Evolution of Simple Behavior Patterns for Autonomous Robotic Agent

Roman Neruda, Stanislav Slušný, Petra Vidnerová
Institute of Computer Science, Academy of Science of the Czech Republic,
Pod vodárenskou věží 2, Prague 8, Czech Republic

Abstract

We study the emergence of intelligent behavior within a simple intelligent agent. Cognitive agent functions are realized by mechanisms based on neural networks and evolutionary algorithms. The evolutionary algorithm is responsible for the adaptation of a neural network parameters based on the performance of the embodied agent endowed by different neural network architectures. In experiments, we demonstrate the performance of evolutionary algorithm in the problem of agent learning where it is not possible to use traditional supervised learning techniques. A case study of three different trained neural networks is performed.
Keywords: Robot control, Evolutionary algorithms, Neural networks, Behavior emergence.

1 Introduction

One of the main approaches of Artificial Intelligence is to gain insight into natural intelligence through a synthetic approach, by generating and analyzing artificial intelligent behavior. In order to glean an understanding of a phenomenon as complex as natural intelligence, we need to study complex behavior in complex environments. In contrast to traditional systems, reactive and behavior based systems have placed agents with low levels of cognitive complexity into complex, noisy and uncertain environments. One of the many characteristics of intelligence is that it arises as a result of an agent's interaction with complex environments. Thus, one approach to develop autonomous intelligent agents, called *evolutionary robotics*, is through a self-organization process based on artificial evolution. Its main advantage is that it is an ideal framework for synthesizing agents whose behavior emerge from a large number of interactions among their constituent parts [9].

In the following sections we introduce multilayer perceptron networks (MLP), and radial basis function networks (RBF). Then we take a look at Khepera robots and related simulation software. In the following section we present an experiment with artificial evolution guiding the self-organization process of a neural robotic controller. We expect an emergence of behavior that guarantees efficient maze exploration. This is tested in the following section by observing I/O dependencies for three different controllers that show different behavioral patterns. In the last section we draw some conclusions and present directions for our future work.

2 Neural Networks

Neural networks are widely used in robotics for various reasons. They provide straightforward mapping from input signals to output signals, several levels of adaptation and they are robust to noise.

A *multilayer perceptron* neural network is an interconnected network of simple computing units called neurons which are ordered in layers, starting from the input layer and ending with the output layer [5]. Between these two layers there can be a number of hidden layers. Connections in this kind of networks only go forward from one layer to the next. The output $y(x)$ of a neuron is defined in equation (1):

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i\right), \tag{1}$$

where x is the neuron with n input dendrites (x_0 ... x_n), one output axon $y(x)$, w_0 ... w_n are weights and $g : \Re \to \Re$ is the activation function. We have used one of the most common activation functions, the logistic sigmoid function (2):

$$\sigma(\xi) = 1/(1 + e^{-\xi t}), \tag{2}$$

where t determines its steepness.

A *radial basis function (RBF)* neural network represents a relatively modern network architecture. In contrast with the multilayer perceptrons the RBF network contains local units, a fact motivated by the presence of many local response units in human brain. Other motivation came from numerical mathematics, where the radial basis functions were first introduced as a solution of real multivariate interpolation problems [12].

The RBF network is a feed-forward neural network with one hidden layer of RBF units and a linear output layer. By the RBF unit we mean a neuron with n real inputs $\vec{x}$ and one real output y, realizing a radial basis function φ, such as Gaussian:

$$y(\vec{x}) = \varphi\left(\frac{\parallel \vec{x} - \vec{c} \parallel}{b}\right). \tag{3}$$

The network realizes the function:

$$f_s(\vec{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\parallel \vec{x} - \vec{c}_j \parallel}{b_j}\right), \tag{4}$$

where f_s is the output of the s-th output unit.

There is a variety of algorithms for RBF network learning, in our previous work we studied their behavior and possibilities of their combinations [8].

3 Evolutionary Learning Algorithms for Robotics

The evolutionary algorithms (EA) [6, 4] represent a stochastic search technique used to find approximate solutions to optimization and search problems. They use techniques inspired by evolutionary biology such as mutation, selection, and crossover. The EA typically works with a population of *individuals* representing abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The better the solution is, the higher the fitness value it gets. The population evolves toward better solutions. The evolution starts from a population of completely random individuals and iterates in generations. In each generation, the fitness of each individual is evaluated. Individuals are stochastically selected from the current population (based on their fitness), and modified by means of *mutation* and *crossover* operators to form a new population. The new population is then used in the next iteration of the algorithm.

Feed forward neural networks used as robot controllers are encoded in order to use them the in the evolutionary algorithm. The network is represented as a floating-point encoded vector of real parameters determining the network weights. Thus, the encoded network is a vector $(P_1, \ldots, P_N)$ where N is a total number of neurons in the network, and each P_j represents an encoded neuron parameters, i.e. $P_j = (w_{j0}, \ldots, w_{jN_j})$ where N_j determines the number of j-th neuron inputs (it is generally different for different layers).

In our approach, rather typical evolutionary operators have been used, namely the uniform and arithmetic crossover and the additive mutation. As is typical for the floating-point encoded individuals of EA, all of these operations work on the floating-point number level and they do not disrupt t he number representation boundaries. The uniform crossover takes two individuals, traverses the whole vector and for each parameter it decides at random weather to exchange this position between parents or not. Arithmetic crossover combines two parents into a new individual by finding each new parameter value as a random number between the boundaries given by the parents parameter values. A mutation performs a random additive change to a parameter value with the probability of change set in advance [13]. The rate of these operators is quite big, ensuring the exploration capabilities of the evolutionary learning. A standard roulette-wheel selection is used together with a small elitist rate parameter. Detailed discussion about the fitness function is presented in the next section.

4 Experiments

4.1 The Khepera Robot

Khepera [7] is a miniature mobile robot supported by two lateral wheels that can rotate in both directions. The sensory system employs eight active infrared light sensors distributed around the body whose positions are numbered in the clockwise direction like this: 1-left, 2-front left, 3-front, 4-front, 5-front right, 6-right, 7-back, 8-back. In active mode these sensors emit a ray of infrared light and measure the amount of reflected light. The closer they are to a surface, the higher is the amount of infrared light measured. The Khepera sensors can detect a white paper at a maximum distance of approximately 5 cm.

In a typical setup, the controller mechanism of the robot is connected to the eight infrared sensors as inputs and its two outputs represent information about the left and right wheel power. For a neural network we typically consider architectures with eight input neurons, two output neurons and a single layer of five to ten hidden neurons is considered in this paper. It is difficult to train such a network by traditional supervised learning algorithms since they require instant feedback in each step, which is not the case for evolution of behavior. Here we typically can evaluate each run of a robot as a good or bad one, but it is impossible to assess each one move as good or bad. Thus, the evolutionary algorithm represent one of the few possibilities how to train the network.

Although evolution on real robots is feasible, serial evaluation of individuals on a single physical robot might require quite a long time. One of the widely

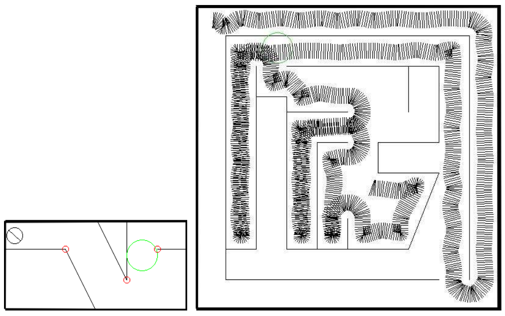
Figure 1: In the maze exploration task, agent is rewarded for passing through the zone, which can not be sensed. The zone is drawn as the bigger circle, the smaller circle represents the Khepera robot. The training environment is of 60x30 cm. For testing, the agent is put in the bigger maze of 100x100 cm. This particular agent strategy is to follow wall on it's left side.

used simulation software (for Khepera robots) is the Yaks simulator [2]. Simulation consists of predefined number of discrete steps, each single step corresponds to 100 ms. To evaluate the individual, simulation is launched several times. Individual runs are called "trials". In each trial, neural network is constructed from the chromosome, environment is initialized and the robot is put into randomly chosen starting location. The inputs of neural networks are interconnected with robot's sensors and outputs with robot's motors. The robot is then left to "live" in the simulated environment for some (fixed) time period, fully controlled by neural network. As soon as the robot hits the wall or obstacle, simulation is stopped. Depending on how well the robot is performing, the individual is evaluated by value, which we call "trial score". The higher the trial score, the more successful robot in executing the task in a particular trial. The fitness value is then obtained by summing up all trial scores.

4.2 Maze Exploration

In this experiment, the agent is put in the maze of 60x30 cm. The agent's task is to fully explore the maze. Fitness evaluation consists of four trials, individual trials differ by agent's starting location. Agent is left to live in the environment for 250 simulation steps.

The three-component $T_{k,j}$ motivates agent to learn to move and to avoid obstacles:

$$T_{k,j} = V_{k,j}(1 - \sqrt{\Delta V_{k,j}})(1 - i_{k,j}). \tag{5}$$

First component $V_{k,j}$ is computed by summing absolute values of motor speed in the k-th simulation step and j-th trial, generating value between 0 and 1. The second component $(1 - \sqrt{\Delta V_{k,j}})$ encourages the two wheels to rotate in the same direction. The last component $(1 - i_{k,j})$ encourage obstacle avoidance. The value $i_{k,j}$ of the most active sensor in k-th simulation step and j-th trial provides a conservative measure of how close the robot is to an object. The closer it is to an object, the higher the measured value in range from 0 to 1. Thus, $T_{k,j}$ is in range from 0 to 1, too.

In the j-th trial, score S_j is computed by summing normalized trial gains $T_{k,j}$ in each simulation step.

$$S_j = \sum_{k=1}^{250} \frac{T_{k,j}}{250} \tag{6}$$

To stimulate maze exploration, agent is rewarded when it passes through the zone. The zone is randomly located area, which can not be sensed by an agent. Therefore, Δ_j is 1, if agent passed through the zone in j-th trial and 0 otherwise. The fitness value is then computed as follows:

$$Fitness = \sum_{j=1}^{4} (S_j + \Delta_j) \tag{7}$$

Successful individuals, which pass through the zone in each trial, will have fitness value in range from 4 to 5. The fractional part of the fitness value reflects the speed of the agent and it's ability to avoid obstacles.

5 Results

All the networks included in the tests were able to learn the task of finding a random zone from all four positions. The resulting best fitness values are all in the range of 4.3–4.4 and they differ only in the order of few per cent. It was observed that MLP networks performed marginally better than RBF networks. More details about the relative performance of different network architectures is given in [13].

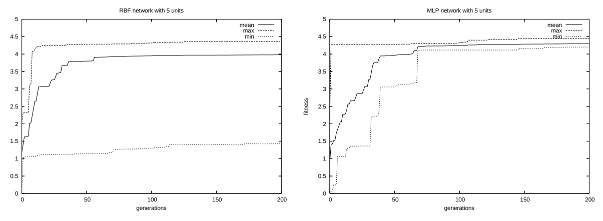

Figure 2: Plot of fitness curves in consecutive populations (maximal, minimal, and average individual) for a typical EA run (one of ten) training the RBF (MLP resp.) network with 5 units.

The important thing is to test the quality of the obtained solution is tested in a different arena, where a bigger maze is utilized (Fig. 1). Each of the architectures is capable of efficient space exploration behavior that has emerged during the learning to find random zone positions. The above mentioned figure shows that the robot trained in a quite simple arena and endowed by relatively small network of 5–10 units is capable to navigate in a very complex environment.

Several behavioral patterns have been observed for the successful controllers. The most successful individuals exhibited a wall-following behavior which is under circumstances a very efficient way to explore a general maze. Depending on the initial position and orientation, either left wall or right wall following controllers have evolved. In order to gain insight into the function of a controller, we have studied the partial I/O mappings from individual sensors to the motor control. We have chosen a typical left wall follower, a right wall follower, and an agent that exhibited general obstacle avoidance behavior without the maze exploration strategy. The plots of partial I/O mappings for these three agents are presented at Fig. 3 (left wall following behavior), Fig. 4 (right wall following behavior), and Fig. 5 (obstacle avoidance behavior).

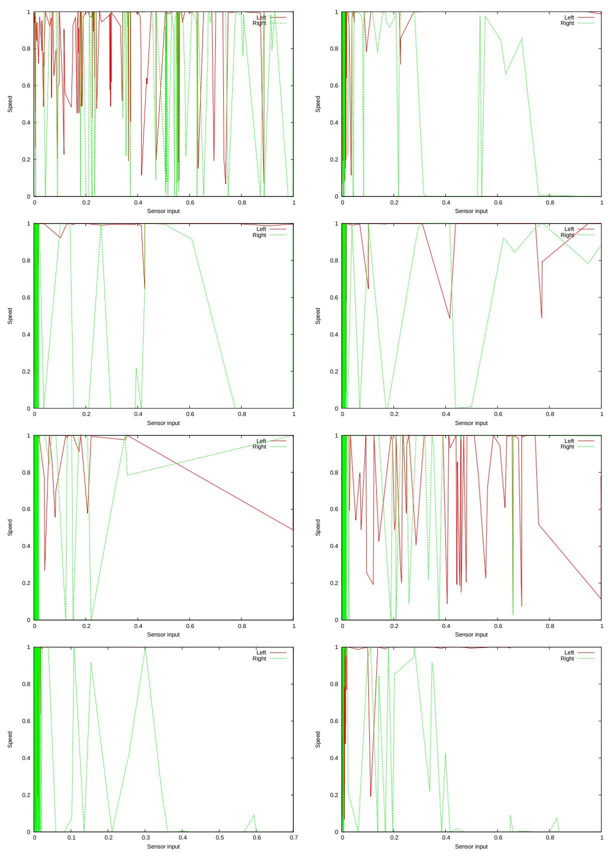

Figure 3: Plot of eight sensor functions for the left wall following agent.

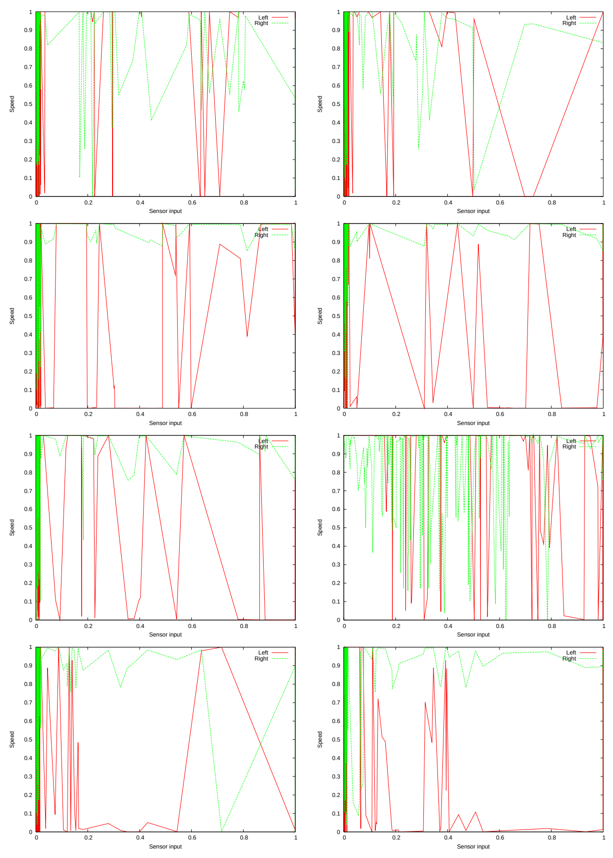

Figure 4: Plot of eight sensor functions for the right wall following agent.

From the Fig. 5 (mainly sensors 2 and 5) one can see a symmetric behavior of the obstacle avoidance agent. When the obstacle is in front-left direction (sensor 2 is active), the agent has a tendency to power left engine more, i.e. it turns right from the obstacle. Inversely, the responses to sensor 5 mean powering the right wheel, i.e. turning left. Similar form of the I/O mapping for sensors 1 and 6 support this behavior. When the obstacle is in front (sensors 3 and 4), the agent chooses to turn right.

For the wall following agents we are interested to observe the symmetry between Fig. 3 and Fig. 4. First, let us note that for the left wall following agent, the important sensors should be sensors 1–4, while for the right wall following one, it should be sensors 3–6. If the maze has wide enough corridors, the sensors 5–6 (or 1–2, respectively) should not get many inputs at all. The back sensors 7–8 should reflect the situation that when they register a wall, it means that the left wall follower is turning right and the right wall follower is turning left. One can see that at the last row of Fig. 3 the left engine runs faster, i.e. the left follower is turning right, while the same row of Fig. 4 shows the opposite situation of an agent turning left. The many peaks of the function of the sensor closest to the wall (i.e. sensor 1 on Fig. 3 and sensor 6 on Fig. 4) demonstrate the typical duck-like swinging ride of both agents while trying to keep the optimal

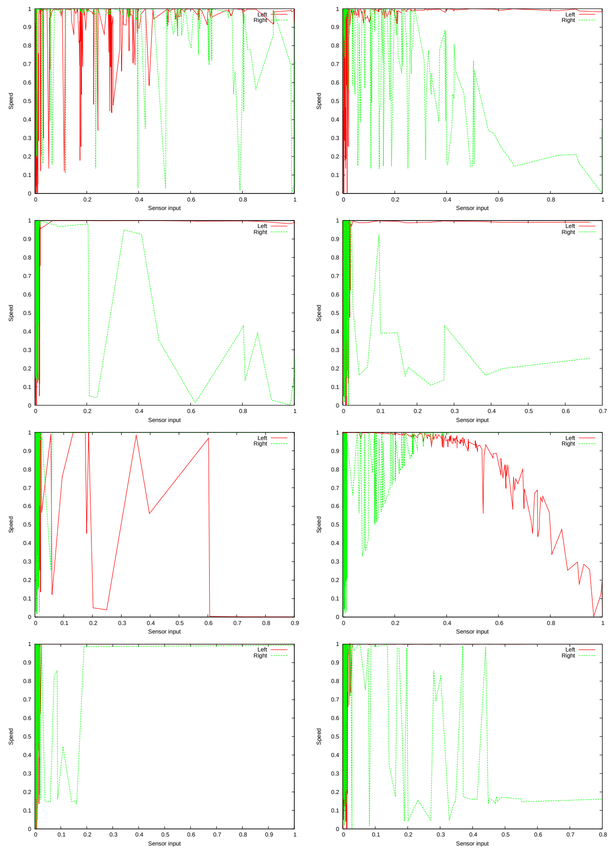

Figure 5: Plot of eight sensor functions for the obstacle avoiding agent.

distance from the wall by small fluctuations. Finally, the comparison of sensors 2 and 3 at Fig. 3 with sensors 4 and 5 at Fig. 4 shows that in case of an obstacle in front-left (front-right, resp.) each agent turns in the opposite direction, i.e. right (or left, resp.), thus following its strategy.

6 Conclusions

The main goal of this paper was to demonstrate the ability of neural networks trained by evolutionary algorithm to achieve non-trivial tasks in controlling the robotic agent with the emphasis on the evolved behavior patterns. The results are quite encouraging, rather small MLP or RBF neural networks are able to develop the exploration behavior. The trained network is able to control the robot in the previously unseen environment. Typical behavioral patterns, like following the right or left wall have been developed, which in turn resulted in the very efficient exploration of an unknown maze. The best results achieved by any of the network architectures are quite comparable, with simpler perceptron networks (such as the 5-hidden unit perceptron) marginally outperforming RBF networks.

The analysis of the evolved controller shows that there can be observed a small set of rather simple and rational rules behind the "black-box" of the neural network. Let us note that the wall following controllers have been MLPs while the obstacle avoiding one was an RBF network. For the future work we would like to make use of the local nature of the RBF networks and its similarity to fuzzy logic controllers in order to automatically obtain a set of fuzzy rules directly from a trained RBF network.

The results reported above represent just a few steps in the journey toward more autonomous and adaptive robotic agents. The robots are able to learn simple behavior by evolutionary algorithm only by rewarding the good ones, and without explicitly specifying particular actions. The next step is to extend this approach for more complicated actions and compound behaviors. This can be probably realized by incremental learning one network a sequence of several tasks. Another—maybe a more promising approach—is to try to build a higher level architecture (like a type of the Brooks subsumption architecture [1]) which would have a control over switching simpler tasks realized by specialized networks. Ideally, this higher control structure is also evolved adaptively without the need to explicitly hardwire it in advance. The last direction of our future work is the extension of this methodology to the field of collective behavior.

Acknowledgments

This work was supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567). The experiments have been performed partly in the EPCC under the project HPC-EUROPA (RII3-CT-2003-506079) with the support of the European Community - Research Infrastructure Action under the FP6 Structuring the European Research Area Programme.

References

[1] Brooks, R. A.: A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* (1986), **RA-2**, 14-23.

[2] J. Carlsson, T. Ziemke, YAKS - yet another Khepera simulator, in: S. Ruckert, Witkowski (Eds.) *Autonomous Minirobots for Research and Entertainment Proceedings of the Fifth International Heinz Nixdorf Symposium*, HNI-Verlagsschriftenreihe, Paderborn, Germany, 2001.

[3] J. Elman, Finding structure in time, *Cognitive Science 14* (1990) 179-214.

[4] D. B. Fogel. *Evolutionary Computation: The Fossil Record.* MIT-IEEE Press. 1998.

[5] S. Haykin. *Neural Networks: A Comprehensive Foundation* (2nd ed). Prentice Hall, 1998.

[6] J. Holland. *Adaptation In Natural and Artificial Systems* (reprint ed). MIT Press. 1992.

[7] Khepera II web page documentation. http://www.k-team.com

[8] R. Neruda, P. Kudová: Learning methods for RBF neural networks. *Future Generations of Computer Systems*, **21**, (2005), 1131–1142.

[9] S. Nolfi, D. Floreano. *Evolutionary Robotics — The Biology, Intelligence and Technology of Self-Organizing Machines.* The MIT Press, 2000.

[10] S. Nolfi. Adaptation as a more powerful tool than decomposition and integration. In T.Fogarty and G.Venturini (Eds.), *Proceedings of the Workshop on Evolutionary Computing and Machine Learning*, 13th International Conference on Machine Learning, Bari, 1996.

[11] S. Nolfi. The power and limits of reactive agents. Technical Report. Rome, Institute of Psychology, National Research Council, 1999.

[12] M. Powel: Radial basis functions for multivariable interpolation: A review. In: *IMA Conference on Algorithms for the Approximation of Functions and Data*, RMCS, Shrivenham, England (1985) 143–167

[13] S. Slušný, R. Neruda: Autonomous robot control by means of three evolutionary neural architectures. Proceedings of the 1st European AROB Workshop, Wien (2007)