Rule-based Analysis of Behaviour Learned by Evolutionary and Reinforcement Algorithms

Stanislav Slušný, Roman Neruda, and Petra Vidnerová

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, Prague 8, Czech Republic
{slusny,roman,petra}@cs.cas.cz

Abstract. We study behavioural patterns learned by a robotic agent by means of two different control and adaptive approaches — a radial basis function neural network trained by evolutionary algorithm, and a traditional reinforcement Q-learning algorithm. In both cases, a set of rules controlling the agent is derived from the learned controllers, and these sets are compared. It is shown that both procedures lead to reasonable and compact, albeit rather different, rule sets.

1 Introduction

We study intelligent behaviours that arise as a result of an agent's interaction with its environment. The ultimate goal of the process is to develop an embodied and autonomous agent with a high degree of adaptive possibilities [9]. Two main approaches to tackle this problem are currently the traditional reinforcement learning (RL) [13] and evolutionary robotics (ER) [8]. Both these approaches fall into the same category of learning algorithms that are often used for tasks where it is not possible to employ more specific supervised learning techniques. Designing an agent control mechanism is a typical example of such a problem where an instant reward of agent actions is not available. We are usually able to judge positive or negative behaviour patterns of an agent (such as finding a particular spot in a maze or hitting a wall) and evaluate it on the coarser time scale. This information is used by different learning algorithms of reinforcement type to strengthen successful partial behaviour patterns, and in the course of adaptation process, to develop an agent solving a given task.

The Q-learning approach considers discrete spaces of possible agent states and actions, and in the course of adaptation creates approximations of the optimal strategy — a way to select a particular action in a given state of an agent such that the potential (delayed) reward from the environment is maximized.

The ER approach attacks the problem through a self-organization process based on artificial evolution [5]. Control mechanisms of an agent are typically based on a neural network which provides direct mapping from agent sensors to effectors. Most of the current applications use traditional multi-layer perceptron networks [4]. In our approach we utilize local unit network architecture called radial basis function (RBF) network which has competitive performance, more learning options, and (due to its local nature) better interpretation possibilities [11, 12].

2 Reinforcement Learning

Let us consider an embodied agent that is interacting with the environment by its sensors and effectors. The essential assumption of RL is that the agent has to be able to sense rewards coming from the environment. Rewards evaluate taken actions, agent's task is to maximize them. There has been several algorithms suggested so far. We have used the Q-learning algorithm, which was first breakthrough of RL [14].

The next important assumption is that agent is working in discrete time steps. Symbol S will denote finite discrete set of states and symbol A set of actions. In each time step t, agent determines its actual state and chooses one action. Therefore, agent's life can be written as a sequence $o_0 a_0 r_0 o_1 a_1 r_1 \ldots$ where o_t denotes observation through the sensors, $a_t \in A$ action and finally symbol $r_t \in R$ represents *reward*, that was received at time t. The most serious assumption of RL algorithms is the *Markov property*, which states, that agent does not need history of previous observations to make decision. The decision of the agent is based on the last observation o_t only. When this property holds, we can use theory coming from the field of *Markov decision processes* (MDP). The direct implication of Markov property is the equality of states and observations. The strategy π, which determines what action is chosen in particular state, can be defined as function $\pi : S \to A$, where $\pi(s_t) = a_t$.

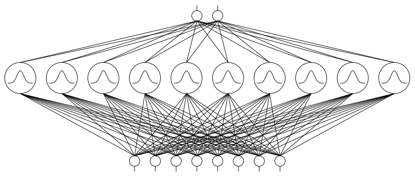
Now, the task of the agent is to find optimal strategy π^*. Optimal strategy is the one, that maximalizes expected reward. In MDP, single optimal deterministic strategy always exists, no matter in what state has the agent started. The quantity $V^\pi(s_t)$ is called discounted cumulative reward. It is telling us, what reward can be expected, if the agent starts in state s_t and follows policy π: $V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0} \gamma^i r_{t+1}$.

Here $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards. Optimal strategy π^* can now be defined as: $\pi^* = \text{argmax}_\pi \{V^\pi(s), \forall s \in S\}$. To simplify the notation, let us write $V^*(s)$ instead of symbol V^{π^*}, value function corresponding to optimal strategy π^*: $V^*(s) = \max_\pi V^\pi(s)$.

```
1. Let $S$ be the finite set of states and $A$ finite set of actions.
   $\forall s \in S, a \in A : Q(s,a) = 0$
2. Process sensors and obtain state $s$
3. Repeat:
   – Choose and carry out action $a$
   – Receive reward $r$
   – Obtain new state $s'$
   – $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$
   – $s \leftarrow s'$
```

Algorithm 1: Q-learning.

The Q-learning algorithm was the first algorithm to compute optimal strategy π^* [14]. The key idea of the algorithm is to define the so-called *Q-values*. $Q^\pi(s,a)$ is



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

Fig. 1. A scheme of a Radial Basis Function Network, f_s is the output of the s-th output unit. φ is an activation function, typically Gaussian function $\varphi(s) = e^{-s^2}$.

the expected reward, if the agent takes action a in state s and then follows policy π: $Q^\pi(s,a) = r(s,a) + \gamma V^\pi(s')$, where s' is the state, in which agent occurs taking action a in state s ($s' = \delta(s,a)$). Q-learning algorithm (Algorithm 1) guarantees convergence to optimal values of $Q^*(s,a)$, if Q-values are represented without any function approximations (in table), rewards are bounded and every state-action pair is visited infinitely often. To fulfil the last condition, every action has to be chosen with non-zero probability. Probability $P(a|s)$ of choosing action a in state s is defined as [6]: $P(a_i|s) = k^{Q(s,a_i)} / \sum_j k^{Q(s,a_j)}$, where constant $k > 0$ determines exploitation-exploration rate. Big values of k will make agent to choose actions with above average values. On the other hand, small values will make agent to choose actions randomly. Usually, learning process is started with small k, that is slightly increasing during the course of learning. Optimal values V^*s can be obtained from $Q^*(s,a)$ by the equality: $V^*(s) = \max_{a'} Q(s,a')$.

3 Evolutionary Learning of RBF Networks

Evolutionary robotics combines two AI approaches: neural networks and evolutionary algorithms. The control system of the robot is realized by a neural network, in our case an RBF network. 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 behaviour. 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.

The *RBF network* [10, 7, 1], used in this work, is a feed-forward neural network with one hidden layer of *RBF units* and linear output layer. The network function is given in Eq. (1) (see Fig. 1). The evolutionary algorithms (EA) [5, 3] represent a stochastic search technique used to find approximate solutions to optimization and search problems. They work with a population of *individuals* representing feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. 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 genetic operators to form a new generation.

In case of RBF networks learning, each individual encodes one RBF network. The individual consists of h blocks: $I_{RBF} = \{B_1, \ldots, B_h\}$, where h is a number of hidden units. Each of the blocks contains parameter values of one RBF units, $B_k = \{c_{k1}, \ldots, c_{kn}, b_k, w_{k1}, \ldots, w_{km}\}$, where n is the number of inputs, m is the number of outputs, $\boldsymbol{c_k} = \{c_{k1}, \ldots, c_{kn}\}$ is the k-th unit's centre, b_k the width and $\boldsymbol{w}_k = \{w_{k1}, \ldots, w_{km}\}$ the weights connecting k-th hidden unit with the output layer. The parameter values are encoded using direct floating-point encoding. Concerning the genetic operators, the standard *tournament selection*, *1-point crossover* and *additive mutation*[1] are used. The fitness function should reflect how good the robot is in given tasks and so it is always problem dependent. Detailed description of the fitness function is included in the experiment section.

4 Experimental Framework

In order to compare performance and properties of described algorithms, we conducted simulated experiment. Miniature robot of e-puck type [2] was trained to explore the environment and avoid walls. E-puck is a mobile robot supported by two lateral wheels that can rotate in both directions and two rigid pivots. The sensory system employs eight active IR sensors distributed around the body. Sensors return values from interval $[0, 4095]$. Effectors accept values from interval $[-1000, 1000]$. The higher the absolute value, the faster is the motor moving in either direction.

Table 1. Sensor values and their meaning.

Sensor value	Meaning	Sensor value	Meaning
0-50	NOWHERE	1001-2000	NEAR
51-300	FEEL	2001-3000	VERYNEAR
301-500	VERYFAR	3001-4095	CRASHED
501-1000	FAR		

Instead of 4095 raw sensor values, learning algorithms worked with 5 preprocessed perceptions (see Tab. 1). Effector's values were processed in similar way: instead of 2000 values, learning algorithm was allowed to choose from values [-500, -100, 200, 300, 500]. To reduce the state space even more, we grouped pairs of sensors together and back sensors were not used at all. Agent was trained in the simulated environment of size 100 x 60 cm and tested in more complex environment of size 110 x 100 cm. We used Webots [15] simulation software.

In the first experiment, we have used Q-learning algorithm as described in Section 2. Each state was represented by a triple of perceptions. For example, the state [NEAR, NOWHERE, NOWHERE] means, that the robot sees a wall on its left side only. Action was represented by a pair [left speed, right speed].

Learning process was divided into episodes. Each episode took at most 1000 simulation steps. At the end of each episode, agent was moved to one from 5 randomly chosen positions. Episode could be finished earlier, if agent hit the wall. The learning process was stopped after 10000 episodes. Parameter γ was set to 0.3.

[1] Additive mutation changes the values by adding small value randomly drawn from $\langle -\epsilon, \epsilon \rangle$.

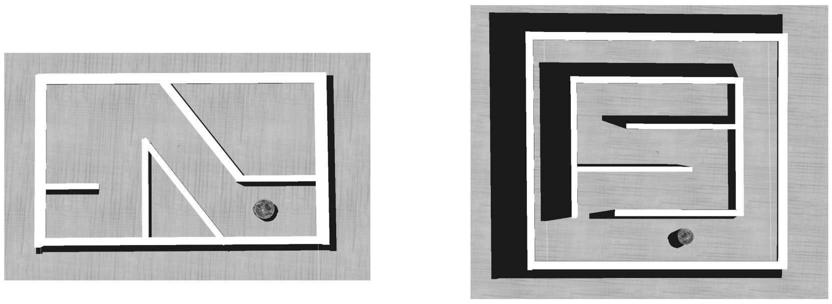


Fig. 2. Simulated environments for agent training and testing: a) Agent was trained in the simulated environment of size 100 x 60 cm. b) Simulated testing environment of size 110 x 100 cm.

In the second experiment the evolutionary RBF networks were applied to the same maze-exploration task (see Fig. 2). The network has 3 input units, 5 hidden Gaussian units, and 2 output units. The three inputs correspond to the coupled sensor values (two left sensors, two front sensors, two right sensors), which are preprocessed in the way described in Tab. 1. The two outputs correspond to the left and right wheel speeds and before applying to robot wheels they are rounded to one of 5 valid values.

Fitness evaluation consists of two trials, which differ by agent's starting location (the two starting positions are in the opposite ends of the maze). Agent is left to live in the environment for 800 simulation steps. In each step, a three-component score is calculated to motivate agent to learn to move and to avoid obstacles:

$$T_{k,j} = V_{k,j}(1 - \sqrt{\Delta V_{k,j}})(1 - i_{k,j}). \tag{2}$$

The first component $V_{k,j}$ is computed by summing absolute values of motor speed (scaled to $\langle -1, 1 \rangle$) 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 (scaled to $\langle 0, 1 \rangle$) 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 is the measured value in range from 0.0 to 1.0. Thus, $T_{k,j}$ is in range from 0.0 to 1.0, too. In the j-th trial, score S_j is computed by summing normalized trial gains $T_{k,j}$ in each simulation step $S_j = \frac{1}{800} \sum_{k=1}^{800} T_{k,j}$. To stimulate maze exploration, agent is rewarded, when it passes through one of predefined zones. There are three zones located in the maze. They can not be sensed by an agent. The reward $\Delta_j \in \{0, 1, 2, 3\}$ is given by the number of zones visited in the j-th trial. The fitness value is then computed as $F = \sum_{j=1}^{K} \Delta_j + \sum_{j=1}^{K} \frac{S_j}{K}$, where $K = 2$ is the number of trials.

5 Experimental Results

Table 2 contains states with biggest and smallest Q-values and their best action. The states with biggest Q-values contain mostly perception NOWHERE. On the other side, states with smallest Q-values contain perception CRASHED.

Learned behaviour corresponds to obstacle avoidance behaviour. The most interested are the states, which contain perception ”NEAR”. Expected rules ”when obstacle left, then turn right” can be found. States without perception ”NEAR” were evaluated as safe — even if bad action was chosen in this state, it could be fixed by choosing good action in next state. Therefore, these actions do not tell us a lot about agent’s behaviour. On the other side, action with perception VERYNEAR leaded to the crash, usually. Agent was not able to avoid the collision.

Table 2. 5 states with biggest and smallest Q-values and their best actions

	State		**Action**	**Q-value**
left	**front**	**right**		
NOWHERE	NOWHERE	VERYFAR	[500, 300]	5775.71729
NOWHERE	NOWHERE	NOWHERE	[300, 300]	5768.35059
VERYFAR	NOWHERE	NOWHERE	[300, 500]	5759.31055
NOWHERE	NOWHERE	FEEL	[300, 300]	5753.71240
NOWHERE	VERYFAR	NOWHERE	[500, 100]	5718.16797
CRASHED	CRASHED	CRASHED	[300, 500]	-40055.38281
CRASHED	NOWHERE	CRASHED	[300, 300]	-40107.77734
NOWHERE	CRASHED	VERYNEAR	[300, 500]	-40128.28906
FAR	VERYNEAR	CRASHED	[300, 500]	-40210.53125
NOWHERE	CRASHED	NEAR	[200, 500]	-40376.87891

The experiment with evolutionary RBF network was repeated 10 times, each run lasted 200 generations. In all cases the successful behaviour was found, i.e. the evolved robot was able to explore the whole maze without crashing to the walls. Table 3 shows parameters of an evolved network with five RBF units. We can understand them as rules providing mapping from input sensor space to motor control. However, these ‘rules’ act in accord, since the whole network computes linear sum of the corresponding five Gaussians.

The following Tab. 4 shows rules from actual run of the robot in the train arena. The nine most frequently used rules are listed. It can be seen that this agent represents a typical evolved left-hand wall follower. Straight movement is a result of situations when there is a wall far left, or both far left and right. If the robot sees nothing, it rotates leftwise (rule 2). The front collision is avoided by turning right, as well as a near proximity to the left wall (rules 6–8).

The evolved robot was then tested in the bigger testing maze. It behaved in a consistent manner, using same rules, demonstrating generalization of the behaviour trained in the former maze.

Table 3. Rules represented by RBF units (listed values are original RBF network parameters after discretization).

Sensor			Width	Motor	
left	**front**	**right**		**left**	**right**
VERYNEAR	NEAR	VERYFAR	1.56	500	-100
FEEL	NOWHERE	NOWHERE	1.93	-500	500
NEAR	NEAR	NOWHERE	0.75	500	-500
FEEL	NOWHERE	NEAR	0.29	500	-500
VERYFAR	NOWHERE	NOWHERE	0.16	500	500

Table 4. Most important rules represented by trained RBF network and their semantics.

Sensor			Motor		
left	**front**	**right**	**left**	**right**	
FEEL	NOWHERE	NOWHERE	500	500	straight forward
NOWHERE	NOWHERE	NOWHERE	100	500	turning left
VERYFAR	NOWHERE	NOWHERE	500	500	straight forward
FEEL	NOWHERE	FEEL	500	500	straight forward
NOWHERE	NOWHERE	FEEL	100	500	turning left
FAR	NOWHERE	NOWHERE	500	300	turning right
FEEL	FEEL	NOWHERE	500	300	turning right
NEAR	NOWHERE	NOWHERE	500	100	turning right

Both approaches where successful in finding the strategy for maze exploration. The 200 generations of evolutionary learning needed on average 8822 fitness evaluations (corresponds approx. 14 115 RL epochs). To acquire the desired behaviour, RBF networks needed from 529 to 2337 fitness evaluations.

6 Conclusion

We have presented experiments with RL and ER algorithms training a robot to explore a maze. It is known from the literature, and from our previous works, that this problem is manageable by both RL and ER learning with different neural representations of the control mechanism. Usually, in such a successful learning episode, an agent with general behavioural pattern emerges that is able to explore previously unseen maze in an efficient way.

In this work we have focused on comparison of rules derived by traditional RL approach and by the evolved neural networks. We have chosen the RBF network architecture with local processing units. These networks are easily to interpret in terms of rules than traditional perceptron networks. A simple analysis shows that both RL and ER resulted in a rules that are reasonable, and easy to interpret as higher-level behaviour traits. The RL approach shows rational obstacle avoidance, while the neuro-evolution approach comes with more compact individuals that can be clearly classified as left-hand wall followers (or right-hand wall followers, respectively).

As we have seen, different learning approaches can lead to different behaviours. Agents trained by evolutionary algorithms usually show simple behaviours. Often, changing basic environment constraints (dimensions of environment, for example) can make learned strategy fail [8]. In our experiment, learned strategy is simple (it can be described by several rules) but effective. Agent learned by Q-learning algorithm showed more complex behaviour. It can cope with situations, in which agent trained by ER would fail. However, effective wall following strategy was not discovered.

In our further work, we would like to take advantages of both approaches. The basic learning mechanism will be evolutionary algorithm. Behavioural diversity could be maintained by managing population of agents that use different learning approaches (RBF networks and reinforcement learning). In both algorithms used, experience can be expressed as a set of rules. Taking this into account, genetic operators could be designed to allow simple rules exchange mechanisms.

7 Acknowledgements

This research has been supported by the the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) “Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization” and by the Institutional Research Plan AV0Z10300504 ”Computer Science for the Information Society: Models, Algorithms, Appplications”.

References

1. D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.
2. E-puck, online documentation. http://www.e-puck.org.
3. D.B. Fogel. *Evolutionary Computation: The Fossil Record*. MIT-IEEE Press, 1998.
4. S. Haykin. *Neural Networks: a comprehensive foundation*. Prentice Hall, 2nd edition, 1999.
5. J. Holland. *Adaptation In Natural and Artificial Systems*. MIT Press, 1992.
6. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
7. J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:289–303, 1989.
8. S. Nolfi and D. Floreano. *Evolutionary Robotics — The Biology, Intelligence and Techology of Self-Organizing Machines.* The MIT Press, 2000.
9. R. Pfeifer and Ch. Scheier. *Understanding Intelligence*. The MIT Press, 2000.
10. T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1989. A. I. Memo No. 1140, C.B.I.P. Paper No. 31.
11. S. Slušný and R. Neruda. Evolving homing behaviour for team of robots. *Computational Intelligence, Robotics and Autonomous Systems. Palmerston North : Massey University*, 2007.
12. S. Slušný, R. Neruda, and P. Vidnerová. Evolution of simple behavior patterns for autonomous robotic agent. *System Science and Simulation in Engineering. - : WSEAS Press*, pages 411–417, 2007.
13. S. Richard Sutton and G. Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
14. C. J. C. H. Watkins. Learning from delayed rewards. *Ph.D. thesis*, 1989.
15. Webots simulator. on-line documentation http://www.cyberbotics.com/.