Neural Networks for Evolutionary Robotics

Slušný, Stanislav
2007

Dostupný z http://www.nusl.cz/ntk/nusl-37425

Neural Networks for Evolutionary Robotics

Post-Graduate Student:
MGR. STANISLAV SLUŠNÝ
Institute of Computer Science of the ASCR, v. v. i.
Pod Vodárenskou věží 2
182 07 Prague 8, CZ
slusnys@gmail.com

Supervisor:
MGR.. ROMAN NERUDA, CSC.
Institute of Computer Science of the ASCR, v. v. i.
Pod Vodárenskou věží 2
182 07 Prague 8, CZ
roman@cs.cas.cz

Field of Study:
Software Systems

The work was fully supported by Czech Science Foundation (GACR), project No: 201/05/H014.

Abstract

The design of intelligent embodied agents represents one of the key research topics of today's artificial intelligence. The goal of this work is to study 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 in a simulated environment. The evolutionary learning is realized for several architectures of neural networks, namely the feed-forward multilayer perceptron network, the recurrent Elmans neural network, and the radial basis function network. 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.

1. Introduction

One of the main goals 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), Elmans networks (ELM) and radial basis function networks (RBF). Then we take a look at Khepera robots and related simulation software. In the following section we present two experiments with Khepera robots. In both of them, the artificial evolution is guiding the self-organization process. In the first experiment we expect an emergence of behavior that guarantees full maze exploration. The second experiment shows the ability to train the robot to discriminate between walls and cylinders. In the last section we draw some conclusions and present directions for our future work.

2. Neural Networks

2.1. Multilayer Perceptron Networks

A multilayer feedforward 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 $g : \Re \rightarrow \Re$ 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.

In our approach, the evolutionary algorithm is responsible for weights modification, the architecture of the network is determined in advance and does not undergo the evolutionary process.

2.2. Recurrent Neural Networks

In recurrent neural networks, besides the feedforward connections, there are additional recurrent connections that go in the opposite direction. These networks are often used for time series processing because the recurrent connection can work as a memory for previous time steps. In the Elman [3] architecture, the recurrent connections explicitly hold a copy (memory) of the hidden units activations at the previous time step. Since hidden units encode their own previous states, this network can detect and reproduce long sequences in time. The scheme how the Elman network works is like this (also cf. Fig. 1):

- Compute hidden unit activations using net input from input units and from the copy layer.
- Compute output unit activations as usual based on the hidden layer.
- Copy new hidden unit activations to the copy layer.

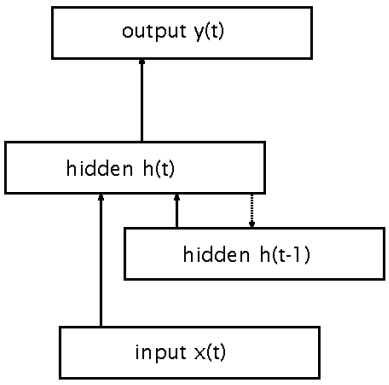

Figure 1: Scheme of layers in the Elman network architecture.

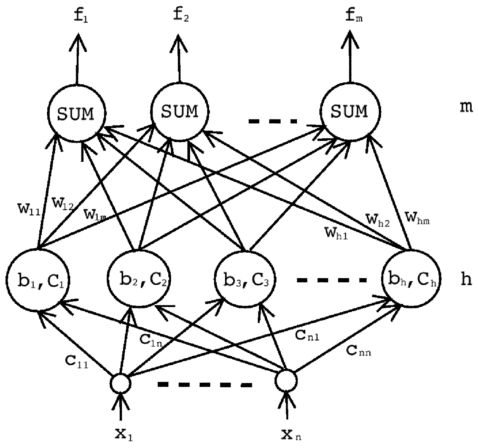

Figure 2: Scheme of layers in the RBF network architecture.

2.3. Radial Basis Function Networks

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

It is a feed-forward neural network with one hidden layer of RBF units and a linear output layer (see Fig. 2). By an 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.

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

The network realizes the function:

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

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].

The learning algorithm that we use for RBF networks was motivated by the commonly used Three-step learning. Parameters of RBF network are divided into three groups: centers, widths of the hidden units, and output weights. Each group is then trained separately. The centers of hidden units are found by clustering (k-means algorithm) and the widths are fixed so as the areas of importance belonging to individual units cover the whole

input space. Finally, the output weights are found by EA. The advantage of such approach is the lower number of parameters to be optimized by EA , i.e. smaller length of individual.

3. Evolutionary Learning Algorithms for Robotics

3.1. The Khepera Robot

Khepera [7] is a miniature mobile robot with a diameter of 70 mm and a weight of 80 g. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and in the back. The sensory system employs eight "active infrared light" sensors distributed around the body, six on one side and two on other side. 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 input 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 neurons, mostly five or 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.

3.2. Evolutionary Algorithm

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 towards 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 operators *mutation* and *crossover* to form a new population. The new population is then used in the next iteration of the algorithm.

3.3. Evolutionary Network Learning

Various architectures of neural networks used as robot controllers are encoded in order to use them the evolutionary algorithm. The encoded vector is represented as a floating-point encoded vector of real parameters determining the network weights.

Typical evolutionary operators for this case have been used, namely the uniform crossover and the mutation which performs a slight additive change in the parameter value. 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 discussions about the fitness function are presented in the next section.

4. Experiments

4.1. Setup

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 used simulation software (for Khepera robots) is the Yaks simulator [2], which is freely available. 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

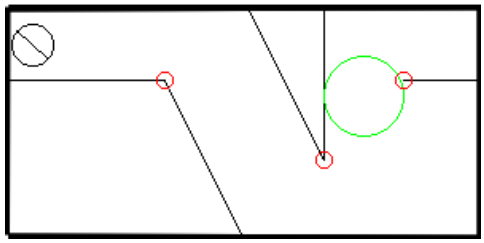

Figure 3: The environment in the maze exploration task. The zone is drawn as the bigger circle, the smaller circle represents the Khepera robot.

In this experiment, the agent is put in the maze of 60x30 cm(Fig. 3). 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:

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

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.

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

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:

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

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.

4.3. 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 (cf. Tab. 1) are all in the range of 4.3–4.4 and they differ only in the order of few per cent. It can be seen that the MLP networks perform slightly better, RBF networks are in the middle, while recurrent networks are a bit worse in terms of the best fitness achieved. According to their general performance, which takes into account ten different EA runs, the situation changes slightly. In general, the networks can be divided into two categories. The first one represents networks that performed well in each experiment in a consistent manner, i.e. every run of the evolutionary algorithm out of the ten random populations ended in finding a successful network that was able to find the zone from each trial. MLP networks and recurrent networks with 5 units fall into this group. The second group has in fact a smaller trial rate because, typically, one out of ten runs of EA did not produced the optimal solution. The observance of average and standard deviation values in Tab. 1 shows this clearly. This might still be caused by the less-efficient EA performance for RBF and Elman networks.

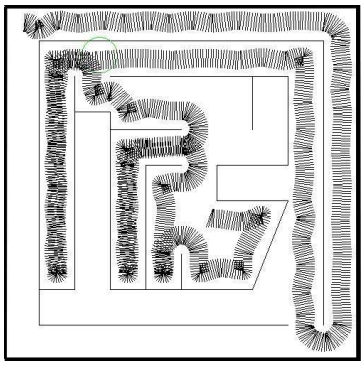

Figure 4: Testing environment in the maze exploration task is the bigger maze of 100x100 cm. Agent's strategy is to follow wall on it's left side.

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. 4). 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.

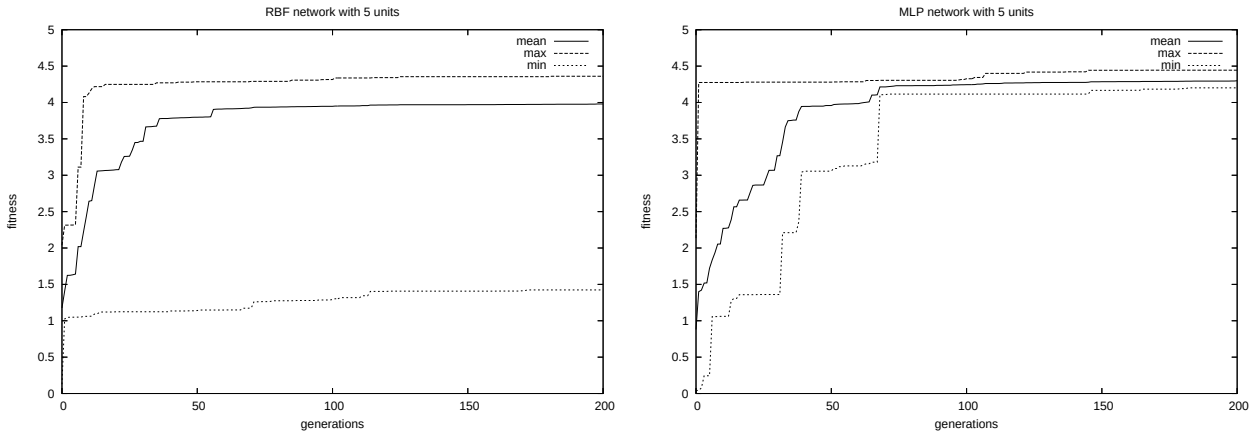

Figure 5: Plots of fitness curves in consecutive populations (maximal, minimal, and average individual) for a typical EA run (one of ten) training the RBF (and MLP, respectively) network with 5 units.

Network type	**Maze exploration**				**Wall and cylinder**			
	mean	std	min	max	mean	std	min	max
MLP 5 units	4.29	0.08	4.20	4.44	2326.1	57.8	2185.5	2390.0
MLP 10 units	4.32	0.07	4.24	4.46	2331.4	86.6	2089.0	2391.5
ELM 5 units	4.24	0.06	4.14	4.33	2250.8	147.7	1954.5	2382.5
ELM 10 units	3.97	0.70	2.24	4.34	2027.8	204.3	1609.5	2301.5
RBF 5 units	3.98	0.90	1.42	4.36	1986.6	230.2	1604.0	2343.0
RBF 10 units	4.00	0.97	1.23	4.38	2079.4	94.5	2077.5	2359.5

Table 1: Comparison of the fitness values achieved by different types of network in the experiments.

4.4. Walls and Cylinders Experiment

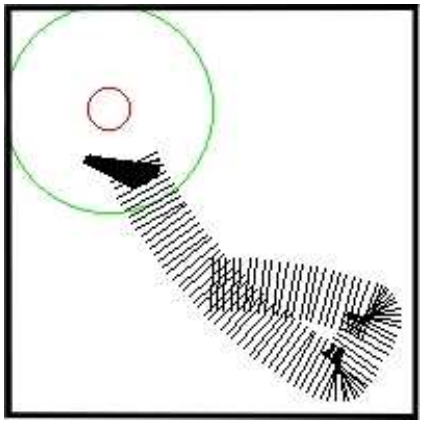

Figure 6: Trajectory of an agent doing the Walls and cylinders task.

Following experiment is based on the experiment carried out by Nolfi [10, 11]. The task is to discriminate between the sensory patterns produced by the walls and small cylinders. As noted in [9], passive networks (i.e. networks which are passively exposed to a set of sensory patterns without being able to interact with the external environment through motor action), are mostly unable to discriminate between different objects. However, this problem can easily be solved by agents that are left free to move in the environment.

The agent is allowed to sense the world by only six frontal infrared sensors, which provide it with only limited information about environment. Fitness evaluation consists of five trials, individual trials differ by agent's starting location. Agent is left to live in the environment for 500 simulation steps. In each simulation step, trial score is increased by 1, if robot is near the cylinder, or 0.5, if robot is near the wall. The fitness value is then obtained by summing up all trial scores. Environment is the arena of 40x40 cm surrounded by walls.

4.5. Results

It may seem surprising that even this more complicated task was solved quite easily by relatively simple network architectures(Fig. 6). The images of walls and cylinders are overlapping a lot in the input space determined by the sensors.

The results in terms of best individuals are again quite

comparable for different architectures with reasonable network sizes. The differences are more pronounced than in the case of the previous task though. Again, the MLP is the overall winner mainly when considering the overall performance averaged over ten runs of EA. The behavior of EA for Elman and RBF networks was less consistent, there were again several runs that obviously got stuck in local extrema (cf. Tab. 1).

We should emphasize the difference between fitness functions in both experiment. The fitness function used in the first experiment rewards robot for single actions, whereas in the second experiment, we describe only desired behavior.

All network architectures produced similar behavior. Robot was exploring the environment by doing arc movements and after discovering target, it started to move there and back and remained in it's vicinity.

5. Conclusions

The main goal of this paper was to demonstrate the ability of neural networks trained by evolutionary algorithm to achieve non-trivial robotic tasks. There have been two experiments carried out with three types of neural networks and different number of units.

For the maze exploration experiment the results are encouraging, a neural network of any of the three types is 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 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 Elman and RBF networks.

In the second experiment it has been demonstrated that the above mentioned approach is able to take advantage of the embodied nature of agents in order to tell walls from cylindrical targets. Due to the sensor limitations of the agent, this task requires a synchronized use of a suitable position change and simple pattern recognition. This problem is obviously more difficult than the maze exploration, nevertheless, most of the neural architectures were able to locate and identify the round target regardless of its position.

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 a 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.

References

[1] R. A. Brooks, 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<96>-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