Learning and Evolution: A Comparison

Hollis Easter Frederick Heckel

May 2003

1 Abstract

We present a system for integrating genetic search and back-propagation learning in a neural network-based robot controller. We show experimentally that this combined approach leads to good results more quickly than with other methods, and that the eventual results are also better.

2 Introduction

Genetic algorithms and neural networks are commonly used mechanisms in artificial intelligence and robotics for a multitude of tasks. Other researchers have shown that GAs and connectionist networks can be combined successfully to generate more effective solutions to various tasks. Our project seeks to confirm those results in the domain of robot control. The specific task we are focusing on is that of wall following, a fairly simple problem for which decent controllers can be hand-crafted. The ease of creating a wall-following control system is an important point of the domain, as one of the relatively unique aspects of our experiment is the need for a teacher. Other work has used implicit teaching methods, such as input prediction, but we decided to explore the effects of using an explicit teacher for the learning portion of the task.

We believe that upon comparing the performance of robot controllers created in three different manners for this task, a combined approach of learning and genetic search will perform better than using either genetic search or learning only. We also believe that our results will reveal that the combination finds adequate solutions more quickly than either genetic search or learning alone.

3 Related Work

A substantial amount of work has been done regarding the interaction of learning and evolution with neural networks. These studies have generally found that a combination can make a great deal of difference for many tasks, and show that the starting state of the network can seriously affect the ability of the network to learn and perform well on a task.

Nolfi and Floreano's review[3] of several different research projects on the topic of interaction between learning and evolution served as the primary inspiration for this paper. None of the projects they discuss are exactly of the same type as our project; the closest project referred to is that of Hinton and Nowlan. Hinton and Nowlan's research used a combination of a genetic algorithm and learning to develop the network. From a broad perspective, it is very similar to this project, except for two major points: network connections as encoded in the gene are one of 0, 1 or a state modifiable by learning, and learning is treated as a **random process**. All weights in the network architecture we describe are modifiable by the learning process, so the networks do not tend to develop into patterns with mostly/entirely fixed states. In addition, we are using back-propagation learning as opposed to the random modifications made during learning by Hinton and Nowlan's network. One advantage to this random method of learning is that perhaps their method could still be effective in a domain without a teacher.

Nolfi and Parisi devote a large amount of time in another paper to explain what they see as flaws in Hinton and Nowlan's approach[4]. Their largest complaint is that the learning and evolutionary goals are the same, and explain what they consider a more effective way to combine learning and evolution. While we understand their complaints, we believe that the Hinton and Nowlan approach, when combined with a better learning method, is still very useful and effective.

The next paper explored by Nolfi and Floreano's review is Nolfi, Elman, and Parisi's research into networks which learn a task different from the one which they are evolved for[2]. This network controlled a simulated robot which earned fitness from consuming food in a grid world, but trained on different criteria. Instead of training to "find food", the network attempted to learn to predict its next sensor inputs. They found that this combination of learning and evolution also tended to guide evolution in a positive manner and resulted in individuals with a higher overall fitness. Our task is different from this task in that we are using the same goal for both learning and evolution(which is made trickier for needing to hand-craft an explicit teacher), but these networks use back-propagation learning like ours.

More explicitly, Kolen and Pollack showed that the initial conditions for back-propagation do make a significant difference in the ability of the network to learn and find a good solution[1]. Kolen and Pollack demonstrated, through trials with a variety of different starting conditions, that the x-or learning problem converges at greatly different rates based on the initial weights of the network. Based on the results of their experiment, they recommend that as good starting conditions and architectures are found for particular problems and networks, these conditions should be published so that future researchers need not waste time exploring a virtually infinite domain for appropriate initial weights.

4 The Experiment

4.1 Problem Domain

Wall-following is a useful behavior for autonomous robots. From basic room navigation to search and rescue, robots in many walks of life depend on being able to follow walls to get where they are going. Wall-following also has the virtue of being a relatively easy problem for which to write a teacher by hand. At the simplest level, a teacher can say "if there's something on my left, move me forward; otherwise, turn left". Teachers increase in complexity from there.

If it's so easy to write a wall-following agent controller by hand, why bother testing learning and evolution on it? Because the handcrafted agents, while function, do not generalize well, and often perform quite poorly. This is the sort of problem where New AI makes a strong case for itself. It seems plausible that a neural network, given hidden nodes and the opportunity for learning, will discover features of an environment that enable it to more effectively follow walls. Another advantage of the wall-following domain is that it proves fairly simple to tell whether a robot is actually succeeding at the task.

5 Method

5.1 Setup

We implemented our architecture using the ConX neural network brain package and the genetic algorithm package from Pyro[1]. Simulations were done using Player/Stage[2]. The simulated robot

[1]Python Robotics, a software package codirected by faculty at Swarthmore and Bryn Mawr Colleges.

[2]A simulator intended for multiple-entity use. It intentionally uses a somewhat lax sensor model to provide a computationally practicable simulation platform.

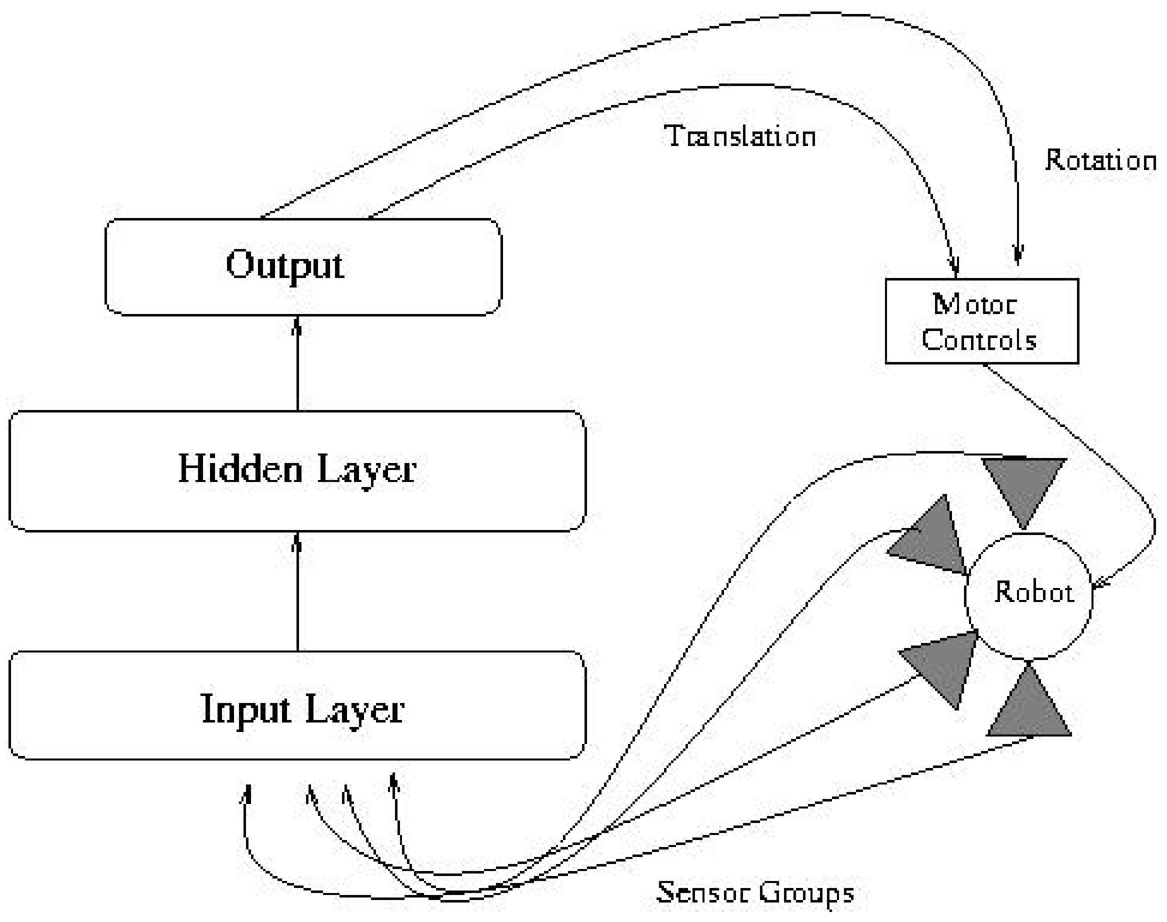

Figure 1: The controller's architecture. Four defined sensor groups present organized data.

was a version of the Pioneer2, using rangefinders and a truth device (to reset the robot's position in the world). ConX uses back propagation for learning, and the genetic algorithm uses real-valued genes to describe the weights between nodes in the network. The simulation world used was one of the standard worlds contained in the Player/Stage package, part of a hospital floor plan with a large number of walls and other obstacles.

Our experiment comes in three pieces: GA, NN, and GA+. GA is, as the name implies, a straight genetic algorithm. NN is a neural network simulation, with back-propagation learning. GA+ is a genetic algorithm that trains its genetically-created weights before beginning a separate run in the world to accrue fitness, with the intent of focusing fitness on the ability to learn effectively. All three experimental procedures use the same robot brain, with identical architecture.

The neural network (Figure 1) received information from only four sensor groups, rather than the full 16 range finding sensors. As the task it was intended to execute was that of lefthand wall following, we believed it need rely on only front, rear and left hand sensors. The groups used were front, front-left, back-left and rear. These four input nodes had connections to four hidden nodes which in turn connected to the two output nodes. Output from these nodes were used to control the translation and rotation of the simulated robot. For the two portions of the experiment which utilized learning, a handwritten lefthand wall-following brain was used as a teacher.

5.1.1 Fitness

Fitness was determined by the robot's speed and distance from a wall at each step. If the robot ran into a wall, the fitness trial was terminated before a full 200 steps had passed and a penalty incurred. Otherwise, fitness would accrue for a full 200 timesteps of 0.1 seconds. To clarify: running into walls is strongly penalized, as it incurs a percentage penalty as well as removing the possibility of accruing further fitness.

Robots earn fitness more quickly by moving at high speed, and by staying near walls. The wall function is a step multiplier function: when side sensors receive high activations, the fitness is added normally. When side sensors are low, but front or back sensors are high, fitness accrues at 50are in high activation state, there is a question of what to do.

Initially, we gave robots that were near no walls no fitness—a 0.0 multiplier. However, this seems a bit unreasonable, especially given that all robots start from predefined starting locations within the world, away from walls. Therefore, later test versions offer a 0.1 multiplier for movement away from walls, so as to allow robots to find walls without failing completely to earn fitness.

5.1.2 Test length

Two hundred generations of twenty individuals were run for the genetic algorithm only trials; GA+ trials ran for one hundred generations. Past experience shows that crossover does not tend to perform well in evolving neural networks. We did some trial runs with crossover turned on, and some with it turned off, and those that did not use it were, indeed, better. Elitism, though present in Pyro's GA implementation, was not enabled.

Training periods for the learning-only robots were approximately 600 steps of 0.1 seconds each. GA+ training periods were cut down to 200 steps. The shorter training period and smaller number of generations used for the GA+ trials were for the sake of time, and also to more convincingly show the success(or failure) of GA+ in achieving our goal. Given that neural network training is very much dependent on a significant amount of experiential learning, it makes sense that, should GA+ behave well while handicapped by shorter learning, it will have greater applicability.

During fitness trials, learning was turned off for GA+ and backprop-only trials. Starting locations for all fitness trials were chosen from one of six random locations preselected in the Player/Stage hospital world. During backprop learning, if the robot ran into a wall, its position was reset to its starting location for the learning period. It is important to distinguish between training runs for the methods involving learning from testing runs for all three methods. NN and GA+ were given a chance to train in the environment for a set period of time before beginning their fitness runs. If they collided with walls *during their training runs* their poses were reset to those where they began.

Whenever a fitness trial begins, learning is turned off, and the robot is placed at a random starting location in the world, whether or not it has previously trained. It then begins a run through the world, accruing fitness as it goes. If a robot hits anything during a fitness run, the penalty is assessed and the run ended.

5.2 GA

The GA method was fairly straightforward. Parameters were, for the most part, set to Pyro defaults. Genes were allowed to vary in real number space between -1.0 and 1.0. Selection rate was set to 0.8, mutation rate to 0.3, crossover rate to 0.5 or 0.0, depending on the test, and the tests were capped at 200 generations.

When created by the genetic algorithm, network weights were simply loaded into the robot, which was then placed in the environment and allowed a test run.

5.3 NN

The NN method was, again, quite straightforward. The network was a three-layer network of 4 input nodes, 4 hidden nodes, and 4 output nodes. Learning momentum was set to 0.1, and ϵ was set to 0.75, by arbitrary fiat. The NN method was given 600 training steps before beginning its testing run, which we hoped would allow it to counteract the lack of contextual learning or non-random initialization.

5.4 GA+

The GA+ method combines the previous two, in the hopes of biasing evolution toward the ability learn quickly and well. GA+ uses backpropagation learning as part of a fitness algorithm to test the evolved weights. It is important to note that this is normal or "real" evolution and learning, not Lamarckian evolution; individuals pass on their genotype, not their phenotype, and so learning has no effect on evolution other than through the fitness function.

The genetic search provides a set of base weights, which are loaded into a brain and allowed to train for 200 time steps. If the robot runs into objects, the robot is simply replaced at its original starting position. After training, the robot is placed at a random starting location and allowed to start its run. If the robot collides with objects during the testing run, its run is considered ended.

GA+ is the most time- and computation-intensive of our tests, because it uses heavy-duty backpropagation on every individual.

6 Results

Our hypothesis was that the quality of solution produced by straight genetic search would be the lowest among the group, and that those produced by GA+ would be of far higher quality. We supposed that learning would play an intermediate part, strongly outperforming a genetic algorithm, but falling to GA+. In addition, we supposed that a neural network would come up with a "decent" solution[3] quite a bit faster than the GA would, and that the GA+ would yield the quickest "decent" solutions.

In point of fact, the question of a quick solution turns out to be less intuitive than we thought. A quick solution for a GA-based approach is one that takes relatively few generations of relatively few individuals—so a solution that requires a small population of smart controllers. But for a learning-based approach, what defines speed? Is it the number of training steps undertaken? Each NN learning-based test stands on its own, without the benefit of surrounding context from the others. Number of steps seems the only reasonable option—we hope to test it in the future.

We did three sets of trials for the GA and GA+ modules, and four sets for the NN module. For the GA-based modules, we present a representative sampling of fitness through time; for NN, this is unintuitive, and so we present simply the "winner", or best fitness score, from each generation.

Blanks in the tables mean that no analysis was available for the specified field. The second trial was cut short by machine crashes, and so the data are small: the best GA score was 86.6 in generation 19; the best NN was 7.4; and the best GA+ was 72.2 in generation 10. Both GA tests used crossover. The first GA+ test's output file seems to have gotten corrupted—the first 8 generations are filled with @ symbols.

7 Discussion and Conclusion

Perhaps the most surprising aspect of our experiment was the execrable performance of the neural network learning method. Even when given 600 time steps for learning, the best NN performance was

[3]One that is 'good enough', for however that ends up being arbitrarily defined. 'Good enough' is easy to know when you see it, but harder to legislate using cold statistics.

Generation	Trial 1 Best/Avg	Trial 3 Best/Avg
1	46.4/14.1	33.7/15.5
2	17.9/5.02	48.3/13.6
3	29.7/11.2	44.3/18.9
4	46.7/10.3	59.6/25.2
5	50.6/9.75	64.4/21.9
6	44.1/11.5	40.5/12.2
7	53.8/12.3	50.2/14.2
8	59.6/13.7	41.9/15.9
9	24.9/6.65	50.7/13.1
10	30.0/9.82	47.3/13.0
11		54.5/26.7
12		60.4/15.9
13		49.0/13.5
14		61.8/22.9
15		57.4/28.8
16		60.3/19.9
17		55.2/14.9
18		57.3/15.5
19		58.9/26.5
20	62.6/13.4	60.6/21.8
25		87.4/33.3
30	62.5/24.2	73.9/29.6
35		80.4/35.4
160	70.6/14.8	

Figure 2: GA results

Trial	Best
1	8.37
2	7.4
3	6.9
4	24.9

Figure 3: NN results

Generation	Trial 1 Best/Avg	Trial 3 Best/Avg
1	missing	37.4/15.1
2	missing	26.0/9.5
3	missing	42.4/9.7
4	missing	50.7/14.0
5	missing	54.0/16.8
6	missing	71.2/17.4
7	missing	77.8/16.7
8	missing	64.9/17.4
9	58.8/13.9	76.8/23.2
10	50.4/21.4	59.4/23.7
11		72.7/29.7
12		69.8/22.5
13		60.6/23.5
14		57.2/19.2
15		71.8/25.6
16		54.5/18.6
17		57.0/19.0
20	79.5/30.1	
30	54.1/13.6	
40	75.1/27.3	
50	67.3/20.0	
60	75.2/23.4	
70	78.5/36.4	
80	66.8/23.8	
90	75.3/30.4	
100	67.8	

Figure 4: GA+ results

24.9, far below the levels reached by the others. The definition of the teacher seems unlikely to be the problem, given that the teacher can be used as a standalone brain that behaves well.

It seems likely, therefore, that genetic search has a significant advantage through the fact that most runs start with a non-random initialization. This presentation of context seems to have a strong effect on the fitness of the individuals in question. With or without learning, genetic search seems to beat straight learning hands down, usually by a multiplier of 2–5.

GA versus GA+ is a slightly muddier picture. Among other questions is that of which counts for more: best score, or average score. Both pieces of data provide useful information, which is why we have included them here. Best scores, obviously, indicate the best solutions that a particular system can find. To some degree, this translates into being the best out-of-the-box behavior you can hope for–the best that a given system can hope for, given correct placement and good genes.

Average value is, perhaps, more interesting. Here, GA+ has clear advantages over straight GA: while GA frequently has high scores, occasionally better than GA+'s, its averages start low and stay there. In general, GA+'s high scores are higher than those of GA, and the average is also higher.

Another point of interest is the ratio of high to average scores. For straight GA, this ratio tends to be somewhere around 0.27, meaning that the high score is usually about 4 times better than the average. For GA+, the ratio is closer to 0.36, with the high score around 3 times better than average. The range between excellent and average is thus smaller for GA+ than for GA.

This leads to one of our earliest questions: which solution is best for finding a "good enough" solution? We see reason for cautious support of GA+ in this area. Both significant GA+ tests began finding good solutions very quickly, and their average solutions quickly increase to the point where they nearly always beat the best NN solutions. GA+ is therefore desirable for both speed and quality of solution.

GA+, and anything derived from it, cannot ever be truly autonomous, hands-free robotics. The need for a teacher, be it handcrafted or otherwise, makes this approach unsuitable for those domains in which teachers cannot easily be made. This approach becomes a bit more feasible for improving existing systems–the pre-existing system could be made into the teacher for a GA+ system. Nevertheless, the teacher is critical to GA+'s function.

Genetic search is prone to self-exploitation. Fitness functions frequently leave loopholes for a genetic algorithm to find. GA+ will work will in other domains, we think, so long as fitness functions are carefully made.

7.1 Future Directions

Among the things we would like to try with GA+, given more time, are

1. Tests on different network architectures. We theorize that GA+'s advantages would manifest themselves even more prominently on a network with more nodes and connections.

2. More tests, with larger generations. These should converge somewhat faster, at least in terms of number of generations.

3. Longer training times for the two learning brains. In particular, longer training for the straight NN brain might allow it to perform better, as might placing it randomly during training when it hits walls, rather than replacing it in its starting location.

4. Mid-training tests for the NN brain, to see how many training steps it takes to get a "decent" solution.

5. Variable-length fitness runs. It is possible that the winning strategies are optimized for short runs, though random placement takes care of a lot of this. Varying the number of steps in a fitness run by $\pm n$ might inject more randomness, which would be good.

6. Lamarckian evolution versus “real” evolution. This sort of system provides an ideal testing ground for discovering whether it would be better to pass on phenotype rather than genotype for the brain.

References

[1] John F. Kolen and Jordan B. Pollack. Back propagation is sensitive to initial conditions. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 860–867. Morgan Kaufmann Publishers, Inc., 1991.

[2] Stefano Nolfi, Jeffrey L. Elman, and Domenico Parisi. Learning and evolution in neural networks. Technical Report 9019, 1990.

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

[4] Domenico Parisi and Stefano Nolfi. The influence of learning on evolution. In Richard K. Belew and Melanie Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and Algorithms*, pages 419–428. Addison Wesley, Reading, MA, 1996.