The Importance of Leaky Levels for Behavior-Based AI

Gregory M. Saunders, John F. Kolen, and Jordan B. Pollack
Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210 USA

saunders@cis.ohio-state.edu
kolen-j@cis.ohio-state.edu
pollack@cis.ohio-state.edu

Abstract

From the many possible perspectives in which an agent may be viewed, behavior-based AI selects observable actions as a particularly useful level of description. Yet behavior is clearly not structure, and anyone using behavior-based constraints to construct an agent still faces many implementational roadblocks. Such obstacles are typically avoided by adopting a finite state automaton (FSA) as a base representation. As a result, potential benefits from alternative formalisms are ignored. To explore these benefits, our work adopts a multi-level view of an agent: behaviors and FSAs are but two of many levels of description. We still focus on behaviors for the expression of design constraints, but we avoid using FSAs as an implementation. Our particular agent, Addam, is comprised of a set of connectionist networks, a substrate which promotes the automatic design of subsumptive systems. Moreover, the implementational choice has important behavioral consequences – some complex behaviors emerge due to interactions among networks and need not be specified explicitly. In this way, the underlying layers *leak* into one another, each affecting the others in subtle and desirable ways.

1 Introduction

Historically, AI has viewed agents from the Knowledge Level (Newell, 1982), in which an individual is characterized by its knowledge, goals, and rationality.[1] The abstract nature of this level has been called into question from many different directions: e.g., connectionism (Hinton et al., 1986; McClelland et al., 1986), situated action (compare Vera and Simon, 1993, with Agre, 1993), the observers' paradox (Kolen and Pollack, 1993, to appear), and others (e.g., Searle, 1993). Most recently, those studying the simulation of adaptive behavior have stressed that intelligence should not be viewed simply as knowledge and goals held together with procedural glue; there is much to learn from studying intelligence through self-sufficient agents competent to exist in the world (Meyer and Guillot, 1991; Wilson, 1991).

Yet we often forget that agents can be viewed at multiple levels of description, and as Chandrasekaran and Josephson (1993) point out, there is no single level of description which captures all aspects of an agent's behavior. To borrow their example, a simple coin sorter can be described as an abstract machine which classifies coins based on their weight and diameter, but if a lever jams, then the physical nature of the device becomes particularly important. Chandrasekaran and Josephson propose that agents be described by a set of "leaky levels," where each level of description contributes to the overall story of agent behavior, but the total picture arises due to the way the various levels interact.

The lesson is an important one, but it fails to address an important question: How does the recognition of multiple levels of description help one to implement an intelligent agent? In particular, how should one approach the task of constructing an agent which satisfies multiple behavioral constraints?

Brooks (1986, 1991) proposes an interesting answer to this question. Rather than observing a set of behavioral constraints and reasoning "The agent must have functional modules for perception, planning, etc.," one can remain more faithful to the actual observations by constructing an agent which satisfies the first behavioral constraint, and then incrementally adding layers of structure to satisfy the remaining constraints sequentially. This behavior-based stance removes

1. While none of these terms is ever rigorously defined, knowledge is the set of "beliefs" of the agent, a goal is a desired state (of the world, for instance), and the principle of rationality stipulates that an agent will use its knowledge to accomplish its goals.

a large bias on the part of the designer: modules arise from directly observable constraints on behavior rather than functional constraints implicit in the mind of the designer.

Unfortunately, Brooks does not go far enough. After performing a behavioral decomposition to define the functionality of a layer, he then proceeds to design a set of finite state automata (FSAs) to implement that layer. Yet, this is precisely the type of functional decomposition he warns against (Brooks, 1991, p. 146). One might appeal to learning to avoid performing this functional decomposition by hand, but current work in automating behavior-based design focuses instead on learning the interactions between preexisting behavioral modules (e.g., Maes, 1991).

We feel that the reliance upon designed modules arises from choosing FSAs as the level in which to implement subsumptive systems; in particular, from the arbitrary ways in which FSAs interact. Brooks achieves modularity through task-based decomposition of complex behavior into a set of simpler behaviors. In his system, for example, layer 0 implements obstacle avoidance and layer 1 controls wandering. Activity in layer 1 suppresses the activity of layer 0, and yet obstacles are still avoided because *layer 1 subsumes the obstacle avoidance behavior of layer 0*. In order to avoid duplication of lower layers as subparts of higher layers, he allows the higher layers to randomly access the internal components of any lower level FSAs. This fact, combined with multiple realizability of layers forces us to question Brooks' design methodology: development of single layer competence, freezing it, and then layering additional competencies 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.

Furthermore, Brooks fails to limit the suppression and/or inhibition which may occur between layers, so that a higher-level may randomly modify a lower-level's computation. This unrestricted suppression/inhibition combined with the unrestricted access problem described above permit complicated interactions among layers. In Brooks' case, careful design keeps the interactions under control, and the resulting behavioral modules perform well together. For evolving subsumptive systems, however, such design-space freedom must be limited.

In this paper, we present an alternative approach to subsumptive learning. Recognizing the multitude of formalisms with which to describe behaviors (Chandrasekaran and Josephson, 1993), we explore the merits and drawbacks of adopting a connectionist implementation for our layers.[2] As will be discussed below, our version of subsumption replaces Brooks' FSAs with feedforward networks and additional circuitry, combined so that each module in a hierarchy respects the historical prerogatives of those below it, and only asserts its own control when confident. Given this basic architecture, we demonstrate how multiple behavioral constraints can be translated into network-level constraints. Finally, we discuss the importance of the connectionist substrate for the implementation of leaky levels which produce emergent behavior in an agent.

2. Cliff (1991) makes a similar proposal from the context of computational neuroethology, but does not offer an implementation.

2 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 (Beer and Gallagher, 1992), we chose to focus on the subsumptive interaction of several behaviors; hence, Addam's actuators are a level of abstraction above leg controllers (similar to Brooks, 1986). Thus, Addam moves by simply specifying δx and δy.

Internally, Addam consists of a set of feedforward connectionist networks, connected as shown in Figure 1. The 12 input lines come 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. Initially, we implemented desired movement as a single scalar value, but this proved inadequate. It did not permit zero as a stable output as the network outputs tended to saturate with training. We then switched to a difference scheme in which the actual movement control was the difference between two outputs ($+\delta x$ and $-\delta x$). This configuration allows the system to stably learn and generate positive and negative movement, as well as no movement at all.

Addam controls its movements as follows. First, the 12 sensors are sampled and fed into layer 0, placing its suggestion for δx and δy on the output lines. Layer 1 combines these same 12 sensor readings with the sum squared output of layer 0, calculates its suggestions 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.

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; 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.

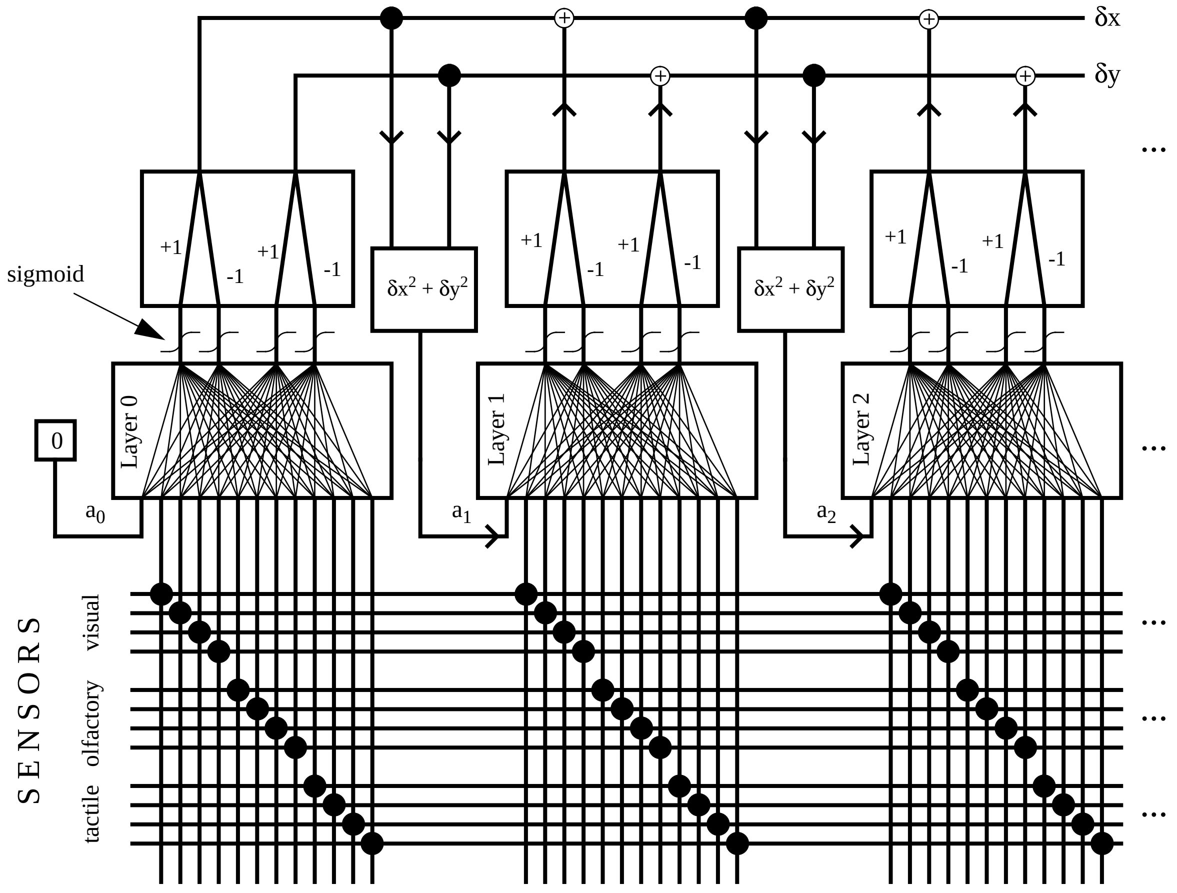

Figure 1: Addam's internal architecture. Each module possesses only limited information about the activity of its predecessors. Layer 1 receives only the sum-squared activation of layer 0, implemented as a_1. Similarly, layer 2 monitors the activity of its predecessors through a single input a_2. Through training, each layer learns to exert control only when relevant based on the current sensors, and when none of its predecessors is active.

Instead of lumping Addam with other subsumptive systems, we prefer to identify our architecture as *preemptive*. The modules are prioritized such that the behaviors associated with the lower levels may take precedence over those associated with the higher levels. Prioritization is reflected both architecturally as well as functionally. Architecturally, a lower level provides its outputs to higher levels. Functionally, 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.

Earlier we criticized Brooks' method of subsumption for two of its freedoms: unrestricted access by one layer to another's internal state, and unrestricted modulation of a lower-layer's computation by suppression/inhibition from a higher-layer. Neither problem is present in Addam. A higher layer has access only to the sum-squared output of all previous layers, and any preemption of layer *i* results from a single real value (a_i). This eliminates the methodological problem with multiple realizability: the input a_i to a layer depends only on *what* is computed below, not on *how* it is being computed.

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). 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 acknowledge the similarity of Addam's internal structure to the cascade correlation architecture of Fahlman and 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

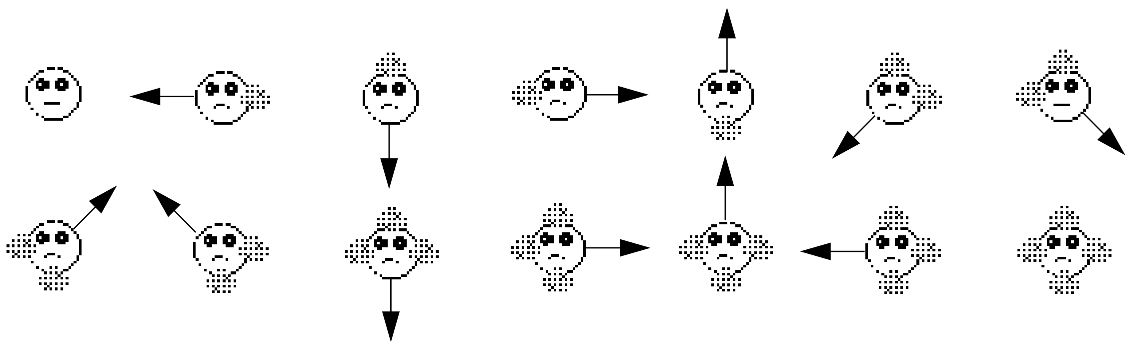

Figure 2: Training scenarios for level 0 behavior, along with desired responses. Circles denote patches of ice. The scenarios capture a range of situations; from each, Addam's target response moves it away from the ice.

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.)

3 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 and Gallagher (1992) attempted to solve this problem by using genetic algorithms, 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 emerge from the complex interactions of the adaptive modules and the environment.

Training begins with level 0 competence, defined as the ability to avoid ice. The training scenarios are shown 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 activation 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.

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.

4 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 applica-

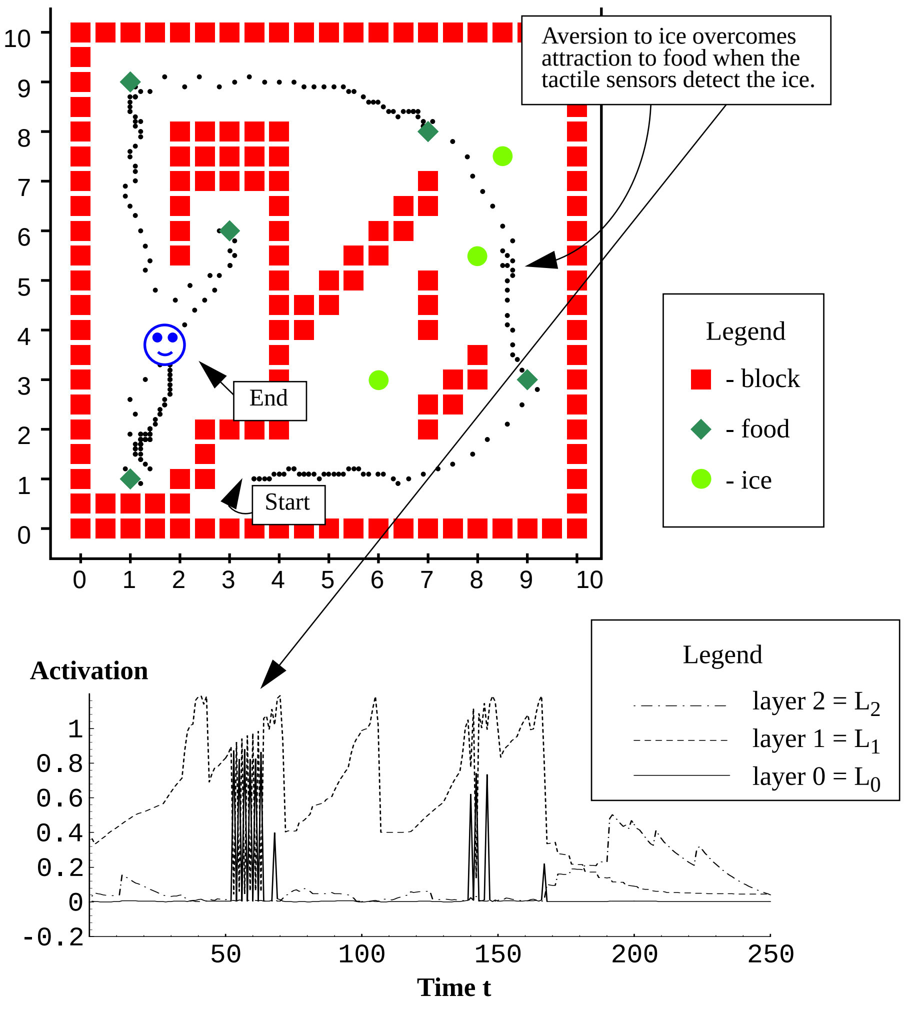

Figure 3: Addam’s emergent behavior in a complex environment. The dots in the upper figure trace Addam’s path as it moves through the environment in search of food. The graph shows the activity of each of Addam’s layers over time.

tion 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 and reaches the food, which is consumed.

Subsequently, 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 ice overcomes its 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. 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). 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 balanced 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.

5 Remarks

The behavior of Chandrasekaran and Josephson's coin sorter is best described by appealing to multiple levels of behavior (Chandrasekaran and Josephson, 1993). Addam is best described in a similar way. At one level, it is an agent which exhibits only three behaviors: avoid ice, go to food, and avoid blocks. But the underlying connectionist levels leak through in the complex interaction that allows Addam to navigate around the ice in Figure 3. Had Addam been implemented as a set of FSAs, such complex behavior would not have emerged; it would have required explicit design (Cariani, 1989). Similarly, had preemption been absolute, Addam would have become stuck at the ice as module 0 and module 1 alternately controlled the agent's behavior.[3]

This performance benefit of simplified subsumption is complemented by a benefit in training. As mentioned above, Brooksian FSAs are difficult to train because of the complicated ways in which they may interact. Our connectionist networks, on the other hand, permit a host of training algorithms. In fact, the work of Beer and Gallagher (1992) or Maes and Brooks (1990) 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 genetic or correlation algorithms.

Our work also sheds light on the issue of neural network representations for agents. Collins and Jefferson (1991) explored such representations, but found them lacking because of their inability to shift behavior based on changing inputs. Preemption offers one way in which these shifts may be obtained.

One drawback, or at least cause for concern, with our method of preemption arises from the way in which the structural modules were defined. First, as with Brooks' subsumption, we used a behavioral decomposition to define the number of modules, and second, we assumed a fixed network architecture for each module. Angeline (1994) has explored how modularization can arise without a behavioral decomposition, and elsewhere, we have explored how the structure of a module (i.e., number of hidden units and network connectivity) can arise from an evolutionary program (Saunders, Angeline, and Pollack, 1994).

Many of the problems of training behavior-based systems stem from the failure to recognize the multiplicity of levels in agents. We whole-heartedly agree with Brooks that the level of behaviors is particularly useful for the expression of design constraints. The level of FSAs may also be useful for refining the behavioral description. Yet, in the context of evolving agents, the network level is more appropriate. Our connectionist approach maintains the benefits of subsumption: a behavior-based view, incremental construction of the agent, and distributed control. But, in addition to the performance and training benefits described above, the neural network substrate offers a many-to-many mapping between structure and behavior: a single module can affect multiple behaviors, and a single behavior can arise from the interaction of multiple modules. Chandrasekaran and Josephson proposed such leaking from a philosophical point of view; here we have shown how leaking occurs naturally and aids performance in an evolved connectionist system.

3. This also illustrates how our work differs from other methods of connectionist modular control (e.g., Jacobs, Jordan, and Barto, 1990), which adopt a negative view of the interactions between modules. In fact, some work along these lines explicitly focuses on training away such interactions (Nowlan and Hinton, 1991).

Acknowledgments

This research has been partially supported by Office of Naval Research grants N00014-93-1-0059 and N00014-92-J-1195. We thank Pete Angeline for his help in an earlier version of this work.

References

Agre, P. E. (1993). The symbolic worldview: Reply to Vera and Simon. *Cognitive Science*, 17(1):61–69.

Angeline, P. J. (1994). *Evolutionary Algorithms and Emergent Intelligence*. Ph.D. thesis, The Ohio State University, Columbus, Ohio.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122.

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.

Cariani, P. (1989). *On the Design of Devices with Emergent Semantic Properties*. Ph.D. thesis, State University of New York at Binghamton.

Chandrasekaran, B. and Josephson, S. G. (1993). Architecture of intelligence: The problems and current approaches to solutions. *Current Science*, 64(6):366–380.

Cliff, D. (1991). Computational neuroethology: A provisional manifesto. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 29–39, Cambridge. MIT Press.

Collins, R. J. and Jefferson, D. R. (1991). Representations for artificial organisms. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 382–390, Cambridge. MIT Press.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation architecture. In Touretzky, D. S., editor, *Advances in Neural Information Processing Structures 2*, pages 524–532. Morgan Kaufmann.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations, pages 77–109. MIT Press, Cambridge, MA.

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.

Kolen, J. F. (In press). The observers' paradox: The apparent computational complexity of physical systems. *Journal of Experimental and Theoretical Artificial Intelligence*.

Kolen, J. F. and Pollack, J. B. (1993). The apparent computational complexity of physical systems. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 617–622, Hillsdale, NJ. Erlbaum Associates.

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

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

McClelland, J. L., Rumelhart, D. E., and The PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 2: Psychological and Biological Models. MIT Press, Cambridge, MA.

Meyer, J. A. and Guillot, A. (1991). Simulation of adaptive behavior in animats: Review and prospect. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 2–14, Cambridge. MIT Press.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

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 Systems 3*, pages 774–780. Morgan Kaufmann.

Saunders, G. M., Angeline, P. J., and Pollack, J. B. (1994). Structural and behavioral evolution of recurrent networks. In *Advances in Neural Information Processing 7*. Morgan Kaufmann.

Searle, J. (1992). *Rediscovery of the Mind*. MIT Press, Cambridge, MA.

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

Vera, A. H. and Simon, H. A. (1993). Situated action: A symbolic interpretation. *Cognitive Science*, 17(1):7–48.

Wilson, S. W. (1991). The animat path to AI. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 15–21, Cambridge. MIT Press.

The Importance of Leaky Levels for Behavior-Based AI

Gregory M. Saunders, John F. Kolen, and Jordan B. Pollack
Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210 USA

saunders@cis.ohio-state.edu
kolen-j@cis.ohio-state.edu
pollack@cis.ohio-state.edu

Abstract

From the many possible perspectives in which an agent may be viewed, behavior-based AI selects observable actions as a particularly useful level of description. Yet behavior is clearly not structure, and anyone using behavior-based constraints to construct an agent still faces many implementational roadblocks. Such obstacles are typically avoided by adopting a finite state automaton (FSA) as a base representation. As a result, potential benefits from alternative formalisms are ignored. To explore these benefits, our work adopts a multi-level view of an agent: behaviors and FSAs are but two of many levels of description. We still focus on behaviors for the expression of design constraints, but we avoid using FSAs as an implementation. Our particular agent, Addam, is comprised of a set of connectionist networks, a substrate which promotes the automatic design of subsumptive systems. Moreover, the implementational choice has important behavioral consequences – some complex behaviors emerge due to interactions among networks and need not be specified explicitly. In this way, the underlying layers *leak* into one another, each affecting the others in subtle and desirable ways.

1 Introduction

Historically, AI has viewed agents from the Knowledge Level (Newell, 1982), in which an individual is characterized by its knowledge, goals, and rationality.[1] The abstract nature of this level has been called into question from many different directions: e.g., connectionism (Hinton et al., 1986; McClelland et al., 1986), situated action (compare Vera and Simon, 1993, with Agre, 1993), the observers' paradox (Kolen and Pollack, 1993, to appear), and others (e.g., Searle, 1993). Most recently, those studying the simulation of adaptive behavior have stressed that intelligence should not be viewed simply as knowledge and goals held together with procedural glue; there is much to learn from studying intelligence through self-sufficient agents competent to exist in the world (Meyer and Guillot, 1991; Wilson, 1991).

Yet we often forget that agents can be viewed at multiple levels of description, and as Chandrasekaran and Josephson (1993) point out, there is no single level of description which captures all aspects of an agent's behavior. To borrow their example, a simple coin sorter can be described as an abstract machine which classifies coins based on their weight and diameter, but if a lever jams, then the physical nature of the device becomes particularly important. Chandrasekaran and Josephson propose that agents be described by a set of "leaky levels," where each level of description contributes to the overall story of agent behavior, but the total picture arises due to the way the various levels interact.

The lesson is an important one, but it fails to address an important question: How does the recognition of multiple levels of description help one to implement an intelligent agent? In particular, how should one approach the task of constructing an agent which satisfies multiple behavioral constraints?

Brooks (1986, 1991) proposes an interesting answer to this question. Rather than observing a set of behavioral constraints and reasoning "The agent must have functional modules for perception, planning, etc.," one can remain more faithful to the actual observations by constructing an agent which satisfies the first behavioral constraint, and then incrementally adding layers of structure to satisfy the remaining constraints sequentially. This behavior-based stance removes

1. While none of these terms is ever rigorously defined, knowledge is the set of "beliefs" of the agent, a goal is a desired state (of the world, for instance), and the principle of rationality stipulates that an agent will use its knowledge to accomplish its goals.

a large bias on the part of the designer: modules arise from directly observable constraints on behavior rather than functional constraints implicit in the mind of the designer.

Unfortunately, Brooks does not go far enough. After performing a behavioral decomposition to define the functionality of a layer, he then proceeds to design a set of finite state automata (FSAs) to implement that layer. Yet, this is precisely the type of functional decomposition he warns against (Brooks, 1991, p. 146). One might appeal to learning to avoid performing this functional decomposition by hand, but current work in automating behavior-based design focuses instead on learning the interactions between preexisting behavioral modules (e.g., Maes, 1991).

We feel that the reliance upon designed modules arises from choosing FSAs as the level in which to implement subsumptive systems; in particular, from the arbitrary ways in which FSAs interact. Brooks achieves modularity through task-based decomposition of complex behavior into a set of simpler behaviors. In his system, for example, layer 0 implements obstacle avoidance and layer 1 controls wandering. Activity in layer 1 suppresses the activity of layer 0, and yet obstacles are still avoided because *layer 1 subsumes the obstacle avoidance behavior of layer 0*. In order to avoid duplication of lower layers as subparts of higher layers, he allows the higher layers to randomly access the internal components of any lower level FSAs. This fact, combined with multiple realizability of layers forces us to question Brooks' design methodology: development of single layer competence, freezing it, and then layering additional competencies 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.

Furthermore, Brooks fails to limit the suppression and/or inhibition which may occur between layers, so that a higher-level may randomly modify a lower-level's computation. This unrestricted suppression/inhibition combined with the unrestricted access problem described above permit complicated interactions among layers. In Brooks' case, careful design keeps the interactions under control, and the resulting behavioral modules perform well together. For evolving subsumptive systems, however, such design-space freedom must be limited.

In this paper, we present an alternative approach to subsumptive learning. Recognizing the multitude of formalisms with which to describe behaviors (Chandrasekaran and Josephson, 1993), we explore the merits and drawbacks of adopting a connectionist implementation for our layers.[2] As will be discussed below, our version of subsumption replaces Brooks' FSAs with feedforward networks and additional circuitry, combined so that each module in a hierarchy respects the historical prerogatives of those below it, and only asserts its own control when confident. Given this basic architecture, we demonstrate how multiple behavioral constraints can be translated into network-level constraints. Finally, we discuss the importance of the connectionist substrate for the implementation of leaky levels which produce emergent behavior in an agent.

2. Cliff (1991) makes a similar proposal from the context of computational neuroethology, but does not offer an implementation.

2 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 (Beer and Gallagher, 1992), we chose to focus on the subsumptive interaction of several behaviors; hence, Addam's actuators are a level of abstraction above leg controllers (similar to Brooks, 1986). Thus, Addam moves by simply specifying δx and δy.

Internally, Addam consists of a set of feedforward connectionist networks, connected as shown in Figure 1. The 12 input lines come 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. Initially, we implemented desired movement as a single scalar value, but this proved inadequate. It did not permit zero as a stable output as the network outputs tended to saturate with training. We then switched to a difference scheme in which the actual movement control was the difference between two outputs ($+\delta x$ and $-\delta x$). This configuration allows the system to stably learn and generate positive and negative movement, as well as no movement at all.

Addam controls its movements as follows. First, the 12 sensors are sampled and fed into layer 0, placing its suggestion for δx and δy on the output lines. Layer 1 combines these same 12 sensor readings with the sum squared output of layer 0, calculates its suggestions 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.

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; 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.

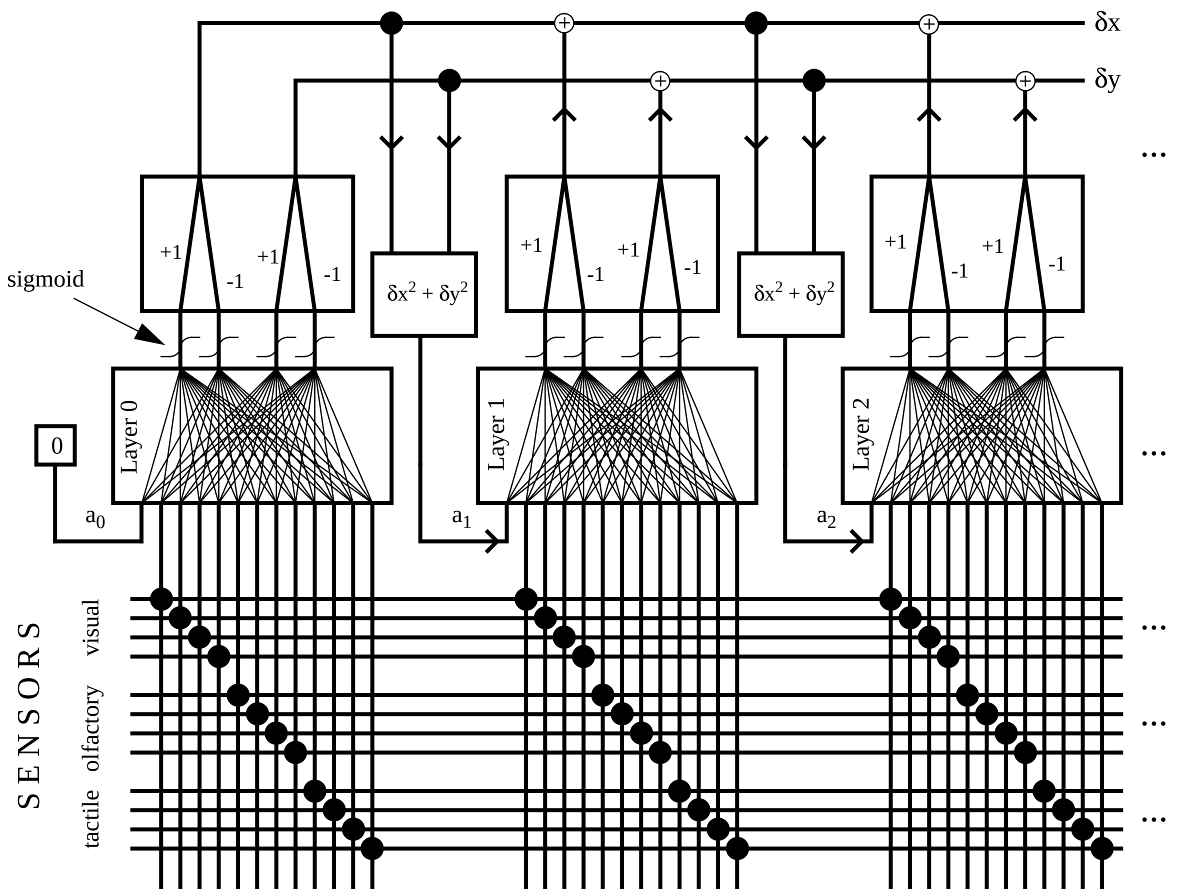

Figure 1: Addam's internal architecture. Each module possesses only limited information about the activity of its predecessors. Layer 1 receives only the sum-squared activation of layer 0, implemented as a_1. Similarly, layer 2 monitors the activity of its predecessors through a single input a_2. Through training, each layer learns to exert control only when relevant based on the current sensors, and when none of its predecessors is active.

Instead of lumping Addam with other subsumptive systems, we prefer to identify our architecture as *preemptive*. The modules are prioritized such that the behaviors associated with the lower levels may take precedence over those associated with the higher levels. Prioritization is reflected both architecturally as well as functionally. Architecturally, a lower level provides its outputs to higher levels. Functionally, 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.

Earlier we criticized Brooks' method of subsumption for two of its freedoms: unrestricted access by one layer to another's internal state, and unrestricted modulation of a lower-layer's computation by suppression/inhibition from a higher-layer. Neither problem is present in Addam. A higher layer has access only to the sum-squared output of all previous layers, and any preemption of layer i results from a single real value (a_i). This eliminates the methodological problem with multiple realizability: the input a_i to a layer depends only on *what* is computed below, not on *how* it is being computed.

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). 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 acknowledge the similarity of Addam's internal structure to the cascade correlation architecture of Fahlman and 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

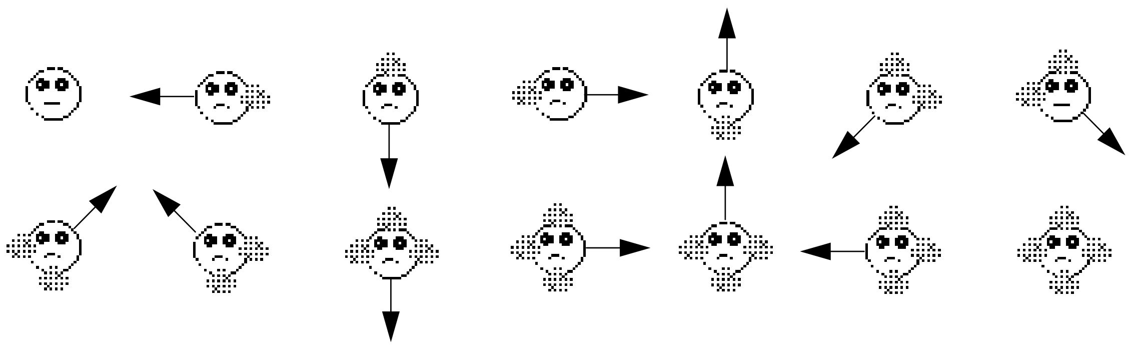

Figure 2: Training scenarios for level 0 behavior, along with desired responses. Circles denote patches of ice. The scenarios capture a range of situations; from each, Addam's target response moves it away from the ice.

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.)

3 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 and Gallagher (1992) attempted to solve this problem by using genetic algorithms, 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 emerge from the complex interactions of the adaptive modules and the environment.

Training begins with level 0 competence, defined as the ability to avoid ice. The training scenarios are shown 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 activation 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.

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.

4 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 applica-

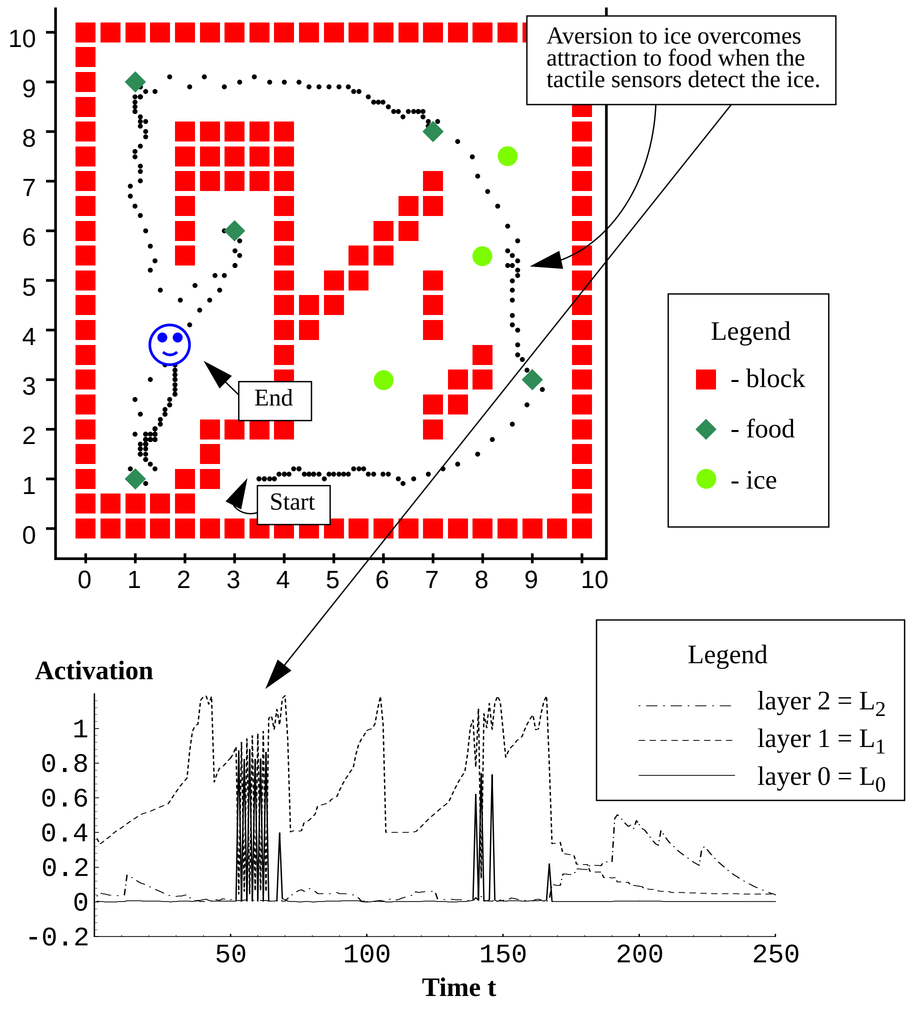

Figure 3: Addam’s emergent behavior in a complex environment. The dots in the upper figure trace Addam’s path as it moves through the environment in search of food. The graph shows the activity of each of Addam’s layers over time.

tion 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 and reaches the food, which is consumed.

Subsequently, 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 ice overcomes its 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. 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). 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 balanced 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.

5 Remarks

The behavior of Chandrasekaran and Josephson's coin sorter is best described by appealing to multiple levels of behavior (Chandrasekaran and Josephson, 1993). Addam is best described in a similar way. At one level, it is an agent which exhibits only three behaviors: avoid ice, go to food, and avoid blocks. But the underlying connectionist levels leak through in the complex interaction that allows Addam to navigate around the ice in Figure 3. Had Addam been implemented as a set of FSAs, such complex behavior would not have emerged; it would have required explicit design (Cariani, 1989). Similarly, had preemption been absolute, Addam would have become stuck at the ice as module 0 and module 1 alternately controlled the agent's behavior.[3]

This performance benefit of simplified subsumption is complemented by a benefit in training. As mentioned above, Brooksian FSAs are difficult to train because of the complicated ways in which they may interact. Our connectionist networks, on the other hand, permit a host of training algorithms. In fact, the work of Beer and Gallagher (1992) or Maes and Brooks (1990) 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 genetic or correlation algorithms.

Our work also sheds light on the issue of neural network representations for agents. Collins and Jefferson (1991) explored such representations, but found them lacking because of their inability to shift behavior based on changing inputs. Preemption offers one way in which these shifts may be obtained.

One drawback, or at least cause for concern, with our method of preemption arises from the way in which the structural modules were defined. First, as with Brooks' subsumption, we used a behavioral decomposition to define the number of modules, and second, we assumed a fixed network architecture for each module. Angeline (1994) has explored how modularization can arise without a behavioral decomposition, and elsewhere, we have explored how the structure of a module (i.e., number of hidden units and network connectivity) can arise from an evolutionary program (Saunders, Angeline, and Pollack, 1994).

Many of the problems of training behavior-based systems stem from the failure to recognize the multiplicity of levels in agents. We whole-heartedly agree with Brooks that the level of behaviors is particularly useful for the expression of design constraints. The level of FSAs may also be useful for refining the behavioral description. Yet, in the context of evolving agents, the network level is more appropriate. Our connectionist approach maintains the benefits of subsumption: a behavior-based view, incremental construction of the agent, and distributed control. But, in addition to the performance and training benefits described above, the neural network substrate offers a many-to-many mapping between structure and behavior: a single module can affect multiple behaviors, and a single behavior can arise from the interaction of multiple modules. Chandrasekaran and Josephson proposed such leaking from a philosophical point of view; here we have shown how leaking occurs naturally and aids performance in an evolved connectionist system.

3. This also illustrates how our work differs from other methods of connectionist modular control (e.g., Jacobs, Jordan, and Barto, 1990), which adopt a negative view of the interactions between modules. In fact, some work along these lines explicitly focuses on training away such interactions (Nowlan and Hinton, 1991).

Acknowledgments

This research has been partially supported by Office of Naval Research grants N00014-93-1-0059 and N00014-92-J-1195. We thank Pete Angeline for his help in an earlier version of this work.

References

Agre, P. E. (1993). The symbolic worldview: Reply to Vera and Simon. *Cognitive Science*, 17(1):61–69.

Angeline, P. J. (1994). *Evolutionary Algorithms and Emergent Intelligence*. Ph.D. thesis, The Ohio State University, Columbus, Ohio.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122.

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.

Cariani, P. (1989). *On the Design of Devices with Emergent Semantic Properties*. Ph.D. thesis, State University of New York at Binghamton.

Chandrasekaran, B. and Josephson, S. G. (1993). Architecture of intelligence: The problems and current approaches to solutions. *Current Science*, 64(6):366–380.

Cliff, D. (1991). Computational neuroethology: A provisional manifesto. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 29–39, Cambridge. MIT Press.

Collins, R. J. and Jefferson, D. R. (1991). Representations for artificial organisms. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 382–390, Cambridge. MIT Press.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation architecture. In Touretzky, D. S., editor, *Advances in Neural Information Processing Structures 2*, pages 524–532. Morgan Kaufmann.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations, pages 77–109. MIT Press, Cambridge, MA.

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.

Kolen, J. F. (In press). The observers' paradox: The apparent computational complexity of physical systems. *Journal of Experimental and Theoretical Artificial Intelligence*.

Kolen, J. F. and Pollack, J. B. (1993). The apparent computational complexity of physical systems. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 617–622, Hillsdale, NJ. Erlbaum Associates.

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

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

McClelland, J. L., Rumelhart, D. E., and The PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 2: Psychological and Biological Models. MIT Press, Cambridge, MA.

Meyer, J. A. and Guillot, A. (1991). Simulation of adaptive behavior in animats: Review and prospect. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 2–14, Cambridge. MIT Press.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

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 Systems 3*, pages 774–780. Morgan Kaufmann.

Saunders, G. M., Angeline, P. J., and Pollack, J. B. (1994). Structural and behavioral evolution of recurrent networks. In *Advances in Neural Information Processing 7*. Morgan Kaufmann.

Searle, J. (1992). *Rediscovery of the Mind*. MIT Press, Cambridge, MA.

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

Vera, A. H. and Simon, H. A. (1993). Situated action: A symbolic interpretation. *Cognitive Science*, 17(1):7–48.

Wilson, S. W. (1991). The animat path to AI. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 15–21, Cambridge. MIT Press.