Giving Nolfi a Fair Fight
A Comparison of Emergent Modularity and Elman Recurrent Networks

Sven Olsen and Matt Fowles

May 14, 2003

Abstract

During the last few years, using neural networks as robot controllers has received heavy attention in robotics research. Searching for a network based controller capable of accomplishing a particular tasks has proven time consuming. Genetic algorithms are often applied to find these neural networks. In this paper we explore how well genetic algorithms can find controllers for a trash removal task given various network architectures. We repeat and expand on the work done by Nolfi in *Using emergent modularity to develop control systems for mobile robots*, comparing his emergent modularity network with both two-layer feed forward and Elman architectures. While we encountered a number of technical difficulties, we were able to verify some of Nolfi's findings: the emergent modularity architecture quickly converges to a solution, and the two-layer architecture spends the first 75 generations stuck using an ineffective strategy. Interestingly the Elman network, which was absent from Nolfi's original experiment, proves far more effective than Nolfi's modular architecture in the long run.

1 Introduction

When applied to robotics, traditional AI demands that we carefully engineer the control system of a robot, first using symbolic logic to abstract the state of the world, and then applying more logic to determine the robot's actions given that internal representation of the world. Neural networks provide an alternative to traditional AI, allowing us to tie a robot's behavior more directly to its sensor values. All we need to do is define simple translations from the robot's sensors and actuators to the input and output nodes of the network, and, given those translations, the nature of the network fully determines the robot's actions. Although the connection thus provided is more direct than those in a symbolic system, hidden layers and recurrences in the network allow abstractions to be built into the system. Unlike symbolic abstractions, these abstractions arise naturally from patterns in the sensor data and the network architecture, rather than being created by engineers who perceive the robot's world very differently than the robot does with its sensors. For this reason neural networks have the potential to produce simpler control systems that make effective use of the data available to the robot.

If we have some way of knowing what a robot's behavior should be given any set of sensor values, then it is relatively easy to use backpropagation of error to find a set of network weights that can approximate that behavior well. The problem with such a method is that it relies on knowledge already inherent in the teacher function. If we, as robot designers, already have access to a teacher capable of performing the desired behavior, then our task is probably a simple enough one or well enough understood that we could create a controller by hand. When attempting to create complex behaviors, we almost never have an easy way of finding a teacher function. Thus we turn to genetic algorithms, which provide us with a more widely applicable, albeit less efficient, method of finding network weights.

Formally, for any function $f : \mathbb{R}^n \to \mathbb{R}^m$ there exists a neural network with a hidden layer that can approximate f to an arbitrary precision [2]. While in practice, we fix the number of nodes in the hidden layer of a network, a two-layer neural network still has the potential to accurately approximate a fairly large range of functions. These functions are naturally parameterized by the weights and biases of the network. Given a robot with m sensors, n actuators, and a method for defining the distance between the behaviors associated with a particular weight set and our goal behavior, we can bring the accumulated knowledge of search algorithms to bear on this problem using a neural network as the control function.[1] The weight space implied by a neural network is typically large, and so we use genetic algorithms as our search method because they are designed to effectively search such high dimensional spaces.

While neural networks have tantalizing potential, they have yet to be used to create complex robotic controllers. Nolfi [6] attempts to pave the way for more complex neural network based controllers by introducing network architectures designed to encourage the network to break complex tasks into simpler subtasks. He applies these architectures to the task of removing trash from a small rectangular world, using genetic algorithm with a fitness function defined by the robot's success as a trash remover. Nolfi compares two of his "modular" architectures to a number of more conventional networks. He finds that the network that defines its own way of breaking up the task is by far the most effective. Nolfi dubs this highly successful architecture an "emergent modularity" (EM) network. The other modular network used in Nolfi's experiment had one set of outputs that was used while the robot was attempting to find the garbage, and one set that was used after it had picked the garbage up. Nolfi explained the poor performance of this network by claiming that the division of the task assigned by a human engineer was not particularly useful given the limitations of the robot's sensors. However, the EM network, free from many of the assumptions of human designers, was better able to develop ways of overcoming its

[1] Recurrent networks cannot be properly considered functions, as they possess time sensitivity.

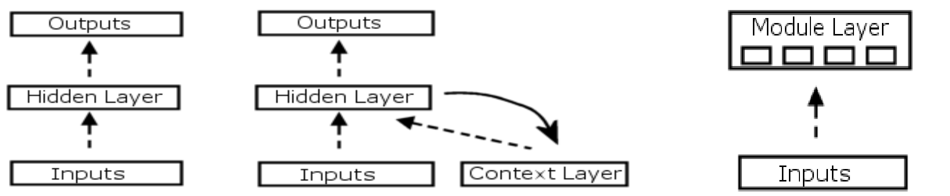

Figure 1: Network Architectures

limitations.

While Nolfi's results are exciting, it can be argued that the EM architecture which he champions has an unfair advantage over the other networks in his experiment. Nolfi's emergent modular system has well over twice the number of weights of any competitor. While it is true that in some cases networks will benefit from having a smaller number of weights, these cases usually involve interesting topologies with at least one hidden layer. The concept of emergent modularity presented by Nolfi is certainly exciting, but it seems that he has limited the competing networks in his experiment to a collection of straw men.

In this paper, we attempt to recreate Nolfi's experiment, while supplying new competitors with whom Nolfi's emergent modularity system can compete on even footing. We chose our networks such that they all have a roughly equal number of weights, as this provides an objective measure of a networks power. To what extent the actual computational potential of these networks is equivalent is left to more competent debaters. We chose two canonical networks to compete against Nolfi's architecture, both of which have proven successful in solving reasonably complex tasks, an Elman network and a two-layer feed forward network (see figure 1). While Nolfi does include a recurrent Network among his set of test architectures, his has only two recurrent nodes, and no hidden layer. Thus, an Elman network allows for considerably more complex behaviors than Nolfi's recurrent net.

2 Related Work

In the paper *Using emergent modularity to develop control systems for mobile robots* [6], on which we base this experiment, Nolfi compares the performance of 5 different network architectures when applied to a trash removal task. A one layer network with only input and output nodes; a one layer network with two additional output nodes that recur as input nodes; a two-layer network with 4 hidden nodes; a one layer network with two predefined modules; and his EM network. These networks have 32, 60, 52, 64, and 128 total weights and biases respectively. All of the architectures except the EM perform relatively poorly. While the feed forward network ends with results almost as good as the emergent modularity's, its behavior does not translate to the real world as well as that of the modular architecture. While Nolfi observes this phenomenon he does not offer any explanation of it. Nolfi argues, after analyzing his results, that the difficult problem for the robots in this task is to identify whether an object in the sensors is a wall or trash. The modular architecture determines this by moving to the object and "dancing" back and forth in front of the object until it has a clear sensor reading on which to base its decision.

When using genetic algorithms to find neural network weights for robot controls, we can only hope to achieve success if we use an architecture such that there exists a set of weights which preform the behavior we want. However, for the algorithm to be successful, we also need the weight space implied by the architecture to be "GA friendly", so that we have a reasonable chance of finding a good weight set. For example, a network architecture may allow a large range of values to preform the same action, making it easier to find, or it may have very strict limits on the values, making the behavior hard to stumble upon. There may well exist weight sets for the simpler networks used in Nolfi's experiment that would allow them to perform as well or better than his champion EM design. What Nolfi's results show is that the the EM architecture implies a weight space in which a GA has a much better chance of finding an effective weight set.

One of the experiments discussed by Floreano in [7] demonstrates how learning can be combined with a genetic algorithm in order to improve the chances that a GA will be able to find a good weight set. Training some of the networks nodes on a prediction task during the life of the robot and running a genetic algorithm on the robot population using a fitness score based on a separate task forces the GA to move to sections of the search space in which better prediction performance coincides with higher fitness scores. In Floreano's terminology, it forces the GA into areas where the learning space and the fitness space are highly correlated. Because of this correlation, as learning improves the network's predictive ability, the learning also improves the network's performance on the fitness task. Thus, instead of being "hit or miss", potential sets of network weights tend to achieve fitness scores which reflect their distance from high fitness weight sets. This makes it easier for the GA to navigate the weight space.

The Elman architecture that we use in our experiments was originally presented in [3]. Elman designed the network with the goal that it be able to effectively detect patterns in time. He describes a series of test which demonstrates the network's success at a number of time dependent tasks. The first test of the network that he presents demonstrates its ability to recognize patterns of XOR in time. His later experiments build up to much more complex tasks, in which the network is used to identify the syntactic and semantic characteristics of words.

While the topic has little relation to neural networks or robotics, Kaplan and Salesin's paper on Escherization [4] includes a description of an interesting search algorithm designed to handle high dimensional spaces. Kaplan runs a number of simulated annealers in parallel, and prunes the population of parallel annealers every n time steps to eliminating the lowest scoring searches.

The behavior of an annealer can be roughly matched by running a GA and creating the next generation by picking and mutating one of the most fit individual in the population (the probability of not picking the most fits decays over time). Thus the search method applied by Kaplan is similar to running a number of separate GA's in parallel, each having a small population, and pruning the population of the GA's every n time steps. The fact that Kaplan found this search method successful suggests that it could be an interesting alternative to more conventional large population GA's.

3 Methods/Mistakes

We began by searching for a simulator in which we could reproduce Nolfi's work. We settled upon Player/Stage[2] because it provided a C++ interface, and allows us to simulate pucks and a gripper module. The pucks function as trash since they can be picked up by the grippers. Initially, we implemented Nolfi's trash world as a player client that used truth objects tied to the pucks to reset the world between runs. Unfortunately, we uncovered a few bugs, not all of which we could track down. Since Player/Stage is conveniently open source, we were able to hack the stage sources directly layering our controllers directly on top of the ordinary update loop. This had the ancillary benefit of giving nearly an order of magnitude increase in speed. We used Annie[3] for our neural networks, which we also customized, adding an Elman network class and code to allow us to set set network weights. We wrote our own Genetic Algorithm library which, along with all of our modified sources, can be downloaded from Sven's website.[4]

In order to maximize our chances of successfully replicating Nolfi's results, we modelled our simulated world closely on his. We used a simulated 5.5 cm diameter circular robot equipped with a gripper as our stand in for Nolfi's Kheperas. The pucks were 2.3 cm in diameter (the same size as Nolfi's cylinders). Our world was a square box approximately 45 cm x 45 cm; Nolfi used a rectangular box, 60 cm x 35 cm. Like Nolfi, we initialized our world to contain 5 randomly placed piece of trash. Trash placement was restricted such that pieces of trash did not appear overlapping the world's walls, the robot, or each other. We assumed that because our world and Nolfi's contained approximately the same amount of trash per unit area, the minor difference in shape would not significantly effect robot performance.

Nolfi's Khepera received input from six forward facing IR sensors. The Stage simulator does not provide IR sensors. Therefore, we did our best to use a scanning laser to approximate IR values. We restricted the range of the sensors to 2.5 cm beyond the body of the robot. We then converted the range information from the laser according to our approxi-

[2]http://playerstage.sourceforge.net
[3]http://annie.sourceforge.net
[4]http://www.sccs.swarthmore.edu/users/03/sven/downloads/NolfiRedux.tar.gz

mation of the expected IR sensor activations given white paper objects [1], which seemed appropriate given that Nolfi covered all the objects in his experiment with white paper. The stage simulator does not provide robots with two parallel motors, but instead has settings for linear and angular velocity. To translate motor values to state, we first scale our outputs into the range of -100 to 100. Then we assign a linear velocity based on the difference of the two values with a bit of logic to handle signs intelligently. We then set the angular velocity to $\frac{rightMotor - leftMotor}{3.4}$. The 3.4 in the previous equation was discover via a manual hill climb and subjective fitness function.

3.1 Network Architecture

All networks in our experiments take seven inputs, six sensor values and a seventh boolean input which is activated when the gripper's break beam is cut. The Elman and two-layer networks each have four outputs. The first two specify the left and right motor values; the last two inputs are used to open and close the gripper. We open or close the gripper when only the corresponding node has an activation greater than .85. If both gripper control nodes are greater than .85 we do nothing. The EM network is faithfully copied from Nolfi's description, a single layer feed forward network with 16 outputs. The 16 outputs are divided into 4 sets, each set corresponding to one of the standard output commands. Two of the nodes specify potential outputs for the set, and the other two nodes are "selector" neurons which compete to determine which of the potential outputs is used.

Our two-layer network has a hidden layer of 10 nodes. The size of the hidden layer implies a network with 124 total weights and biases, slightly fewer than Nolfi's EM network which has 128.

Elman networks are two-layer feed forward networks, with all the nodes in the hidden layer made recurrent. The Elman network has seven hidden nodes, which implies 116 total weights and biases. We could have used a hidden layer of 8 nodes, which would have implied 132 total weights and biases (closer to Nolfi's 128), but we decided that our point against Nolfi would be made more strongly if we found a network with fewer weights and better performance.

3.2 Fitness Function

Initially we copied Nolfi's fitness function, which ran 15 trials but halted each after the robot successfully drops a piece of trash outside the world. If a robot removed a piece of trash during a trial it would receive 1 fitness point, but if it only managed to pick up a piece of trash it would get .1 fitness points instead. After starting several runs with this function, we decided that the time required for 15 tests was too great. With this in mind, we lowered the number of trials per run to seven. We soon noticed that now networks were achieving nearly maximum fitness scores at very early generations. When we watched the high scoring behavior for ourselves, however, we saw that all the robot was doing was driving forward, grabbing a puck if it happened to run into one, and then opening its gripper as soon as it hit the wall. While most of the time this was an ineffective behavior, in a population of 75 individuals all using this strategy, one or two will be lucky enough to have pucks placed directly in front of them nearly all the time. Because a more inventive approach would not be able to outscore these one or two lucky individuals, once the population is filled with robots which used this strategy, new behaviors are unlikely to emerge.

Having learned our lesson, we returned the number of trials to 15, acting out of the belief that while one or two individuals might get lucky 6 or 7 out of 7 times, they would be much less likely to get lucky in 14 or 15 out of 15 trials. When we watched the preliminary results from the next round of experiments, we noticed that some of our networks were developing relatively simple behaviors, which failed in the cases when they encountered a puck while holding another. In his own experiment, Nolfi had his simulator move one of the other pieces of trash directly in front of a robot immediately after the robot picked up a piece of trash. We decided that doing this might provoke more sophisticated behaviors, and so implemented it in our own experiment. We then went a step beyond Nolfi in our quest to encourage more sophisticated robot behaviors, and added what we called a "full trial" option. If a robot manages to score over 10 points in its partial trials, we grant it the right to compete in a full trial. In a full trial, the simulation does not terminate after the robot successfully removes a piece of trash. Instead, it has a total of 800 actions, 4 times the length of a partial trial, in which to clear the world, gaining an additional fitness point for each piece cleared. In a full trial, a puck is not teleported in front of the robot when it first picks up another puck.

We did all of this preliminary work using the two-layer network as our guinea pig. In retrospect, this was probably a bad idea as the two-layer network takes a long time to display interesting behaviors. We should have used the EM network to test our runs as we know what sort of behavior it should manifest over time. Also, the EM network tends to produce interesting results slightly more quickly then other networks.

3.3 Genetic Algorithm

Nolfi's genetic algorithm used a population of 100 individuals. After evaluating the fitness of each individual, the top 20 scorers created 5 children each, which formed the next generation's population. Nolfi's experiment represented each weight as an 8 bit number. He randomly mutated the bits strings representing a neural net's weights with a 2% chance of replacing any bit with a new random bit. Since he mutated each bit with a 2% chance, only 1% of the bits are actually changed on average. He did not use crossover in his GA.

Our GA used vectors of doubles to represent the weights of a network. Although we experimented with other population sizes and reproduction rates, we ended up using a population size of 100, and creating 5 children from each of the top 20 individuals in a given generation, just as Nolfi did. However, in order to more quickly search the space, we included a special case in our children generating method. If a parent had 0 fitness, each of its children had a 50% chance of being an entirely new gene. We hoped that this would have the effect of introducing "new blood" into a population that had been filled up with individuals that all performed the same simple strategy, relying on luck to propagate. We mutated each weight in a gene with a 1% probability.[5] We used a normal gaussian, centered at 0 and scaled by 5 to mutate each double. We chose a gaussian distribution because a bit selected at random from a number is more likely to be a low order bit (as there are more of them), thus when bits are mutated randomly, the effect is more likely to be a small change in the number.

4 Results

Unfortunately, there seems to be a lurking bug in our experiment implementation. After running for a considerable amount of time (usually more than 14 hours / 70 generations), all of our fitness scores abruptly drop to 0. We hypothesize that some quirk in the simulation is causing the re-initialization of the world to fail in such a way that it becomes impossible for the robots to interact with the pucks. However, as we have no way of viewing the world mid-run, the problem is incredibly difficult to track down. Were we professional researchers, we would leave several versions of the experiment running in graphical mode, and with a little luck, after about 4 days (each trial take notably longer when running in graphical mode), we would be able to start isolating the bug. However, as we are college students with limited time, we have instead decided that the 75 generations of good data that we do have is enough on which to base a paper.

Our Elman experiment mysteriously crashed after running for about 11 generations. We wrote a quick program to resume the experiment from our log files. However, analysis of the resumed experiment shows some odd trends. Rather than continuing from the high fitness of the earlier data, the population starts entirely filled with individuals of nearly 0 fitness. After about 70 generations, the Elman population is performing very well, but the learning curve seems much slower than that of the initial Elman run. We imagine that the initial data is not the resumed values but in fact data that is worse than random values as a starting point for a GA. We suspect that our "new blood" addition to the GA gradually phased out these undesirables, leading to the eventual high performance. But given the admirably fast convergence rate demonstrated before the crash, we suspect that Elman networks can converge much faster than our results suggest.[6]

A general trend that we noticed in all of the network populations is that a single behavior

[5] In retrospect, our GA would have been closer to Nolfi's if we had used a 8% mutation rate.

[6] Owing to this bug, we treats the beginning of the "resumption" as generation 0 in the results section.

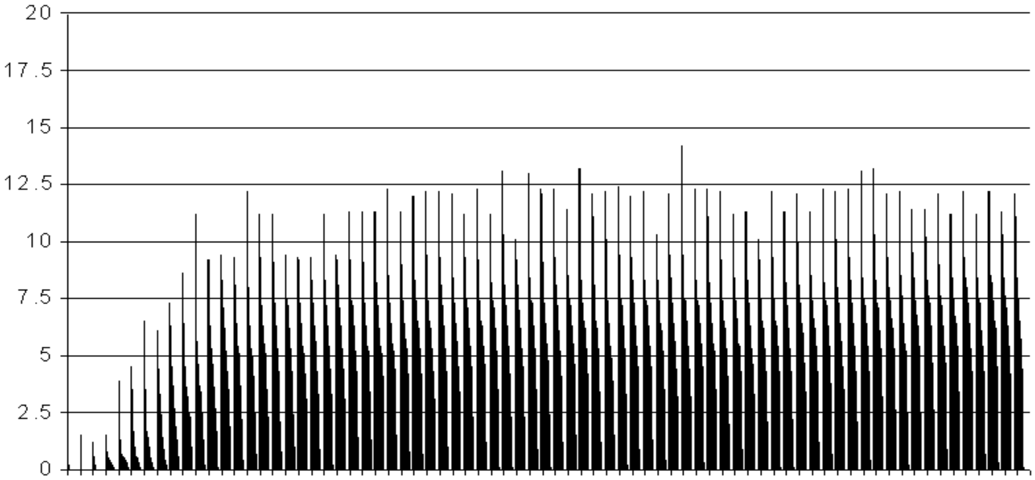

Figure 2: Emergent Modularity Network Fitness

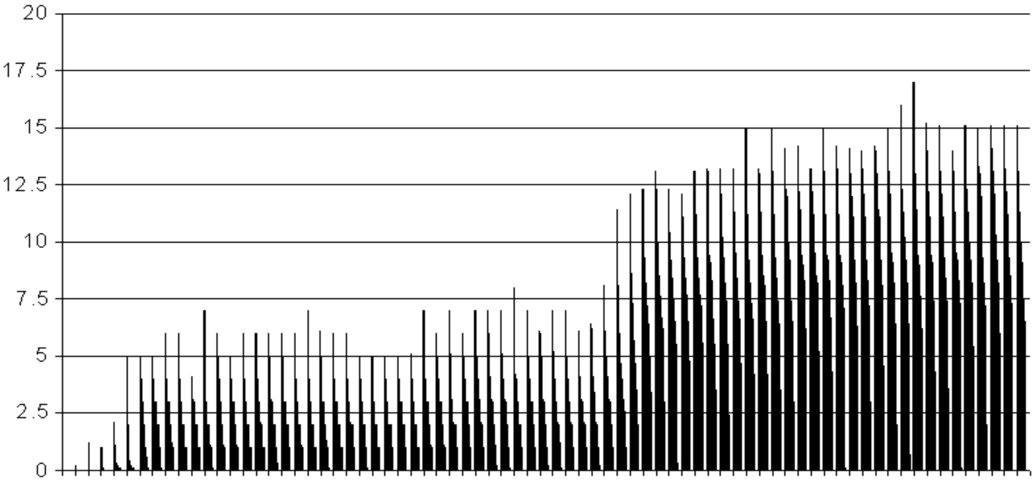

Figure 3: Elman Network Fitness

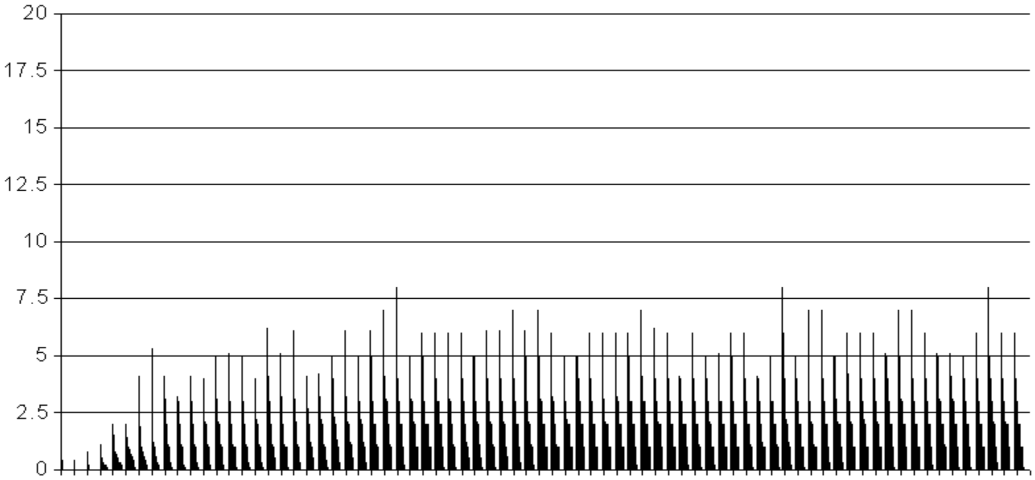

Figure 4: Two-Layer Network Fitness

clearly comes to dominate the population relatively early on. As that behavior is refined, the overall fitness of the population increases. We have yet to see two distinct behaviors coexisting within a single population.

4.1 Two-Layer Network Results

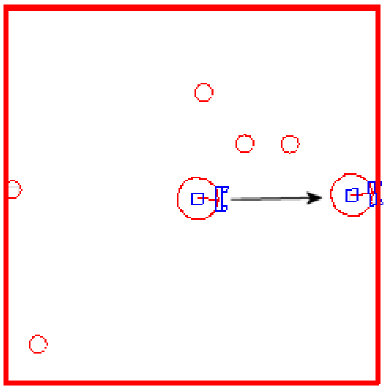

Figure 5: Two-Layer Behavior

When randomly initialized, a two-layer network tends to display one of three basic behaviors: spinning madly in place, driving forward, or driving backward. Over time these behaviors tend to mature into driving forward (sometimes with a slight arc) and grasping at anything that breaks the gripper beam (see figure 5). As discussed in our methods/mistakes section, this simple strategy can provide fitness scores anywhere between 0 and 10 depending on luck, though the maximum in a given population is usually somewhere in the range of 5 to 7. This behavior seems to dominate the population for the length of time that we were able to run the GA.

4.2 Elman Network Results

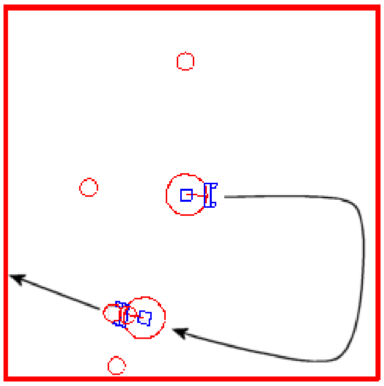

Figure 6: Elman Behavior

Randomly initialized Elman networks display much of the same behavior as two-layer networks; though on occasion a randomly initialized Elman net will switch between two simple behavior mid run. However, over time these mature into an effective and comparatively quite complex strategy. The Elman network will perform a simple wall follow, driving around the edge of the world. It will snap up any puck that it encounters, quickly drive itself into the wall, deposit the puck, and resume its circuit (see figure 6). The wall following strategy is well suited for the standard trials, where the time limit implies that the robot cannot expect to search the entire world for pucks. By limiting the area that it searches to the borders of the world, the robot maximizes its chance of finding a puck and still having enough time left to drop it over the edge of the world. On occasion the Elman driven robot will get stuck after dropping off it's puck, but more often it continues it's search. Thus if it makes it to the "full trial", the robot has a chance of scoring on all pucks positioned near the edges of the world. We have observed lucky Elman nets quickly clearing 4 of the 5 pucks in a world.

The evolved Elman net also adopts the strategy of "herding" pucks, capitalizing on a subtle flaw in the stage simulator. While a real Khepera would require a notable amount of time to pick up or release an object, the simulated robot can grab and release pucks at no time cost. The robot will herd pucks in front of it by quickly opening and closing the gripper. To imagine this behavior effectively one must realize that stage allows a newly dropped puck to overlap an already existing puck, but will only pick up the closest puck when the gripper closes. A dropped puck will appear slightly farther away from the robot than one being pushed along in front of it. Under normal circumstances, pucks pushed in front of a robot will tend to slide off to the side. However, when a puck starting to slide out of line is picked up and dropped, it is recentered with respect to the robot's line of motion. We have seen robots carry as many as 3 pucks along with them by using this strategy. During partial runs, the robot deals with the puck teleported in front of it by simply adding the new puck to the herd that it is currently carrying. During full trial runs, the robot will sometimes herd multiple pucks towards a wall before dropping them over it. The evolved Elman behavior is capable of garnering fitness scores between 11 and 17 with startling regularity.

4.3 Emergent Modularity Network Results

Randomly initialized EM networks tend to show a greater range of behavior than either of the other network types. Like the others types, these networks will occasionally just spin or drive straight; however, they are also likely to display complex behaviors like rudimentary wall following or obstacle avoidance. The evolved EM networks displayed an initial wall follow similar to that shown by the Elman networks. Unlike the Elman network, the EM network will begin to spin quickly in place after picking up a puck (see figure 7). The robot lunges for a wall if it sees one during the spin. Because it has been following a wall prior to picking up the puck, this lunge will almost always bring it to a wall. This behavior sometimes mistakes other pucks for walls; however, this is extremely uncommon as the pucks are rarely close enough to each other. Because the teleported puck appears just outside of sensor range, the spinning robot never notices it and cannot be confused by the new puck. If the network is unlucky enough to find a puck before it finds a wall, it will just spin until time runs out. In full trials the network will continue spinning after releasing the first puck by the wall.

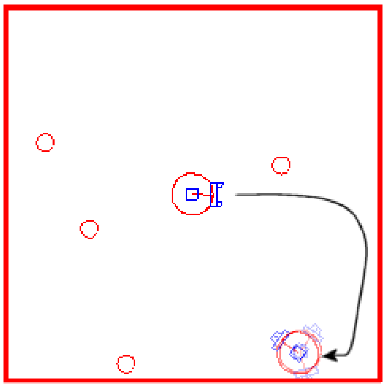

Figure 7: EM Behavior

The activation in the selector neurons, which define what module we use for a particular output, is interesting. One module continually controls the left wheel and the drop command. The right wheel switches between its two modules (one of them moving it forward, and the other causing it to spin or turn), depending on sensor values. The pick up module switches the moment anything is in its sensor range or it has its gripper closed. The switching of the right wheel accounts for both the wall follow and the spin states described above.

5 Conclusions

Despite the difference in hidden layer size, our two-layer network produces results that are similar to those that Nolfi reported for his own two-layer network. Up through generation 75, our own hidden layer fails to produce any interesting behavior. Nolfi found that his own hidden layer networks were very slow to show interesting behaviors, only evolving successful strategies around generation 400.

To some extent our results confirm Nolfi's. The EM architecture quickly finds a successfully strategy. Nolfi's EM architecture can be seen as a special case of a multi layer network. We could design a multi-layer network in which the first 16 hidden nodes produce the same outputs as Nolfi's network, and then use another hidden layer to implement the selection logic. Thus, the EM networks have a weight space that is a proper subspace of that of an appropriately sized multi-layer network. The high degree of complexity shown by randomly weighted EM networks, as well as the network's success in later generations, indicates that this subspace is a friendly one and that it can helpfully restrict our search. However, given enough time, a GA run to find multi-layer network weights should be able to find behaviors which are at least as good as the best behaviors produced using Nolfi's architecture, which is also true for two-layer networks. This is not true in the case of Elman networks.

In addition to Elman's superior performance in our experiment, Elman networks have a greater potential range of behavior than any non-recurrent network. Elman networks are sensitive to the past experiences of the robot. For this reason the outputs of Elman networks cannot be described as stateless functions. Given a situation in which the robot needs to infer whether it is looking at a puck or a wall, the only way that a non-time sensitive network such as Nolfi's can hope to distinguish between the two cases is to vary its position until the sensors values become unambiguous. However, an Elman network has the option of varying its perspective and then making a conclusion on the nature of the object based on its accumulated perceptions. Because of its recurrent character, an Elman network is able to collect information to inform its future decisions in ways that are simply inaccessible to networks without recurrences.

Just as Nolfi did, we see the modules associated with a part of the network being used in many different circumstances. Nolfi observed that only one module in his architecture was actually used, but that it was switched at frequent places throughout the entire life time of the robot, rather than being associated with a single distal task. We observed the same behavior in our EM networks, as they would activate the right motor module for obstacle avoidance, puck tracking, and to spin at the last stage.

Recall that the motivation behind the emergent modularity architecture is that the different modules will encourage the network to break the complex task of trash removal into a set of simpler subtasks. Observing the behavior of the evolved EM network, it certainly seems like the network is breaking the task up into two subtasks, grabbing a puck, and spinning until it can place the puck outside of the world. Though the EM network's strategy does not lead to as high fitness as that found by the Elman net, the strategy obviously divides the task into two distinct phases. And thus we can claim that Nolfi's architecture does successfully promote the division of the task. However, it is interesting that the mechanism through which division is implemented is not as simple as changing the network's selector neurons depending on the current subtask.

While the Elman network outperforms Nolfi's architecture in the long run it is interesting to note that even given random values, an EM network has a fairly good chance of displaying complex behaviors. Ordinarily a GA has to bootstrap its behavior, searching randomly for behaviors that will garner some fitness and thus help direct it towards more effective areas on which to focus. During this bootstrapping phase, the GA will not be able to differentiate between partial solutions and complete failures. If many of the complex behaviors that are common to the weight space of Nolfi's architecture provide these partial solutions, then the GA has a simpler task. Thus with this architecture a GA has fewer complete failures to go through in its bootstrapping phase and will thus have a shorter bootstrapping phase and more time to focus on developing solutions.

6 Future Work

While we do feel that the data we have managed to collect is significant, our experiment has been hampered by numerous technical difficulties. Many of these were resolved, but a few are still outstanding. We would like to be able to run our experiment for 1000 generations, giving us data symmetric to that in Nolfi's original paper. Unfortunately doing this would require tracking down the lurking bug in our simulation code - a task beyond our current means.

As our data reveals, there are a number of ways in which behaviors that have evolved in the stage simulator would not translate to a Khepera in the real world. The herding technique adopted by the Elman network would be a disaster if attempted by a real Khepera collecting cylindrical trash, as in the real world two objects cannot occupy the same space. The act of grabbing and releasing objects can also be accomplished instantaneously in the simulated world–another impossibility for a Khepera in the real world.

In order to legitimately challenge and extend Nolfi's experiment, we would need to alter the stage simulator to the point where behaviors evolved using it could be translated to an actual Khepera. This would require considerably improving the current collision detection and response algorithms, imposing a time penalty on gripper actions, properly implementing simulated IR sensors, and possibly modifying the robot motion logic to explicitly simulate 2 wheeled robots.

We would like to run tests using Elman and two-layer networks that contain a variety of hidden layer sizes, so that we could evaluate the value of the extra weights. When training networks with hidden layers, it is often advantageous to use a smaller hidden layer, as this encourages the network to create more compact representations of its sensor values. It would be interesting to thoroughly study the effects of hidden layer size of a network when the weight space is traversed by a genetic algorithm, rather than the gradient descent of backpropagation.

As Nolfi found in his experiment, we found that the EM network did not use several of its modules. In fact, in both experiments only the right motor module is ever switched. We could try reducing the number of modules available to the EM architecture by providing modules for the motors only. This would shrink the implied search space considerably, and thus potentially speed up the already intimidating convergence rate of Nolfi's architecture.

While somewhat outside the realm of simple extensions to Nolfi's work on emergent modularity, we think that the trash world experiment has the potential to provide an interesting context in which to study the interactions of evolution and learning. Nolfi and Floreano [7] argue that training a network to predict the next sensor states during its fitness trial can substantially improve the convergence speed of a GA. The Elman and two-layer networks used in our experiment could be given an additional six outputs nodes, which would represent the network's prediction of its next IR sensor states. Thus we could use the task and environment already built up for our earlier experiments in order to further test Nolfi and Floreano's hypothesis.

Nolfi and Floreano [7] also discuss a network in which half of the outputs act as teachers for the other half. Their analysis of this network indicates a sort of controlled instability in the weights that allows it to achieve its task. Perhaps having modules for the teacher function would allow it to control its behavior in interesting ways.

Since both Nolfi's emergent architecture and Elman networks provide reasonable results on this task, perhaps a more difficult task would provide more compelling evidence for or against one of them. For this reason our initial design of this experiment included two tasks. The second experiment, which we could not implement do to time constraints, was "puck segregation". Given a world that is split into two colors and pucks of each color, we require that the robot set the world into order. We had planned on providing the robot inputs which would specify the half of the world it was on and the color of the puck that it was holding. We expect that both Nolfi's EM and Elman networks would have found this task more challenging; and for that reason, the degree to which they succeed could provide important insights into the limitations of each.

We have made a point of avoiding biological metaphors when justifying our own work.

We use genetic algorithms because they have proven to be effective tools for searching high dimensional spaces, not out of any desire to argue for the biological plausibility of our experiments. For this reason an appropriate extension to our work would be to experiment with alternate search algorithms. As mentioned in our related work section, the parallel simulated annealer used by Kaplan and Salesin [4] has also proven effective at searching high dimensional spaces, and would probably have a greater chance of exploring more distinct behaviors than the genetic algorithm.

Recall that properly a genetic algorithm includes crossover, though in our own experiment we have followed Nolfi's lead and set the GA's crossover rate to zero. However, abandoning crossover limits a genetic algorithm's ability to cover a large section of the space effectively. For that reason it might be worth experimenting with algorithms such as Kaplan's, which keep the attention of the search algorithm spread over a larger range of potential solutions.

Nolfi states that he

> did not obtain better performance with respect to simulations without crossover ... This may be due to the fact that the crossover points were randomly chosen. Restricting the crossover points so as to preserve the organization of the network may produce better results (Montana, an Davis, 1989)[5]

Providing those restrictions and rerunning the GA with crossover would undoubtedly provide interesting information about the search spaces of our networks.

References

[1] K-Team S. A. *Khepera 2: User Manual*, version 1.1 edition, 2002.

[2] G. Cybenko. Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Systems*, 2:303–314.

[3] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[4] Craig S. Kaplan and David H. Salesin. Escherization. In Kurt Akeley, editor, *Siggraph 2000*, pages 499–510. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[5] D. J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 762–767, 1989.

[6] S. Nolfi. Using emergent modularity to develop control system for mobile robots, 1997.

[7] Stefano Nolfi and Dario Floreano. Learning and evolution, 1999.

Appendix: Simulating Khepera IR values

In Nolfi's emergent modularity experiments, the robots had to react to their environment given only the measurements provided by their IR sensors. Many of the more interesting behaviors that Nolfi observed seemed to have developed for the purpose of coping with the limitations of this sensor data.

The Khepera 2 User manual [1] provides information on the expected activation of the robot's IR sensors. Figure 8 charts the activation of the robot's four forward facing sensors, given a 5cm x 5cm white paper square surface placed in front of the Khepera at various distances.

We have modified the sensor values from one of stage's simulated scanning lasers to approximate a Khepera's IR sensors. The scanning laser shoots out 8 beams, spaced so as to evenly cover a 180 degrees arc directly ahead of the robot. We ignore the first and last beams, the positions of the remaining six roughly approximate the positions of the Khepera's IR sensors. The value returned by the scanning laser is the distance that the beam travels from the center of the robot before hitting an obstacle, or its maximum range in the case that the beam hits nothing (in our case, max range was 6cm). Because the diameter of the simulated robot is 5.5cm, it can detect objects up to 3.25cm in front of it. We generate our simulated IR values by considering any laser value closer to the robot's center than 3cm to create an IR activation of 0, we then scaling the last 3cm worth of data to approximate the values reported by the IR manual for an IR sensor with an obstacle dead ahead of it (see Figure 9).

There is a problem with our approximation method. As you can see from Figure 8, even the IR sensors not pointed directly at the obstacle will still pick up some amount of activation. However, if our laser beams miss an obstacle, they always return 0; there is no partial activation that results from having a beam close to an obstacle, as is the case with the IR sensors.

We imagine that even though the behavior of our simulated IR sensors are not quite like that of real IR sensors, the data that they return will create some of the same sorts of problems for the robot that real IR data would.

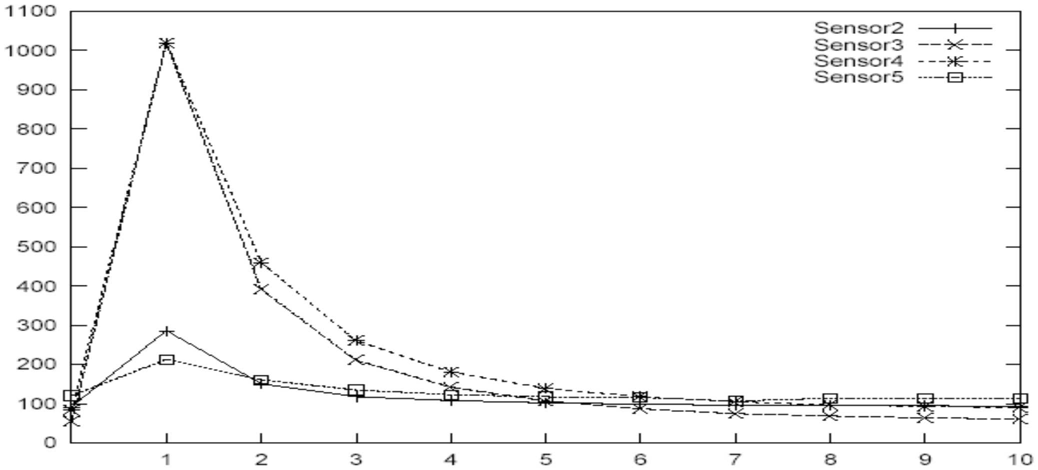

Figure 8: Actual IR values vs distance (cm)

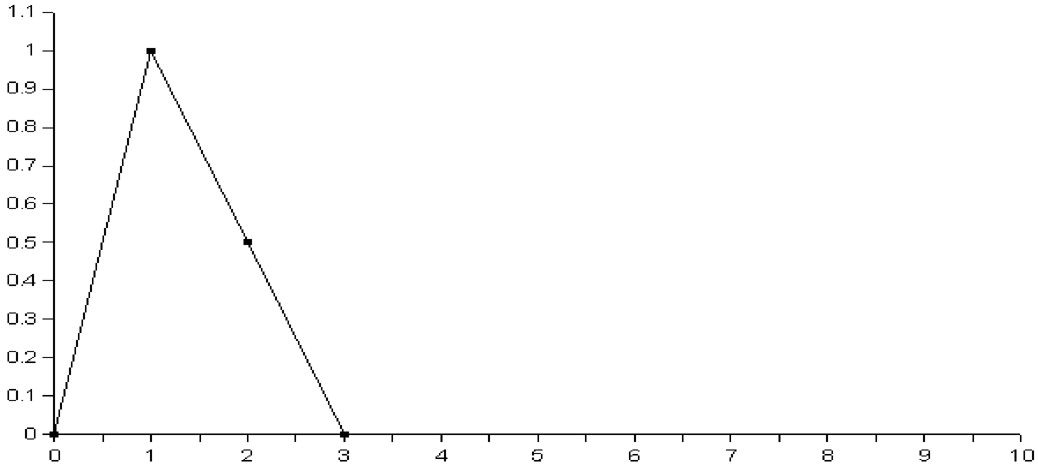

Figure 9: Approximated IR values vs distance (cm)