and δy, and adds these to the output lines. Layer 2 works similarly, and the final δx and δy values are translated automatically to motor controls which move Addam the desired amount and direction.

Subsumption in our architecture captures the spirit of Brooks (1986, 1991), where modularity is achieved by a task-based decomposition of complex behavior into a set of simpler behaviors. In his system, layer 0 is obstacle avoidance and layer 1 is wandering. When layer 1 is active, it suppresses the activity of layer 0, and yet obstacles are still avoided because *layer 1 subsumes the obstacle avoidance behavior of layer 0*. Brooks avoids duplicating layer 0 as a subpart of layer 1 by allowing the higher layer random access to the outputs of any of the lower level FSAs. This fact combined with the multiple realizability of layers creates questions regarding Brooks' design methodology of developing a single layer of competence, freezing it, and then building a second layer on top of the first. If layer 0 can be realized equally well by method M_1 or M_2, then under Brooks' methodology we will not know until layer 0 is fixed which methodology's internal modules better facilitate the design of layer 1. Note that Addam does not have this problem with multiple realizability since layer 1 only has access to the *outputs* of layer 0.

In addition to the random access problem, Brooks also permits layer 1 to have unlimited suppression of layer 0's outputs. This works well when a human is engineering the robot, but such unbridled design-space freedom must be limited if we wish to have any chance of evolving the system. Thus Addam's different behavioral layers communicate only in the limited ways shown above.

Instead of being called subsumptive, our architecture is more aptly labeled *preemptive*. The modules are prioritized such that the behaviors associated with the lower levels take precedence over those associated with the higher levels. This is reflected architecturally as well as functionally, so that higher-level modules are trained to relinquish control if a lower-level module is active. For example, suppose that layer 0 behavior is to avoid predators, and layer 1 behavior is to seek out food. In the absence of any threatening agents, layer 0 would remain inactive and layer 1 would move Addam towards food. However if a predator suddenly appeared, layer 0 would usurp control from layer 1 and Addam would flee.

Note that we could have avoided feeding the sum-squared activation line into each module M_i by gating the output of M_i with the sum-squared line. We did not do this because our architecture is more general in that gating can be learned as one of many behaviors by each M_i. Our goal was to have each module decide *for itself* whether it should become active – had we used gating, this decision would have been made by M_i's predecessors.

A few more things should be noted about Addam's architecture. First, it has no internal state (or equivalently Addam's entire state is stored external to the agent in the environment, as in Simon, 1969), and thus Addam has no memory. Second, a few of Addam's connections are fixed a priori. (The changeable connections are those in the boxes labelled layer 0, 1, and 2, above.) This minimal structure is the skeleton required for preemption, but it does not assume any prewired behaviors.

Finally, we should point out the similarity of Addam's internal structure to the cascade correlation architecture of Fahlman & Lebiere (1990). There are several important differences, however. First, our system is comprised of several cascaded *modules* instead of cascaded *hidden units*. Second, Fahlman and Lebiere's higher-level hidden units function as higher-level feature detectors and hence must receive input from all the preceding hidden units in the network. This can lead to a severe fan-in problem. Due to the preemptive nature of our architecture, higher-level modules need only know if any lower-level module is active, so they require only a single additional input measuring total activation of the previous modules. Third, Fahlman's system grows more hidden units over time, correlating each to the current error. The nodes of our architecture are fixed throughout training, so that modularity is not achieved by simply adding more units. Finally, there is a difference in training: Fahlman gives his network a single function to learn, whereas our system attempts to learn a series of more and more complex behaviors. (More on this below.)

Training Addam

As mentioned above, Addam's environment consists of three types of objects: ice, food, and blocks. Ice is transparent and odorless, and is hence detectable only by the tactile sensors. Blocks trigger both the tactile and visual sensors, and food emits an odor which diffuses throughout the environment and triggers the olfactory sensors. Addam eats (in one time step) whenever it comes into contact with a piece of food.

Addam's overall goal is to move towards food while avoiding the other obstacles. This makes training problematic – the desired response is a complex behavior indexed over many environmental configurations, and yet we do not wish to restrict the possible solutions by specifying an entire behavioral trajectory for a given situation. Beer & Gallagher (1991) attempted to solve this problem by using GA's, which respond to the agent's overall performance instead of to any particular movement. We take a different approach, namely, we train Addam on *single moves* for a given number of scenarios, defined as one particular environmental configuration. Under this methodology, the *extended moves* which define Addam's behavior emerges from the complex interactions of the adaptive modules and the environment.

Training begins with level 0 competence, defined as

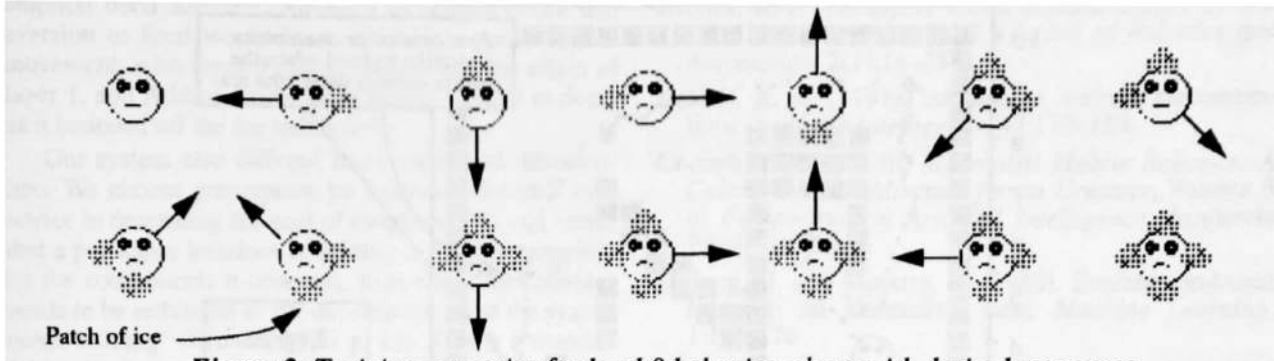

***Figure 2:** Training scenarios for level 0 behavior, along with desired responses.*

the ability to avoid ice. The training scenarios are shown below in Figure 2, along with the desired response for each scenario. Module 0 can successfully perform this behavior in about 600 epochs of backpropagation (adjusted so that the fixed +1/-1 connections remain constant), and the connections of this module are then frozen.

We next train Addam on level 1 behavior, defined as the ability to move towards food, *assuming no ice is present*. Once again, training is problematic, because there are a combinatorial number of environmental configurations involving food and ice. We solve this problem as follows. First, we define 14 scenarios as above, but with food replacing ice. This defines a set S of {(SensorValues, MoveToFoodOutput)} pairs. Note that this does not define a value for a_1, the activations of the system prior to module 1. (See Figure 1.) Instead of forcing module 1 to recognize the presence of ice, we assume that module 0 is doing its job, and that when ice is present a_1 will be >> 0. This allows us to define a training set T for level 1 behavior by prepending the extreme values of a_1 to the SensorValues in S, thus doubling the number of configurations instead of having them grow exponentially:

T={ {(0-SensorValues, MoveToFoodOutput)},
{(1-SensorValues, ZeroOutput)}}

Thus layer 1 (which is initially always active) must learn to suppress its activity in cases where it is not appropriate. This training method was motivated by studies on development in which a dynamical system had to learn to suppress its behavior when not appropriate (Thelen, 1990).

After level 1 competence is achieved (about 3500 epochs), a training set for level 2 competence (avoid blocks) is obtained in a similar manner. Note again that this avoids the combinatorial explosion of specifying the many possible combinations of ice, food, and blocks. Level 2 competence is achieved in about 1000 epochs.

Results

Once Addam was trained, we placed it in the complex environment of Figure 3. Its emergent behavior is illustrated in the top half of the figure, where the small dots trace out Addam's path. Each dot is one time step (defined as one application of the trained network to move one step), so the spacing indicates Addam's speed.

Addam begins at (3.5, 1) touching nothing, so its tactile sensors register zero and layer 0 is inactive. The olfactory sensors respond slightly to the weak odor gradient, causing a slight activation of layer 1, disabling the block-avoidance behavior of layer 2. Thus we observe a constant eastward drift, along with random north-south movements due to the noise inherent in the sensors. As Addam approaches the food, the odor gradient increases, the olfactory sensors become more and more active, and layer 1 responds more and more strongly. When the random noise becomes negligible at about (6.5, 1), Addam speeds up, reaches the food, and devours it.

After completing its first meal, Addam detects the faint odor of another piece of nearby food, and once again layer 1 controls its movement. However, at about (9, 5.5) Addam's tactile sensors detect the presence of a piece of ice, activating layer 0, and usurping control from layer 1. In other words, Addam's aversion to cold feet overcomes its zealous hunger, and it moves southeast. After "bouncing off" the ice, the tactile sensors return to zero, and layer 1 regains control, forcing Addam back towards the ice. However this time it hits the ice just a little farther north than the last time, so that when it bounces off again, it has made some net progress towards the food. After several attempts, Addam successfully passes the ice and then moves directly towards the food.

To reach the third piece of food, Addam must navigate down a narrow corridor, demonstrating that its layer 1 behavior can override its layer 2 behavior of avoiding blocks (which would repel it from the corridor entrance). This is shown even more directly in Addam's docking behavior (cf: Lin, 1990) as it eats the fourth piece of food. After finishing the last piece of food, Addam is left near a wall, although it is not in contact with it. Thus both the tactile and olfactory sensors output zero, so both layers 0 and 1 are inactive. This allows Addam's block avoidance behavior to become activated. The visual sensors respond to the open area to the north, so Addam slowly makes its way in that direction. When it reaches the middle of the enclosure, the visual sensors are bal-

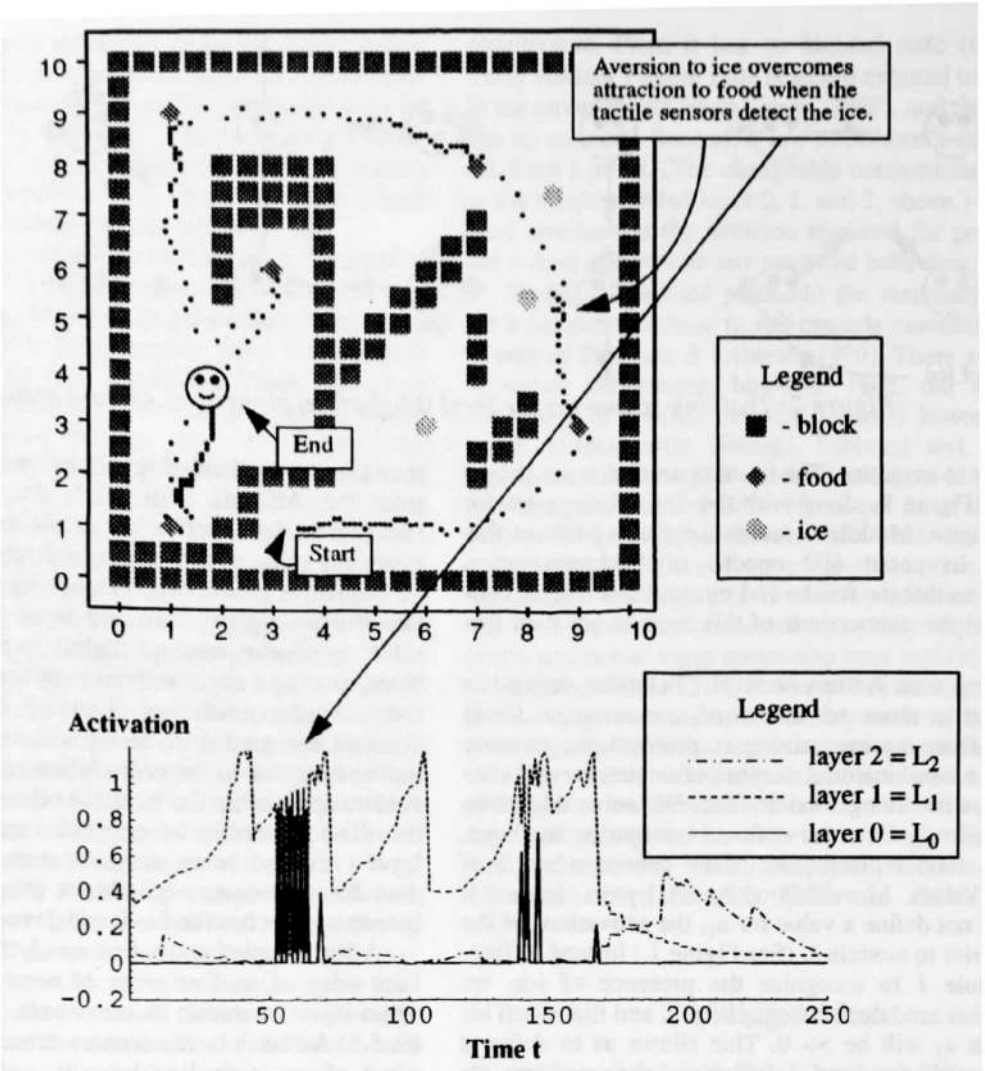

Figure 3: *Addam's emergent behavior in a complex environment, with graph showing the activations of layers 0, 1, and 2.*

anced and Addam halts (except for small random movements based on the noise in the sensors).

The bottom half of Figure 3 shows the activation of each layer i of the system (where the activation of layer i is $\|(\delta x,\delta y)_i\|$, the norm of layer i's contribution to the output lines). L_0 is generally quiet, but becomes active between time t=52 and t=64 when Addam encounters an ice patch, and shows some slight activity around t=140 and t=168 when Addam's tactile sensors detect blocks. L_1 ("approach food" behavior) is active for most of the session except when preempted by the "avoid ice" behavior of L_0, as between t=52 and t=64. The 5 peaks in L_1's activity correspond to Addam's proximity to the 5 pieces of food as it eat them; when the last piece of food is consumed at t=164, L_1's activity begins to decay as the residual odor disperses. Finally, we see that L_2 ("avoid blocks" behavior) is preempted for almost the entire session. It starts to show activity only at about t=160, when all the food is gone and Addam is away from any ice. The activity of this layer peaks at about t=190, and then decays to 0 as Addam reaches the center of its room and the visual sensors balance.

Remarks

Addam was trained on only 42 simple scenarios, yet it was able to perform well in a complex environment. Unlike other connectionist modular systems, our method of control is distributed – each module decides for itself whether it should exert control in any given situation. Furthermore, there is no gating network which receives a specialized task bit – Addam has three sets of sensors all treated equally and must learn the proper behavior on the basis of these inputs. Finally, instead of limiting activations of the modules to being 0 or 1, we exploited the underlying connectionist nature of our architecture, allowing us to produce interactions between modules more interesting than absolute preemption. For example, the presence of ice overrode Addam's attraction to food, yet Addam's "go-get-it" response to the food had a slight influence on its "runaway" response to the ice. Had pre-

emption been absolute, Addam's attraction to ice and aversion to food would have alternately controlled its movement, with layer 0 exactly countering the effect of layer 1, and Addam would have slowly starved to death as it bounced off the ice indefinitely.

Our system also differed from traditional subsumption. We choose preemption by following Brook's own advice in describing his goal of simplicity: "If you notice that a particular interface is starting to rival in complexity the components it connects, then either the interface needs to be rethought or the decomposition of the system needs redoing" (Brooks, 1986, p. 15). This is a wonderful credo for engineers, but as cognitive scientists, we must generalize it to this: If you notice that your model of a particular aspect of cognition starts to rival in complexity the components of the underlying system, then both the underlying system and the model of the environment need to be reexamined. Learning to adapt to the environment is extremely difficult in Brooks' subsumption architecture, but became possible after switching to a simplified, additive model of modularity.

Our ideas for modular adaptive control are independent of the internal structure of the modules. In fact, the work of Beer & Gallagher or Maes & Brooks is really complementary to ours, for although Addam's modules were instantiated with feedforward networks trained by backpropagation, they could have just as easily been trained by either GA's or correlation algorithms. Moreover, feedforward networks need not have been used either. We could have substituted sequential cascaded networks (Pollack, 1987), endowing Addam with internal state (cf Kirsh, 1991) and allowing even more complex behaviors.

Finally, we note a significant difference in methodology between our work and that of Brooks. In creating his agents, Brooks first performs a *behavioral* decomposition, but in implementing each layer, he performs a *functional* decomposition of the type he himself warns against (Brooks, 1991, p. 146). In training Addam, on the other hand, we first perform a behavioral decomposition, and then let backpropagation decompose each behavior appropriately. This automation significantly lessens the arbitrary nature of behavior-based architectures which has thus far limited the import of Brooks' work to cognitive science.

Acknowledgments

The authors thank David Stucki and Barbara Becker for their critiques of earlier drafts of this paper.

References

Beer, R. D. and Gallagher, J. C. (1991). Evolving dynamical neural networks for adaptive behavior. Technical Report CES-91-17, Case Western Reserve University, Cleveland.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Brooks, R. A. (1991). Intelligence without representations. *Artificial Intelligence*, 47:139–159.

Connell, J. H. (1990). *Minimalist Mobile Robotics: A Colony-style Architecture for an Creature*, Volume 5 of *Perspectives in Artificial Intelligence*. Academic Press, San Diego.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: an alternative view. *Machine Learning*, 1:145-176.

Elman, J. L. (1988). Finding structure in time. Technical Report CRL 8801, University of California, San Diego.

Fahlman, S., and Lebiere, C. (1990). The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, Pittsburgh.

Jacobs, R. A., Jordan, M. I., and Barto, A. G. (1990). Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15:219-250.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). "Adaptive mixtures of local experts", *Neural Computation*, 3(1):79-87.

Kirsh, D. (1991). Today the earwig, tomorrow man? *Artificial Intelligence*, 47:161–184.

Lin, L. J. (1990). Programming robots using reinforcement learning and teaching. In *Proceedings of AAAI-90*. pages 781–786, Menlo-Park, CA. AAAI, MIT Press.

Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth National Conferences on AI*, pages 769–802. AAAI-90.

Maes, P. (1991). The agent network architecture. In *AAAI Spring Symposium on Integrated Intelligent Architectures*, March.

Mitchell, T., Keller, R., and Kedar-Cebelli, S. (1986). Explanation-based generalization: a unifying view. *Machine Learning*, 1:47-80.

Nowlan, S. J. and Hinton, G. E. (1991). Evaluation of adaptive mixtures of competing experts. In Lippmann, R., Moody, J., and Touretzky, D., editors, *Advances in Neural Information Processing 3*. Morgan Kaufmann, pages 774-780.

Pollack, J. B. (1987). Cascaded back propagation on dynamic connectionist networks. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 391-404, Seattle.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II – recent progress. *IBM Journal of Research and Development*, pages 601-617.

Simon, H. (1969). *Sciences of the Artificial*. MIT Press.

Thelen, E., Dynamical systems and the generation of individual differences. In Columbo, J. and Fagen, J. W., editors, *Individual Differences in Infancy, Reliability, Stability, and Prediction*, Lawrence Erlbaum Associates, pages 19-43.