Evolving an Adaptable Robot

Benjamin Turner and Stephen St.Vincent

15 May 2006

Abstract

In this paper we present an experiment designed to integrate the developmental concepts of learning and evolution. The motivation for this integreation is the observation that a design for a robot which is highly specific to one environment is of little use outside that environment. Thus, robot control systems should incorporate a sense of adaptability if they are to be generally useful. Our experiment uses a genetic algorithm to evolve a population of robots which are then placed in several different environments and allowed to learn tasks specific to those environments over the course of their lifetime. Although our results are largely inconclusive, we have established a methodology for combining evolution, learning, and multiple robot environments, and have seen indications of the possible success of this approach.

1 Introduction

One of the aims of developmental robotics is to create a robot that is adaptable to a variety of tasks. An adaptable robot would not be engineered to perform a particular task, but rather have a general set of low-level motivations and behaviors from which it could learn the task presented to it. This paper describes one attempt to develop such a robot.

We use a combination of evolutionary computation and neural network learning to evolve a robot brain which can learn to perform three separate tasks. The goal is for the evolved robot to out-perform a randomly generated neural network brain with the same topology.

Our work is inspired largely by the research of Nolfi, who has demonstrated the effectiveness of an evolutionary approach in tasks such as distinguishing between objects in an environment [5] and moving objects outside the perimeter of an arena [3]. A third paper discusses the usefulness of combining learning with evolution [4].

The paper is organized as follows. Section 2 discusses the experimental setup and methodology. Section 3 describes the performance of the evolved robot, and compares it to a benchmark brain. Section 4 provides a discussion of the results.

2 Experimental Description

Our experiment consists of evaluating the performance of a population of individuals on three separate tasks, which we refer to as Wall-Following, Chasing, and Escaping (see section 2.4). Each individual consists of a neural network brain whose initial weights, learning parameters, and topology are determined by an evolved genotype. Within its lifetime, an individual can change its network weights through complementary reinforcement backpropagation. The evolutionary fitness of each individual is a function of its performance, relative to the performance of each other individual, on all three tasks. We use the Pyro (Python Robotics [1]) software package to run the experiment in simulation.

2.1 Network Architecture

Each individual robot's brain is a standard three-layer neural network with 11 input nodes, a variable number of hidden nodes (initially 10), and 16 output nodes. Figure 1 shows this network architecture. The inputs to the network are the floating-point values of eight sonars (see figure 3 for a visualization of the sonars), plus a three-dimensional binary "position vector" (figure 2 explains the position vector in detail). This vector is necessary because two of the three tasks involve a second robot (called the "target robot") whose behavior is human-engineered, and the developing robot needs to be given information about where the target robot is in relation to itself (see sections 2.4.2 and 2.4.3 for more on these tasks). The outputs are two 8-bit vectors which are used to stochastically determine the robot's translational and rotational speed and direction.

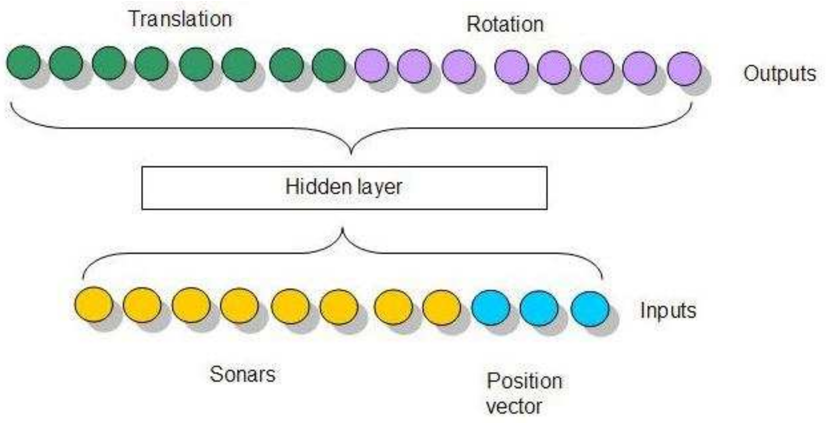

Figure 1: Network Architecture. The input layer consists of 8 real-valued sonar inputs and a binary, three-component position vector indicating the position of the target robot in the world. The hidden layer is of variable size as determined by the genetic algorithm. The output layer has 16 nodes: the first 8 are used to stochastically determine speed and direction of translation, and the latter 8 are used to determine speed and direction of rotation.

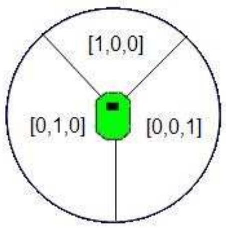

Figure 2: The Position Vector. This vector has three possible values for the Chasing and Escaping tasks; which one is passed in on a given timestep is dependent on which of three sectors the target robot is in relative to the orientation of the developing robot (in front, to the left, or to the right). Note that in the Wall-Following task, there is no other robot. In this case, the vector [0,0,0] is passed to the input layer. In this image, the robot is facing upwards towards the vector [1,0,0].

2.2 Learning via Complementary Reinforcement Backpropagation

Each individual neural network brain in the population can perform on-line learning during the timesteps it is allowed to run on each task. The networks learn through a technique called Complementary Reinforcement Backpropagation (CRBP). CRBP is a variant of the standard neural network backpropagation algorithm in which a specific output need not be specified for the backpropagation to work[1]. Instead, the target values for the output nodes are determined stochastically based on whether the robot received a reward or a punishment for its behavior on a given timestep. Determining the output targets is a four-step process:

1. A binary output vector B is generated from the actual output vector O by treating each element of O (whose values are floating-point decimals that range from 0 to 1) as the probability that the corresponding element of B should be a 1.
2. If the feedback the robot received on this timestep was positive (i.e. the robot was rewarded), then set the target vector, T, equal to B, and set the learning rate equal to *rewardEpsilon* (see below).
3. If the feedback the robot received on this timestep was negative (i.e. the robot was punished), then set T equal to the inverse of B, and set the learning rate equal to *punishEpsilon* (again, see below).
4. Calculate an error by comparing T to O, and backpropagate as normal.

This implementation of CRBP is based on the one found in [2].

The two possible learning rates (*rewardEpsilon* and *punishEpsilon*) allow the impact of reward on network weights to be different than the impact of punishment. *punishEpsilon* is always set to $\frac{1}{3}$ of *rewardEpsilon.* Initially, *rewardEpsilon* is set to 0.3 (the Pyro default; *punishEpsilon* is therefor 0.1), but this parameter can be mutated during evolution. In addition, network tolerance and momentum are initially set to the Pyro defaults of 0.4 and 0.9, respectively; these values could also be changed during evolution.

Feedback is not simply a matter of positive and negative rewards for a given timestep. We take the average fitness values of the previous five timesteps (see the task descriptions for details about fitness values)

[1] Normal backpropagation requires that, for all possible inputs to the network x, a specific desired output $f(x)$ must be given. Backpropagation compares this target output to the actual output of the neural network, calculates an error, and propagates this error backward to update the weight of each hidden and input node.

Network component	**Range**	**Max possible change**
Connection weight	[-10,10]	±10
Reward learning rate	[0,1]	±0.1
Punishment learning rate	[0,0.33]	±0.1
Momentum	[0,1]	±0.1
Number of hidden nodes	$[0,\infty)$	±1

Table 1: Mutatable Network Values. The range indicates valid values for the parameters, while the maximum possible change indicates the maximum values by which the genetic algorithm can mutate the parameters in one generation.

and subtract that average from the fitness value of the current timestep. This new value is taken to be the reward for that timestep. This way, the brain gets rewarded for doing better than it had been recently.

It is important to note that the network weights learned by an individual during its lifetime are not the weights that are passed on to the next generation of individuals (see section 2.3). That is, we employ Darwinian evolution (in which genotypes are passed on) rather than Lamarckian evolution (in which phenotypes are passed on).

2.3 The Genetic Algorithm

A single generation in the evolutionary process consists of each individual in the population running for 1000 timesteps on each of the three tasks. During each of these tasks, an individual receives rewards for its performance which are specific to the particular task (see sections 2.4.1 - 2.4.3 for details on the reward metrics). The genetic algorithm keeps track of the sum totals of these three rewards for each individual. At the end of the generation, each robot receives three rankings according to its performance on each task (relative to the other robots); for simplicity, higher rankings imply better performance. These three rankings are averaged to produce an overall fitness measure; the robot that has the highest average ranking across all tasks is considered the best.

Once this ranked fitness has been determined for each individual, we create the next generation stochastically. Each individual from generation x is assigned a probability that it will propagate to generation $x+1$. The higher an individual's fitness, the higher its probability of propagation. A new generation of individuals is then created by randomly selecting the individuals from this probability distribution. The reasoning behind this method is that while the more fit individuals have a higher probability of propagation (the most fit individuals will likely be copied multiple times, while the least fit are rarely, if ever, copied), possible evolutionarily advantageous mutations that do not happen to score well can still propagate through.

After the creation of the new generation, each individual in the new generation is subject to random mutation (we do not employ crossover). The following components of any individual's neural network brain can potentially be mutated: connection between nodes; the reward learning rate (and thus the punishment learning rate); the network momentum; the network tolerance; and the number of hidden nodes. Table 1 shows the maximum amount by which each of these parameters can be changed in a single mutation, in addition to its range of possible values. When a new hidden node is added, its connection weights are randomly initialized; when a hidden node is removed, its connections are destroyed. When all individuals have been subjected to possible mutation, the next generation is ready to begin learning.

In this experiment, there were 100 generations of 30 individuals each.

2.4 The Three Tasks

The three tasks on which we test our evolving robots are Wall-Following, Escaping and Chasing. The robot's goal in the Wall-Following task is to keep its left side close to, but not touching, a wall. In the Escaping task, it must stay as far away as it can from a second robot programmed to move towards it. In the Chasing task, the developing robot must stay as close as possible to a robot programmed to run away from it.

2.4.1 Wall-Following

The environment for the Wall-Following task is a simple square room with no obstacles and no other robots (figure 3). The reward scheme is as follows:

1. If the robot is not moving at all, it is punished.
2. If the robot is too close to a wall (within 0.1 robot lengths), it is punished.
3. Otherwise, evolutionary fitness scales as the inverse of distance from the wall.
4. To calculate a final reward, this fitness value is compared to the average fitnes value that the robot received on the previous five timesteps. If it is higher, the robot is rewarded; if it is lower, the robot is punished.

After the robot completes its run in this envirnonment, its network weights are reset to the values specified by its genotype (that is, the robot "forgets" what it has learned about this particular task) and it begins running on the next task.

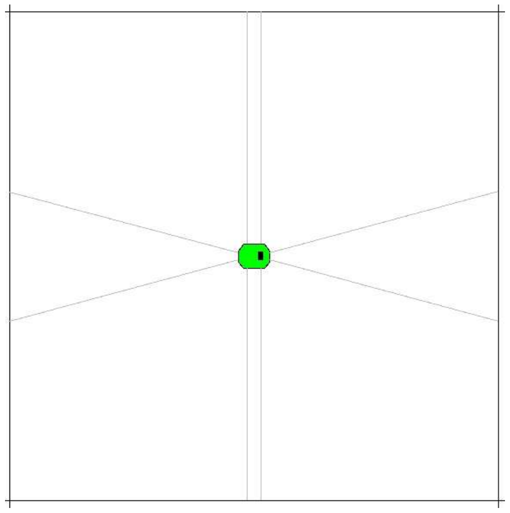

Figure 3: Wall-Following Environment. The lines emanating from the green robot show the positions of the 8 sonars, whose values are fed to the input layer of the neural network.

2.4.2 Escaping

The environment for the Escaping task is identical to that of the Wall-following task, except for the addition of a second robot, which we call the target robot (figure 4). In this task, the target robot is programmed to "chase" the developing robot–that is, on each timestep, it attempts to reduce the distance between the two robots[2]. The reward scheme for this environment is simpler than that for Wall-following, since we no longer explicitly punish the robot for being motionless. There are three rules:

1. If the robot is too close to a wall (within 0.1 robot lengths), it is punished.
2. Otherwise, evolutionary fitness scales with distance between the two robots.
3. Final reward is determined as in step 4 in the Wall-following task (section 2.4.1).

After the robot's run in this environment is complete, the network weights are reset to those specified in the genotype, and the robot begins running the third and final task.

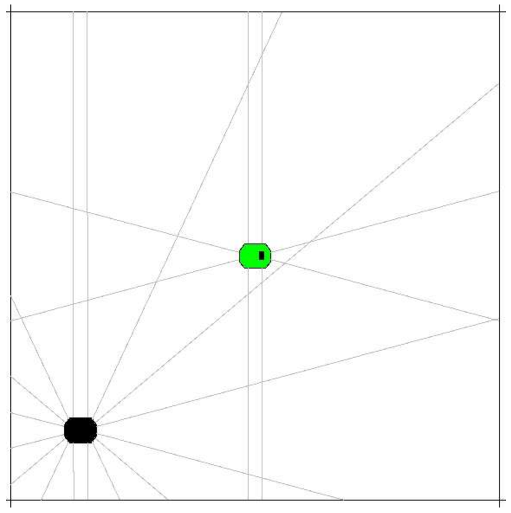

Figure 4: The Chasing/Escaping Environment. The developing robot is initially placed in the center of the room; the target robot is placed in the lower left-hand corner.

2.4.3 Chasing

The environment for the Chasing task is identical to that of the Escaping task (figure 4). The difference is in the behavior of the target robot, which is now programmed to run away from the developing robot by trying to increase the distance between the two robots, while also avoiding walls. The reward scheme for this task is exactly the same as that for the Escaping task, except that now the robot is rewarded if it is closer to the target than before (as opposed to farther away, as in Escaping), and evolutionary fitness for a particular timestep scales inversely with distance between the two robots.

[2]It should be noted here that for the Chasing and Escaping tasks, the maximum possible translation speed for the target robot changed throughout evolution. For the first generation, the target robots could not move at all. From there, their speeds increased incrementally with each generation, finally reaching the maximum speed on the last generation.

After completing its run in this environment, the lifetime of this individual is complete. The genetic algorithm has recorded the total fitness of this individual on all three tasks, and will use this information to rank it against the other individuals in its generation.

3 Results

To evaluate the performance of the evolved robots, we save the genotype of the most fit individual from the each generation. At the end of the final generation, we ran the most fit robot's brain through each task multiple times, recording the fitness it received. Then we did the same for a non-evolved brain (initialized with random network weights) with the same topology as the most fit evolved individual. We refer to this as the "dummy brain," since although it is allowed to learn in the same fashion as the evolved brain, its initial configuration has not been at all optimized.

3.1 GA/Learning results

The best individual on from the 100th (final) generation had the following attributes:

- Hidden layer size: 9
- rewardEpsilon: 0.19 (implicitly, punishEpsilon=0.06)
- Tolerance: 0.46
- Momentum: 0.87

Below we present the results of 10 trials of this individual on the three tasks.

Trial	**Wall-Following**	**Chasing**	**Escaping**
1	1771.95	1586.48	369.87
2	-211.98	-481.57	-429.87
3	861.40	331.45	-59.23
4	1037.46	-394.23	-184.79
5	1329.18	346.44	146.10
6	1056.30	305.19	-24.07
7	325.58	432.39	138.05
8	1958.24	338.17	60.72
9	1836.13	-822.74	-431.29
10	561.5	367.23	284.81
Mean	**1052.58**	**200.88**	**-12.97**
Standard Deviation	**701.88**	**661.38**	**272.72**

Table 2: Evolved Brain Results. We ran 10 trials of the evolved genotype separately on each of the three tasks. The values above are the sums of the fitness values achieved during the lifetime of each individual on each task.

3.2 Dummy Brain results

Below are the results of the dummy brain, with *rewardEpsilon*, *tolerance*, and *momentum* set to their defaults as designated by Pyro (0.3, 0.4, and 0.9, respectively). All network weights were initialized to random values.

Trial	Wall-Following	Chasing	Escaping
1	68.05	313.04	10.67
2	-331.86	328.10	214.43
3	71.40	-518.52	-331.59
4	750.42	348.06	0.28
5	929.82	225.10	36.11
6	1041.30	263.33	188.42
7	774.78	373.00	152.25
8	1458.05	197.28	268.06
9	1144.79	307.77	179.85
10	1198.83	-22.73	184.77
Mean	**710.56**	**181.44**	**90.33**
Standard Deviation	**582.80**	**270.75**	**174.10**

Table 3: Dummy Brain Results. These are determined in the same manner as the evolved brain results described in table 2.

3.3 Comparison

To see if these sets of results are statistically significantly different, we subtract the means (μ) of the two data sets to find the mean difference and add the standard deviations (σ) in quadrature to find the overall standard deviation:

$$\mu_{diff} = \mu_{GA} - \mu_{dummy} \tag{1}$$

$$\sigma_{diff} = \sqrt{\sigma_{GA}^2 + \sigma_{dummy}^2} \tag{2}$$

Here, the subscript *GA* indicates the values for the evolved brain, *dummy* indicates the values of the non-evolved brain, and *diff* represents the difference of the two.

For the three tasks, we have the following results:

Task	μ_{diff}	σ_{total}	Statistical significance
Wall-Following	342.02	912.30	0.14
Chasing	19.44	714.65	0.03
Escaping	-103.3	323.55	0.13

Table 4: Comparison Statistics. This table shows the results of the comparison of the evolved brain to dummy brain. The "Statistical Significance" column measures the probability that the difference of the means (μ_{diff}) differs from zero. Clearly, none of the results differ significantly from zero.

4 Discussion

Based on the quantitative results presented in section 3, we were disappointed with the overall performance of the most-evolved individual relative to a non-evolved brain. It appears that the evolutionary computation has not significantly improved the robot's ability to adapt to multiple tasks. However, there are some promising aspects to this data. If we examine the highest performance by the evolved individual on any single task (table 2) and compare it to the highest performance by a non-evolved individual (table 3), we can see that, in every case, the evolved individual out-performed the non-evolved one. In addition, the first trial of the evolved individual out-performs all trials of the non-evolved individual on all tasks. This suggests that it is indeed possible to evolve a set of initial network weights and parameters that is more able to adapt to its environment than a randomly generated brain. Of course, the fact that this comparison does not hold for all evolved individuals against all non-evolved individuals means that no general conclusion of this sort can be drawn.

These results demonstrate both the advantages and disadvantages of the evolutionary approach. On one hand, we have established a solid methodology for the evolution of network brains using fitness determined by performance across a variety of tasks, and have seen promise in the initial experimental results. On the other hand, our results fail to show anything conclusive, which is at least partially due to the large time cost incurred when using a genetic algorithm. Perhaps, if allotted more time to run the experiment for more generations, with more individuals, or with longer lifespans per individual, we would have seen more conclusive results.

5 Conclusion

The goal of this paper was to evolve a neural network brain which could learn to perform well on any task. We have not been able to evolve such a brain, but we have demonstrated the potential of a genetic algorithm to do so.

Possible modifications of the experiment that may increase its success include:

- A higher number of individuals in the population
- Allowing the experiment to run for a larger number of generations
- Extending the lifespan of each individual on each task
- Simplifying the tasks which the robots must learn
- Using a different network architecture, for example a cascade correlation network [6] rather than a standard three-layer network

Any future experiment based on this one should keep in mind the lessons we have learned: that evolution is a lengthy process, and when combined with neural network learning, there is a high degree of variance in the performance of even the same individual on multiple runs of the same task.

References

[1] D. Blank, L. Meeden, and D. Kumar. Python robotics: an environment for exploring robotics beyond legos. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 317–321. ACM Press, 2003.

[2] L. Meeden. An incremental approach to developing intelligent neural network controllers for robots, 1995.

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

[4] S. Nolfi and D. Floreano. Learning and evolution, 1999.

[5] S. Nolfi and D. Marocco. Evolving robots able to visually discriminate between objects of different size. *Connection Science*, 14:163–170, 2002.

[6] Thomas Schultz. *Computational Developmental Psychology.* The MIT Press, 2003.