Grounding Robotic Control with Genetic Neural Networks

Diane Law and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin, Austin, TX 78712
dianelaw,risto@cs.utexas.edu

Technical Report AI94-223
May 1994

Abstract

An important but often neglected problem in the field of Artificial Intelligence is that of grounding systems in their environment such that the representations they manipulate have inherent meaning for the system. Since humans rely so heavily on semantics, it seems likely that the grounding is crucial to the development of truly intelligent behavior. This study investigates the use of simulated robotic agents with neural network processors as part of a method to ensure grounding. Both the topology and weights of the neural networks are optimized through genetic algorithms. Although such comprehensive optimization is difficult, the empirical evidence gathered here shows that the method is not only tractable but quite fruitful. In the experiments, the agents evolved a wall-following control strategy and were able to transfer it to novel environments. Their behavior suggests that they were also learning to build cognitive maps.

1 Introduction

In his Chinese Room thought experiment, John Searle (1980) posed a serious challenge to Artificial Intelligence. He argued that the real problem with the computational theory of mind as espoused by AI Functionalism is that it leaves out semantics, since this theory implies that intelligence requires only the right kind of symbolic manipulation (i.e., syntax alone). His article provoked a spate of replies. Although most of them make important points, none have really addressed Searle's main objection. In order to answer his argument directly we must stipulate causal connections between the environment and the system. If we do not, there can be no referents for the symbol structures that the system manipulates and the system must therefore be devoid of semantics. This is not only a philosophical problem; the lack of semantics in traditional AI programs has given rise to serious practical problems as well. Cases in point are the well-known frame problem (McCarthy and Hayes, 1969), the ability to react appropriately in novel situations and the problem of scalability in general.

How can we address this requirement for Artificial Intelligence? In a LISP program, the symbol `obstacle` means no more to the machine than does `G1089`, although it means something to the programmer or to a human user. As Harnad (1990) puts it, "How can the semantic interpretation of

a formal symbol system be made *intrinsic* to the system, rather than just parasitic on the meanings in our heads?" In order to meet this requirement, first, it is necessary for the system to be able to interact causally with its environment. This implies embedding the system in a robotic agent. Secondly, there seems at present to be no alternative to using neural networks, since this is the only computational method available that can map sensory inputs directly to internal representations without the explicit intervention of the programmer. Lakoff, arguing for the necessity of using neural networks in conjunction with a sensorimotor system connects these two requirements:

> One cannot just arbitrarily assign meaning to activation patterns over neural networks that are connected to [a] sensorimotor system. The nature of the hookup to the body will make such an activation pattern meaningful and play a role in fixing its meaning (1988).

If we can agree that the knowledge that an agent eventually has must be learned, rather than hard-coded, then we will also want to avoid building too much a priori knowledge into the neural network that represents its processor. However, we still require a network that is compact and that is sufficiently organized to do the job. We want the demands of the task itself to impose structure on the network and we want the network to learn from the environment, rather than from an explicit teacher. These prerequisites leave us with evolution as a guiding principle and thus, the natural choice for developing such a network is the use of a genetic algorithm (Holland, 1975; Goldberg, 1989), which optimizes the networks both in terms of connection strength and topology.

This project is a first step towards building such a grounded system, starting from the most basic capabilities. Since the first requirement for a mobile robotic agent is merely to be able to move about in a reasonable fashion, it is necessary to begin with basic control strategies. It is important that the robot be able to roam its environment without getting stuck up against obstacles and to have some strategy for determining (at least roughly) where it is, as well as a strategy for moving reliably from one point to another. The idea is similar to one presented as the Critter Problem, originally formulated by Ron Rivest in 1984 (Kuipers 1985). The agent is placed in an environment with a set of sensors and effectors, but without any a priori knowledge of the relationship between its sensors, effectors and surroundings. The task is to learn by experience what those relationships are and to learn to respond appropriately in a wide range of situations.

While there are several groups doing similar research, all differ in some respects from the current approach. Brooks' (1991) subsumption architecture is an attempt to control robot behavior by reaction to the environment, but the emphasis is not on learning the relation between sensors and effectors and much more knowledge must be built into the system. Pfeifer and Verschure (1992) are interested in grounded robotic agents, but they specify the network architecture, explicitly building in several assumptions and basic control strategies. Similarly, Patel and Schnepf (1992) also start with basic control rules, proposing the use of modular parallel learning classifier systems to elicit controlled robotic behavior. In the area of tabula rasa learning, Rivest and Schapire (1993) have approached the problem with deterministic finite state automata that depend on a discrete environment, while Pierce and Kuipers(1991) have used reinforcement learning. Collins and Jefferson's Ant Farm (1990) is closest in spirit to the present study, since they also rely on neural networks developed by genetic algorithms to simulate foraging behavior, but the environment and movement are discrete, and wandering behavior is entirely random during the search phase, while it is directed by explicit pheromone paths during the recruitment phase. The approach developed here, in contrast, explicitly rejects built-in task-specific knowledge, works within a continuous (simulated) environment and leaves the entire structure of the processing machinery up to evolution. Behavior

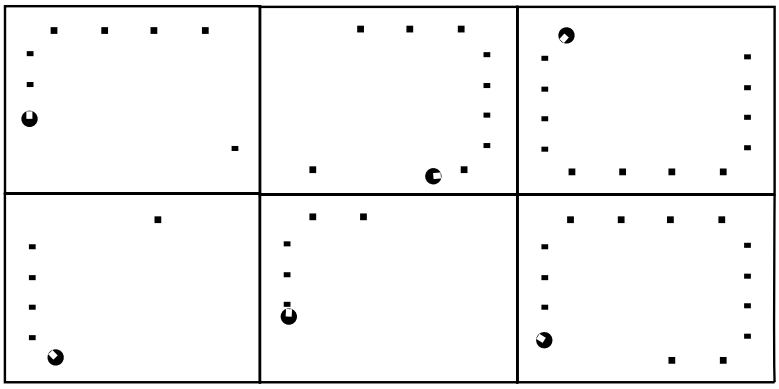

Figure 1: **Screen image showing the environments for six of the robots.** The robots are represented as small circles, each with a white radius to show current orientation. The small black dots around the perimeters are markers used to measure wall-following behavior.

is not elicited through any sort of external feedback, such as that provided by supervised learning techniques, but by a much more general, unsupervised method.

2 Task description and representation

This project was essentially a feasibility study to see whether neural networks developed through genetic algorithms would be able to learn the connection between the input they receive from the environment and the appropriate motor outputs that would allow them to pursue a wall-following strategy. Wall-following is one of the basic control mechanisms for robotic agents and is a non-trivial task, since walls must simultaneously attract and repel the robots. This ability is fundamental, since unless the agents can move systematically, they cannot begin to map their environments, nor pursue higher-level goals.

2.1 The robots and their environment

There were 196 simulated agents per generation and each was placed in a random location in a separate room-like environment with a random initial orientation (figure 1). During training, this room was a simple rectangle. The agents themselves are represented as a circle with a diameter of 5 pixels and are equipped with a bump sensor and 10 ray-tracing range-finders, arranged at 20-degree increments in a semi-circle around the front of the robots. These sensors were chosen because they are common on physical robots. The agents produce three numbers as output, determining the agents' movement. The first specifies the change in angular direction (between 0 and 180 degrees), the second determines the distance to travel and the last indicates whether the change in direction should be positive or negative (i.e., a turn to the left or to the right). The cell for each robot measures 72 by 54 pixels and the range-finders can detect an object within no more than 15 pixels, so this area qualifies as large-scale space. The agents must move between two and fourteen pixels on each iteration. This variability in distance allows the agents to move to a nearby wall without colliding with it from virtually any starting position. A continuous environment is simulated as closely as possible by allowing the agents to change direction by one-degree increments and travel distances varying by one pixel.

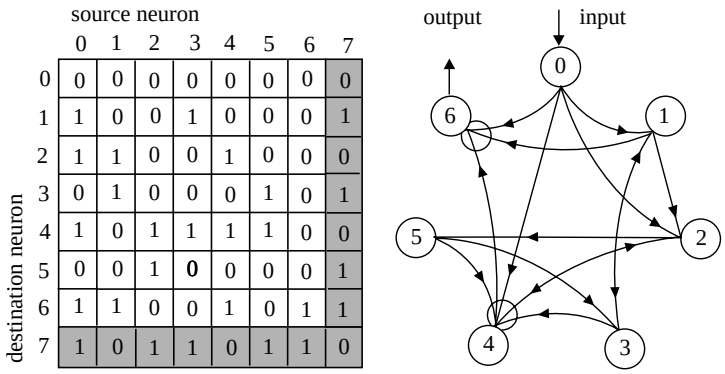

Figure 2: **Network encoding.** To the left is a small sample connectivity matrix, where grayed cells represent an unused node. To the right is depicted the corresponding network topology. Node 0 is designated as input and node 6 as output in this network.

2.2 Network representation

The neural networks are represented genetically in three parts. The first is a bit string interpreted as a two-dimensional array (figure 2). This array specifies the connectivity and overall topology of the network. The neural units are numbered and connections go from column to row. Thus if the bit in row i, column j is set, it specifies a connection from node j to node i. Using this method provides for arbitrary connectivity, specifically allowing for recurrent connections, including self-reinforcing or inhibiting connections from a unit back to itself, which is important if the task can be performed more efficiently using memory. Figure 2 shows a small example matrix and the corresponding network. The first units are designated as input units and the last as output units.

The second part of the encoding consists of the weights, which are kept as a parallel two-dimensional array. They are stored as 16-bit integers (scaled to small real numbers in the phenotype). The third part of the encoding is an integer which gives the number of nodes, which may be less than the size of the topology and weight arrays. If this is the case, part of the arrays are unused. The actual size of the networks is determined randomly when two networks are mated and is restricted to be in the closed interval specified by the sizes of the two parent networks. If the size of the child is larger than that of one of the parents, then a formerly unused part of the bit-string will become part of the encoding. This previously unused information provides for genetic diversity. There is an upper bound of 208 units in the network. This relatively low number of units so far has not proved limiting, since all of the effective networks have been considerably smaller.

The encoding outlined above is not the only one possible; others may prove to be effective. Still, it has several properties that make it well suited for the current task. Theoretically, the longer the bit string to be optimized, the more difficult the genetic optimiazation (Goldberg, 1989). Since any string which completely specifies a non-trivial neural network is inevitably quite long, it becomes important to conserve wherever possible. Using integers rather than real numbers makes the weight representation fairly compact, while the topology array obviates the need for node identifiers, which are usually required in topology-optimizing network encodings such as the marker-based method used by Fullmer and Miikkulainen (1992). Furthermore, since the bit string is only divided into two parts during mating (one part from each parent), large pieces of the parent networks remain intact in the offspring. This keeps the search process of the genetic algorithm more focused on promising genetic material. In contrast, with methods such as the marker based encoding, the

offspring may not significantly resemble the parents, the search process is much more exploratory, and was actually found to converge more slowly in a similar task.

2.3 The genetic algorithm

The generations for the genetic algorithm span 25 time steps each. In each time step, inputs are clamped for each agent, the input is propagated through the network and the agents are moved according to their outputs. Time is represented discretely, since each robot must be processed in turn. Twenty-five time steps is enough to distinguish which agents are doing well, but at the same time, limits the waste of time that would otherwise be incurred by agents that are unable to produce useful output. Such a short generation time was found empirically to speed up the optimization process .

The fitness function determines what behavior will emerge from evolution. One goal of this project was to develop a general method for deriving fitness functions, given a rough description of the desired movement. Since the idea is not for the agents to learn to follow an explicit path, it is enough to get a rough measure of whether they are in the right part of the environment.. Such a measure is implemented in terms of markers to be picked up by the agent. For this experiment, where wall-following behavior was the goal, a ring of 16 evenly spaced markers was placed just inside the perimeter of the room. The agents cannot sense these markers and must spatially cover more than half of a marker to pick it up. Once all markers are retrieved the ring of markers is replaced. The agents should avoid collisions with walls, but since the main purpose was to evolve wall-following behavior, the number of collisions was given much less weight than the number of markers retrieved. The fitness function was defined as

$$f(n) = 3m_n - c_n, \tag{1}$$

where m_n is the number of markers retrieved and c_n is the number of collisions for agent n . If collisions are costly, the weighting factors for these parameters can be adjusted.

When the time span for a generation is up, the 28 best agents are chosen to reproduce and furnish the population for the new generation. According to the usual practice, this elite group is also carried over unchanged into the next generation. The pairing of the robots is done with a nested loop which allows each agent to mate with each other agent while still favoring those with the highest rank, since the best mate first and therefore most often. Genetic operators consist of crossover and mutation. The cut-point for crossover is chosen randomly and the probability of a mutation is 0.004. Both the connectivity matrix and the weight matrix are involved in these genetic operations.

3 Performance and results

The wall-following task proved to be relatively easy to learn, given the network and task encoding described above. Initially, the agents were able to pick up a few markers just by random wandering, but the number nearly quadrupled during 2000 generations of evolution (figure 3). The number of collisions was quite low in the beginning because the agents were not yet consistently trying to get near the walls, but increased as they began to remain close to them. By the 500th generation, the number of collisions stabilized at a level that represents a tradeoff between moving faster to pick up more markers and avoiding the penalty for collisions. The greatest number of markers

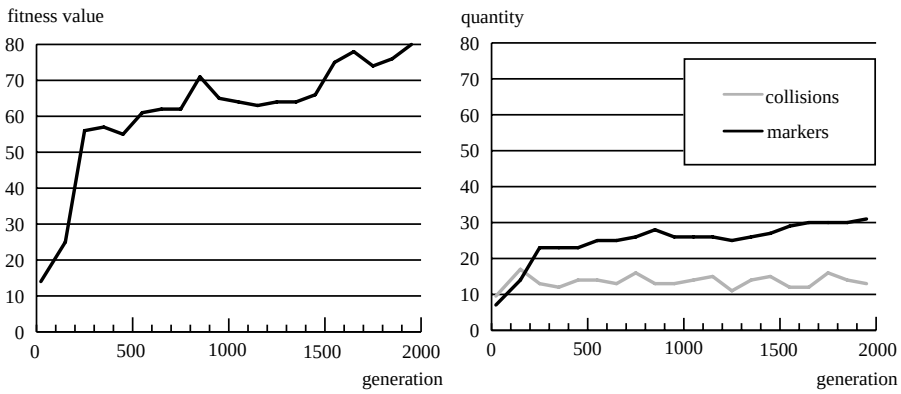

Figure 3: **Evaluation of wall-following behavior.** Each data point represents the best agent out of each 100 generations. The graph to the left shows the increase in fitness and the graph to the right shows the increase in the number of markers the agent picked up along with the number of collisions.

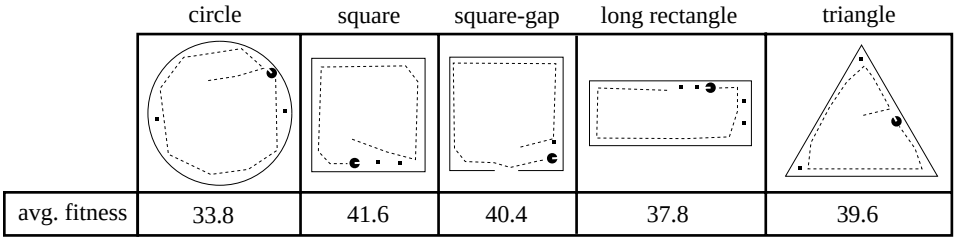

Figure 4: **Average fitness for an agent placed in a new environment.** The numbers represent an average over 10 trials, starting from random positions and random orientations. The dotted lines indicate a typical path taken in each environment.

that can picked up during any generation is 31. To do this, the agent must be initially placed on the periphery of the room in a position that allows it to move the maximum distance most of the time (i.e., it cannot start too close to a wall). It must then immediately begin a perfect clockwise or counterclockwise perambulation of the room. Given these constraints, along with the random initial placement of the robots, it is impossible to achieve it consistently. The best fitness score of 80 occurred at generation 1961, with the agent picking up all 31 markers, which is remarkably good performance.

To confirm that the agent had indeed evolved a general wall-following control strategy, it was then tested in five new environments (figure 4). These included a circle, a perfect square, a square with a 12 pixel gap in the wall, an elongated rectangle, and a triangle. In order to get a fair comparison with the original environment, all the rooms had the same perimeter as the the original one and contained the same number of markers. Statistics were kept over 10 runs of 25 time steps each.

In both squares and in the triangle, the agent did quite well, regularly picking up more than 20 markers per run, although collisions were more frequent in all the new environments. The more acute angles of the triangle were more difficult for the robot to negotiate, since it had not previously experienced a corner sharper than 90 degrees. Not surprisingly, the circle proved to be the most problematic. Since the robot cannot follow a curved trajectory, it must either move shorter distances to approximate the curve or collide more often with the walls. The run in the room with a gap in the wall confirmed the hypothesis that the agents were following the wall, since

the path showed a slight outward bump as the robot passed the gap. The elongated rectangle runs showed another effect which at least partially explains the increase in collisions. The robot tended to cut the rectangle slightly short on several runs, which suggests that not only are the robots merely following the walls, but are also learning a map of the environment and have certain expectations of how far to travel along each trajectory before having to make a turn.

4 Conclusions and future work

The success of this experiment shows that the genetic symbol grounding approach is a fruitful one. The agents successfully evolved the desired wall-following behavior and this behavior generalized to novel environments. They also showed evidence of learning a map of their environment. The next step will be to open a door between neighboring cells. This will serve two purposes. Not only does the environment become more complex, but the creatures will also have to deal with moving obstacles in the form of other agents. Once the agents have learned to wander successfully in this expanded environment, they will be required to begin in one cell, find a goal in another cell and return to their home cell. Using different colored floors (detectable by light sensors) as landmarks, the robots must learn to build rough topological maps of their environment to aid them in this task. If this can be done successfully (and current work indicates that it can), then the environment can become increasingly complex and the cognitive maps that the agents must build should become correspondingly more complex as well. Such an adaptive mapping ability will be an important contribution to the problem of robot navigation (Kuipers and Levitt, 1988).

Based on the results so far, using genetically determined neural networks as processors for robotic agents seems to be a very feasible approach to the tabula rasa learning problem. Most importantly, the agents that are a product of this system will be undeniably grounded in their simulated world, since they will have begun from ground zero, knowing nothing at all. The internal representations of the walls that delimit their world have referents and thus, inherent meaning and causal power. As a result the robots should be well equipped to respond appropriately in unexpected circumstances. This should make them extremely robust and immune to many of the problems which plague ungrounded agents when confronted with novel situations.

References

Collins, R. J. and D.R. Jefferson, 1990. Ant-Farm: Towards Simulated Evolution. UCLA-AI-90-10. AI Laboratory, UCLA.

Fullmer, B. and Miikkulainen, R. 1992. Using Marker-based Genetic Encoding of Neural Networks to Evolve Finite-state Behavior. In Varela, F.J. and P. Bourgnine (eds.), *Proceedings of the First European Conference on Artificial Life.* Cambridge, MA: Cambridge University Press/Bradford Books, pp. 285–262.

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley.

Harnad, S. 1990. The Symbol Grounding Problem. Physica D 42:1–3 pp. 335–346.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Kuipers, B. J. 1985. The Map-Learning Critter. AITR-17. AI Laboratory, University of Texas at Austin.

Kuipers, B.J. and T. Levitt, 1988. Navigation and Mapping in Large-Scale Space. *AI Magazine*, 9:2, pp. 25–43.

Lakoff, G. 1988. Smolensky, Semantics and the Sensorimotor System. *Behavioral and Brain Sciences* 11:1 pp. 39–40. Cambridge University Press.

McCarthy, J. and P.J. Hayes, 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie (eds.) *Machine Intelligence 4*. Edinburgh: Edinburgh University Press.

Patel, M.J. and U. Schnepf, 1992. Concept Formation as Emergent Phenomena. In F. J. Varela and P. Bourgnine (eds.), *Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press/Bradford Books, pp.11–20.

Pfeifer, R. and P. Verschure 1991. Distributed Adaptive Control: A Paradigm for Designing Autonomous Agents. In F. J. Varela and P. Bourgnine (eds.), *Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press/Bradford Books, pp. 21–30.

Pierce, D. and B.J. Kuipers, 1990. Learning Hill-climbing Functions as a Strategy for Generating Behaviors in a Mobile Robot. AI90-137. AI Laboratory, University of Texas at Austin.

Rivest, R.L. and R.E. Schapire, 1993 Inference of Finite Automata Using Homing Sequences. Information and Computation, 103:2, pp. 299–347.

Searle, J.R. 1980. Minds, Brains and Programs. *Behavioral and Brain Sciences* 3, pp. 417–58.