Real Time Robot Policy Adaptation Based on Intelligent Algorithms

Genci Capi, Hideki Toda, Shin-Ichiro Kaneko

Real Time Robot Policy Adaptation Based on Intelligent Algorithms

Genci Capi[1], Hideki Toda[1], Shin-ichiro Kaneko[2]
[1] Department of Electric and Electronic Eng.,
University of Toyama, Toyama, Japan
e-mail: capi@eng.u-toyama.ac.jp
[2] Toyama National College of Technology

Abstract. In this paper we present a new method for robot real time policy adaptation by combining learning and evolution. The robot adapts the policy as the environment conditions change. In our method, we apply evolutionary computation to find the optimal relation between reinforcement learning parameters and robot performance. The proposed algorithm is evaluated in the simulated environment of the Cyber Rodent (CR) robot, where the robot has to increase its energy level by capturing the active battery packs. The CR robot lives in two environments with different settings that replace each other four times. Results show that evolution can generate an optimal relation between the robot performance and exploration-exploitation of reinforcement learning, enabling the robot to adapt online its strategy as the environment conditions change.

Keywords: Reinforcement learning, policy adaptation, evolutionary computation.

1 Introduction

Reinforcement learning (RL) ([1], [2]) is an efficient learning framework for autonomous robots, in which the robot learns how to behave, from interactions with the environment, without explicit environment models or teacher signals. Most RL applications, so far, have been constrained to stationary environments. However, in many real-world tasks, the environment is not fixed. Therefore, the robot must change its strategy based on the environment conditions. For small environment changes, Minato et al., (2000) has pointed out that current knowledge learned in a previous environment is partially applicable even after the environment has changed, if we only consider reaching the goal and thereby sacrifice optimality ([3]).

Efforts have also been made to move in more dynamic environments. Matsui et al. ([4]) proposed a method, which senses a changing environment by collecting failed instances and partially modifies the strategy for adapting to subsequent changes of the environment by reinforcement learning. Doya incorporated a noise term in policies, in order to promote exploration ([5]). The size of noise is reduced as the performance improves. However, this method can be applied when the value function is known for all the states.

Previous approaches on combining learning and evolution ([6], [7], [8], [9]) reported that combination tends to provide earlier achievement of superior performance. Niv et al. have considered evolution of RL in uncertain environments ([10]). They solve near-optimal neuronal learning rules in order to allow simulated bees to respond rapidly to changes in reward contingencies. In our previous work, we considered evolution of metaparameters for faster convergence of reinforcement learning ([11], [12]). However, in all these approaches the robot learned the optimal strategy in stationary environment.

In difference from previous works, in this paper, we combine an actor-critic RL and evolution to develop robots able to adapt their strategy as the environment changes. The metaparameters, initial weight connection, number of hidden neurons of actor and critic networks, and the relation between the energy level and cooling factor are evolved by a real number Genetic Algorithm (GA). In order to test the effectiveness of the proposed algorithm, we considered a biologically inspired task for the CR robot ([13]). The robot must survive and increase its energy level by capturing the active battery packs distributed in the environment. The robot lives in two different environments, which substitute for each other four times during the robot's life. Therefore, the robot must adapt its strategy as the environment changes.

The performance of proposed method is compared with that of (a) RL and (b) evolution of neural controller. In the actor-critic RL, we used arbitrarily selected metaparameters, randomly initialized initial weight connections, and a linearly proportional relationship between cooling factor and energy level. The robot controlled by the evolved neural controller applies the same strategy throughout all its life, which was optimal only for one environment. The performance of the actor-critic RL was strongly related to the metaparameters, especially the relation between cooling factor and energy level. Combining learning and evolution gives the best performance overall. Because of optimized metaparameters and initial weight connections, the robot was able to exploit the environment from the beginning of its life. In addition, the robot switched between exploration and exploitation based on the optimized relation between the energy level and cooling factor.

2 Cyber Rodent Robot

In our simulations, we used the CR robot, which is a two-wheel-driven mobile robot, as shown in Fig. 1. The CR is 250 mm long and weights 1.7 kg. The CR is equipped with:

- Omni-directional C-MOS camera.
- IR range sensor.
- Seven IR proximity sensors.
- 3-axis acceleration sensor.
- 2-axis gyro sensor.
- Red, green and blue LED for visual signaling.
- Audio speaker and two microphones for acoustic communication.
- Infrared port to communicate with a nearby robot.

9 Wireless LAN card and USB port to communicate with the host computer.

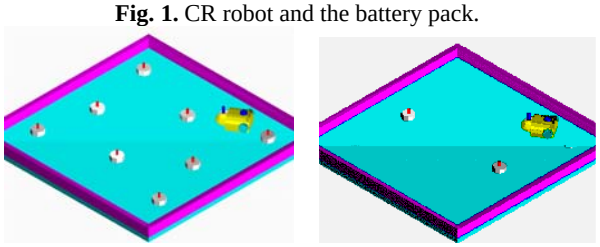
Five proximity sensors are positioned on the front of robot, one behind and one under the robot pointing downwards. The proximity sensor under the robot is used when the robot moves wheelie. The CR contains a Hitachi SH-4 CPU with 32 MB memory. The FPGA graphic processor is used for video capture and image processing at 30 Hz.

2.1 Environment

The CR robot has to survive and increase its energy level by capturing the active battery packs distributed in a rectangular environment of 2.5m x 3.5m (Fig. 2). The active battery packs have a red LED. After the charging time, the battery pack becomes inactive and its LED color changes to green and the battery becomes active again after the reactivation time. The CR robot is initially placed in a random position and orientation.

The robot lives in two different environments that alternatively substitute each-other four times. Based on environments settings, the robot must learn different policies in order to survive and increase its energy level. As shown in Fig. 2, the first and second environments have eight and two battery packs, respectively. In the first environment, the batteries have a long reactivation time. In addition, the energy consumed for 1m motion is low. Therefore, the best policy is to capture any visible battery pack (the nearest when there are more than one). When there is no visible active battery pack, the robot have to search in the environment. In the second environment, the reactivation time is short and the energy consumed during robot motion is increased. Therefore, the optimal policy is to wait until the previously captured battery pack becomes active again rather than searching for other active battery packs.

Fig. 1. CR robot and the battery pack.



(a) Environment 1. (b) Environment 2.

Fig. 2. Environments.

3 Intelligent Algorithms

Consider the Cyber Rodent robot in an environment where at any given time *t*, the robot is able to choose an action. Also, at any given time *t*, the environment provides the robot with a reward r_t. Our implementation of the actor-critic has three parts: 1) an input layer of robot state; 2) a critic network that learns appropriate weights from the state to enable it to output information about the value of particular state; 3) an actor network that learns the appropriate weights from the state, which enable it to represent the action the robot should make in a particular state. Each time step, the robot selects one of the following actions: 1) Capture the battery pack; 2) Search for a battery pack; 3) Wait for a determined period of time. The wait behavior is interrupted if a battery becomes active. Both networks receive as input a constant bias input, the battery level and distance to the nearest active battery pack (both normalized between 0 and 1).

3.1 Critic

The standard approach is for the critic to attempt to learn the value function, $V(x)$, which is really an evaluation of the actions currently specified by the actor. The value function is usually defined as, for any state x, the discounted total future reward that is expected, on average, to accrue after being in state x and then following the actions currently specified by the actor. If x_t is the state at time t, we may define the value as:

$$V(x_t)=<r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+\ldots>, \quad (1)$$

where γ is a constant discounting factor, set such that $0<\gamma<1$ and $<\cdot>$ denotes the mean over all trials. $V(x)$ can actually suggest an improvement to the actions of the actor, since an action, which leads to a large increase in the value, is guaranteed to increase the battery level. Therefore, a good strategy for the actor is to try several actions for each state, with aim of choosing the action that involves the largest increase in value.

However, the value function is not given; the critic must learn it using TD learning, i.e., the weights must be adapted so that $O_c(x)=V(x)$. TD works by enforcing *consistency* between successive critic outputs. Specifically, the following relationship holds between successively occurring values, $V(x_t)$ and $V(x_{t+1})$:

$$V(x_t)=<r_t>+\gamma V(x_{t+1}). \quad (2)$$

If it were true that $O_c(p)=V(p)$, then a similar relationship should hold between successively occurring critic outputs $O_c(x_t)$ and $O_c(x_{t+1})$:

$$O_c(x_t)=<r_t>+\gamma O_c(x_{t+1}). \quad (3)$$

TD uses the actual difference between the two sides of eq. 5 as a prediction error, δ_t, which drives learning:

$$\delta_t=r_t+\gamma O_c(x_{t+1})-O_c(x_t). \quad (4)$$

The TD error is calculated as follows:

$$\hat{r}[t+1]=\begin{cases}0 \text{ if the start state}\\ \mathrm{r}[\mathrm{t}+1]+\gamma^{\mathrm{k}} v[t+1]-v[t] \text{ otherwise}\end{cases}, \quad (5)$$

using the reward $r_{t+1}=(En\_level_{t+1}-En\_level_t)/50$. TD reduces the error by changing the weights.

3.1 Actor

The robot can select one of three actions and so the actor make use of three action cells, p_j, $j=1,2,3$. The captured behavior is pre-evolved (Capi et al. 2003) using the angle to the nearest battery pack as input of neural controller. When the search behavior is activated, the robot rotates 10 degrees clockwise. The robot does not move when the wait behavior becomes active. A winner-take-all rule prevents the actor from performing two actions at the same time.
The action is selected based on the softmax method as follows:

$$P(a, s_t) = \frac{e^{p_i(a,s)\beta}}{\sum_{i=1}^{3}(e^{p_i(a,s)\beta})}, \tag{6}$$

where β is the cooling factor.

Following the logic described above, the actor should try various actions at each state, with the aim of choosing the action, which produces the greatest increase in value. The stochastic action choice ensures that many different actions are tried at similar states. To choose the best action, a signal is required from the critic about the change in that result from taking an action. It turns out that an appropriate signal is the same prediction error, $\mathcal{H}_t$, used in the learning of the value function.

3.3 Combining learning and evolution

In our implementation, we used the actor-critic RL as explained previously. The metaparameters У 1, У 2, Γ 2−! Γ 3− Ε− the initial weight connections and the number of hidden neurons of both actor and critic networks are evolved by GA. In addition, GA optimizes the relation between the cooling factor β and energy level, by optimizing the values of β_0, en_1, β_1, en_2, β_2, and β_3, as shown in Fig. 3.

In order to force the evolution process to select individuals that live longer and have a higher energy level, the fitness is designed as follows:

$$fitness = \begin{cases} \dfrac{\sum_{i=1}^{4} En_i}{4} + \dfrac{CR_{\max\_life}}{100} \text{ if the agent survives} \\ En_{\min} + \dfrac{CR_{life}}{100} \text{ if the agent dies} \end{cases}, \tag{7}$$

where En_i is the energy level at the moment of each environment change and CR_{max_life} is maximum life time in seconds, En_{min} is the energy level when robot dies, CR_{life} is the time in seconds until the robot dies.

A real-value GA was employed in conjunction with the selection, mutation and crossover operators. Many experiments comparing real-value and binary GA show that real-value GA generates superior results in terms of the solution quality and CPU time.

3.4 Evolution of neural controller

We evolved the weight connections and number of hidden units of a feedforward neural network for the surviving behavior. The inputs of the neural network are the energy level and distance to the nearest battery pack. The weight connections have been fixed throughout the robot's life. Every time step one action is selected based on the selection probability, as follows:

$$P(a_i) = \frac{p'_i}{\sum_{i=1}^{3} p'_i} \tag{8}$$

where $P(a_i)$ is the probability of selecting action a_i, and p_i is the output of *i-th* neuron. The fitness value is calculated according to eq. 7.

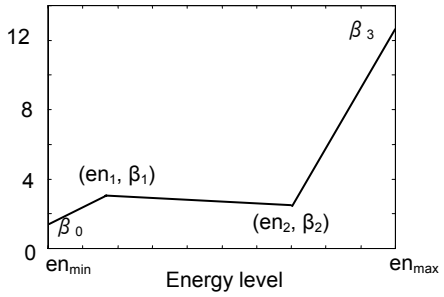


Fig. 3. Optimized cooling factor.

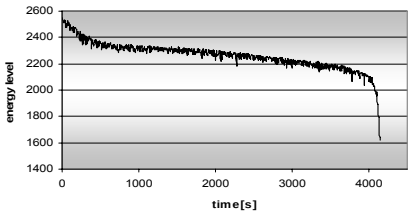


Fig. 4. Energy level during CR motion.

4 Results

In order to determine the energy after each action, we recorded how the energy level changes by time, as the CR robot moves in the environment. The digital readings of energy level are shown in Fig. 4. Initially, the battery is fully recharged. The robot stops moving when the battery level goes under 1900. Because of the nonlinear relation, we used the virtual time to determine the energy level after each action, as shown in Table 2. Except for capturing the battery pack, all the other actions increased the virtual time. The maximum lifetime of the robot is 360 min and the environment changes every 90 minutes.

Table 1. Change in the virtual time for each action.

Environment settings	Env. 1	Env. 2
Capturing the battery pack	-60s	-30s
Moving 1m distance	4s	15s
Searching	1s	1s
Waiting	5s	5s
Battery reactivation time	100s	10s

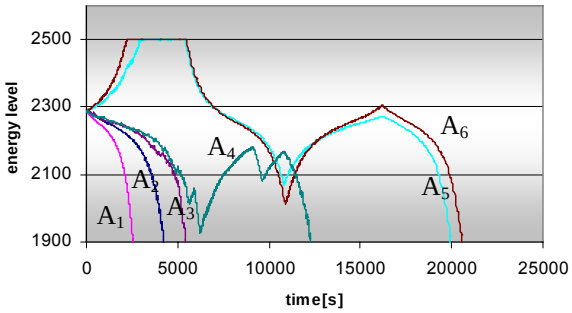


Fig. 5. Performance of different individuals from the first generation.

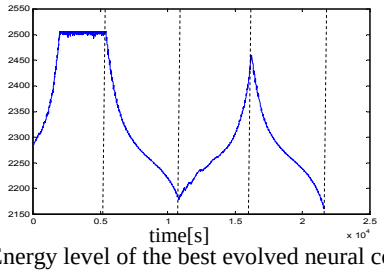


Fig. 6. Energy level of the best evolved neural controller.

4.1 Evolution of neural controller

Initially 100 neural controllers were generated randomly. The performance of six different individuals selected from the first generation, is shown in Fig. 5. The robots A_1, A_2, and A_3 die shortly after they were placed in the environment. The robot A_4 reaches any visible active battery pack. When there is no visible active battery pack, the probability of selecting the wait action is higher than search action. Therefore, the robot performs better in the second environment. However, the search action is also selected, which resulted in a slow decrease of energy level in the first environment and some rapid energy decrease in second environment. Robots A_5 and A_6 apply a strategy, which is suitable for the first environment and the energy reduces rapidly in the second environment.

The best neural controller generated by evolution has 2 hidden units. Fig. 6 shows the energy level during the robot life. The robot applied the same strategy throughout its life, which is the optimal strategy for the first environment. When there is no visible active battery pack, mainly search action is generated. The probability of selecting the wait behavior was low, but sometimes it was generated which resulted in slower decrease of energy level in the second environment.

4.2 Actor-critic RL

In this section, we compare the performance of the actor-critic RL using arbitrary selected metaparameters (Table 3) and random initial weight connections in the interval [-0.5 0.5]. The relation between β and energy level is considered linear where 1 and 10 correspond to empty and full battery level, respectively.

Fig. 6 shows the energy level during the time course of learning. At the beginning of robot life, the energy decreases because the initial weight connections are randomly generated. As the learning continues, the weight connections of both actor and critic networks are modified and energy level is increased. When the environment changes, the energy level decreases. Therefore, the robot starts to explore the environment. However, the robot was unable to survive in all four environments. The battery becomes empty after 20000 sec.

Table 3. Metaparameters used in actor-critic RL.

RL parameters	Values
γ_1	1
$\gamma_2!$	0.5
$\Gamma_2!$	0.4
$\Gamma_3!$	1
E	0.9
$\beta!$	Linear 1-10

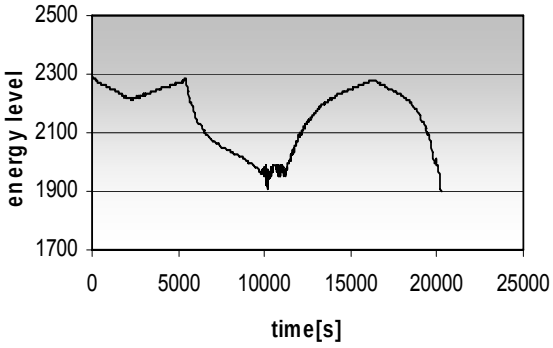


Fig. 7. Performance of actor-critic RL.

Table 4. Optimized metaparameters.

Optimized parameters	Searching interval	Results
Γ_2!	[0 1]	0.4584
Γ_3!	[0 1]	0.4614
$У_1$	[0 1]	0.1722
$У_2$!	[0 1]	0.4521
E	[0 1]	0.6360

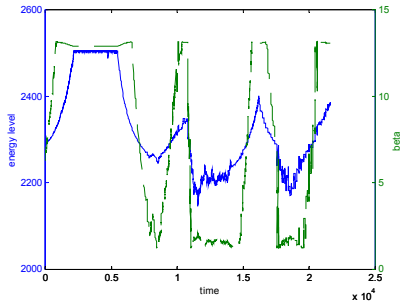


Fig. 8. Energy level and cooling factor during the robot life.

4.3 Combining learning and evolution

In our simulations the population size is 100 and the evolution terminated after 20 generations. Each individual of the population has different values of metaparameters and initial weight connections. In the first generation, most of the individuals could not survive in four environments. Based on the fitness value, the individuals that survived longer have higher probability to continue in the next generation.

The searching interval and GA results are shown in Table 4. The actor and critic networks have 2 and 4 hidden neurons, respectively. The optimal relation between the energy level and cooling factor (β) shows that β is slightly increased when the energy level goes to minimum. The minimum value of β is for 59% of full battery level. When the energy level is higher than 72%, β becomes high.

Fig. 8 shows the energy level during the robot life, utilizing the evolved metaparameters, initial weight connections and optimized relation between cooling factor and energy level. At the beginning of robot life, due to large value of β and optimized initial weight connections, the robot starts exploiting the environment. Because the energy consumed for 1m motion is small, the best strategy is to capture any visible active battery pack or search otherwise.

When the environment changes, due to the large value of β, the robot follows the previous strategy. As the energy decreases, β gets lower. Therefore, the robot starts to explore the environment and to adapt its strategy to the new environment conditions. In the second environment, the reactivation time is very short and energy consumed for 1m motion is higher compared to the first environment. Therefore, the robot, instead of searching for an active battery pack, waits until the previous captured battery pack becomes active.

4 Conclusion

In this paper, we considered combining learning and evolution in order to deal with non-stationary environments. The results of this paper can be summarized as follows:

- Metaparameters and initial weight connections optimized by GA helped the robot to adapt much faster during the first stage of life.
- Based on the relation between the energy level and cooling factor, the robot was able to adapt its strategy as the environment changed.
- The robot controlled by an evolved neural controller applied always the same strategy, which was the optimal only in one of the environments.

The performance of actor-critic RL was strongly related to the values of metaparameters, especially the relation between cooling factor and energy level.

References

1. Sutton, R.S. and Barto, A.G.: Reinforcement learning, MIT Press, Cambridge, MA, USA, (1998).
2. Kaelbling, L. P., Littman, M. L. and Moore, A. W.: Reinforcement learning: A survey, Journal of Artificial Intelligence Research, 4, 237-285, (1996).
3. Minato, T. and Asada, M.: Environmental change adaptation for mobile robot navigation, Journal of Robotics Society of Japan, 18(5), 706-712, (2000).
4. Matsui, T., Inuzuka, N. and Seki, H.: Adapting to subsequent changes of environment by learning policy preconditions, Int. Journal of Computer and Information Science, 3(1), 49-58, (2002).
5. Doya, K.: Reinforcement learning in continuous time and space, Neural Computation, 12, 219-245, (2000).
6. Belew, R. K., McInerney, J., and Schraudolph, N. N.: Evolving networks: using the genetic algorithm with connectionist learning, in: Artificial Life II , edited by C.G. Langton, et al., Addison Wesley, 511-547, (1990).
7. Unemi, T.: Evolutionary differentiation of learning abilities - a case study on optimizing parameter values in Q-learning by a genetic algorithm, in: Artificial Life IV - Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, edited by R. A. Brooks and P. Maes, MIT Press, 331-336, (1994).
8. French, R. M., and Messinger, A.: Genes, phenes and the baldwin effect: learning and evolution in a simulated population, Proc. of Forth Int. Workshop on the Synthesis and Simulation of Living Systems, 277-282, (1977).
9. Nolfi, S., and Parisi, D.: Learning to adapt to changing environments in evolving neural networks, Adaptive Behavior, 5(1), 75-98, (1996).
10. Niv, Y., Joel, D., and Meilijson, I., and Ruppin, E.: Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors, Adaptive Behavior, 10(1), 5-24, (2002).
11. Capi, G. and Doya, K. (2005). Application of evolutionary computation for efficient reinforcement learning, Applied Artificial Intelligence, 20(1), 1-20.
12. Eriksson, A., Capi, G., and Doya, K.: Evolution of metaparameters in reinforcement learning, IROS2003, (2003)
13. Capi, G.: Multiobjective Evolution of Neural Controllers and Task Complexity, IEEE Transactions on Robotics, 23(6), 1225-1234, (2007).