

# Evolution of metaparameters in reinforcement learning algorithm

Anders Eriksson<sup>1</sup>, Genci Capi<sup>2</sup>, Kenji Doya<sup>2</sup>

<sup>1</sup> Numerical Analysis and Computer Science, KTH  
KTH, NADA, 100 44 Stockholm, Sweden

<sup>2</sup> Human Information Science Laboratories, ATR International  
2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan

## Abstract

In most Reinforcement Learning approaches, the meta-parameters such as learning rate and "temperature" for exploration are adjusted manually. In order to build fully autonomous learning agents, it is important to develop methods for adjusting these parameters to match the demands of the task and the environment. In this paper, we propose a new method to determine the values of meta parameters in reinforcement learning, based on evolutionary approach. Simulations and experimental results with the Cyber Rodent robot show that meta parameters have a strong effect on agent performance and they are strongly related with each-other.

## 1 Introduction

Reinforcement learning (RL) has been receiving increased attention as a method with little or no a priori knowledge for acquiring adaptive behaviors [1]. However, a crucial issue in RL is to determine the values of meta parameters [2], which play a very important role in the agent performance. In most of the previous approaches the meta parameters are determined based on the experience of the experimenter. In this paper, we present a new method to determine the optimal values of meta parameters by evolution. In [4] the authors evolved the meta parameters in a simulated environment using a discrete state and action.

In this study, the agent's task is to capture the battery packs. The agent state is the angle to the nearest battery, which has continuous state space and action space. A real number GA was employed to evolve the learning rate and cooling factor. Real number GA gives better results compared to binary GA in terms

of CPU time and quality of the solution. Simulations and experiments were realized using the Cyber Rodent (CR) robot. The omni-directional camera determines the angle to the nearest battery pack. First learning and optimization of meta-parameters were completed by simulation. The optimized meta-parameters were utilized to learn the capturing behavior in real hardware. The simulation and experimental results show that meta parameters are related with each other and they strongly influence the agent performance. The evolved meta parameters used in the hardware implementation of RL gives good results in terms of the agent task performance and learning time.

This paper is organized as follows: The proposed method is discussed in Section 2. In Section 3 is presented the CR robot. Section 4 deals with the task and reward. Simulation and experimental results are given in Section 5. Finally, conclusions are given in Section 6.

## 2 Proposed method

In RL many parameters has to be determined before learning takes place. Meta parameters and initial weight connections influences the learned policy, learning time and agent performance. By combining RL and GA, the initial parameters can be optimized in order to get the best agent performance and minimize the learning time.

In our method, first an initial population is generated randomly. Every individual of the population encodes the meta-parameters that we have to optimize. After the RL is completed, the agent behaviour is tested in different environment settings. Based on the agent performance, the fitness value is calculated and attached to every individual of the population.

The fitness is calculated based on the number of steps used to reach the battery pack after learning and the reward accumulated by the agent during last part of learning. The best individuals are selected and the genetic operators are performed until the termination criteria is satisfied. The evolution is terminated when the individuals became very similar. The most common individual is considered as the optimal solution. The optimal values of meta-parameters generated by GA are used to learn a capturing behavior in simulated and real environments. We considered the optimization of learning rate and cooling factor. The proposed method can also be applied to optimize other meta parameters and initial weight connections.

### 3 CR robot

The CR robot is a two wheel driven mobile robot as shown in Figure 1. It is 250mm long and weights 1.7 kg. The CR is capable of moving wheelie and it is equipped with:

- Omni-directional CCD camera.
- Ultrasonic range sensor.
- Seven infrared proximity sensors.
- 3-axis acceleration sensor.
- 2-axis gyro sensor.
- Red, green and blue LEDs for visual signaling.
- Audio speaker and two microphones for acoustic communication.
- Infrared port to communicate with a near-by-agent.
- Wireless LAN card and USB port to communicate with the host computer.

The CR has a Hitachi SH-4 CPU with 32 MB memory. The FPGA graphic processor is used for video capture and image processing at 30 Hz. 5 proximity sensors are in the front of robot, 1 behind and 1 under the robot pointing downwards. The proximity sensor under the robot is needed when the robot moves wheelie.



Figure 1: CR robot and battery pack.

### 4 Task and reward

In our work, the agent’s task is to capture the battery packs, which are distributed in the environment. It allows us to study the delayed reward which is strongly related to meta-parameters. Also, it makes easier to know the learning dynamics and make correct conclusions.

We considered a square environment where the battery packs are randomly distributed. At each time step the agent receives the angle to the closest battery as state input. The agent range of vision is  $[-\pi/2, \pi/2]$ . The agent can choose from seven discrete actions. The wheel velocities are calculated as follows:

$$\begin{aligned} \omega_{LeftWheel}(a) &= \omega_{Const} * \frac{(a-1)}{A}, \\ \omega_{RightWheel}(a) &= \omega_{Const} * \frac{A-(a-1)}{A}, \end{aligned} \quad (1)$$

where  $A$  is the total number of actions,  $a$  is the action  $1, 2, \dots, A$  and  $\omega_{Const}$  is the constant angular velocity. In this way, the action spans from turning left to turning right.

The reward, which consists of two parts, is calculated as follows:

$$reward = \begin{cases} 0 & |v| > 0.2\pi \\ \frac{0.1}{\pi}(0.2\pi - |v|) & |v| \leq 0.2\pi \\ 1 & \text{reaching battery,} \end{cases} \quad (2)$$

where  $v$  is the angle to the battery. A small reward is given based on the angle to the battery which is maximum 2 % of the total reward. The main reward is given when the agent captures the battery pack.

#### 4.1 RL

The agent state is the angle to the nearest battery pack which has continuous value. We used a radial basis

function network to represent the action value with  $n$  Gaussian basis functions as follows:

$$\phi(i, s) = e^{-\frac{\|s-c_i\|^2}{2\sigma_i^2}}, \quad (3)$$

where  $c_i$  is the center and  $\sigma_i$  is the width of the  $i$ th basis function. The action value function is defined as

$$Q(s, a) = \sum_{i=1}^n \phi(i, s)\theta(i, a), \quad (4)$$

where  $\theta(i, a)$  is the weight value for basis function  $i$  and action  $a$ . The Gaussian basis functions are distributed over one dimension angle input space and a weighted sum of these gives the action value function for each discrete action. Two states have been added to represent with the situation when the battery packs get out of agent's sight field, one for losing the battery pack to the left and one for losing to the right.

The weight matrix that controls the behavior of the agent is updated according to the *gradient decent Sarsa*( $\lambda$ ) [1] RL algorithm. *Gradient decent Sarsa*( $\lambda$ ) is based on  $TD(\lambda)$  and the incremental update equation is:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \delta_t \vec{e}_t, \quad (5)$$

where  $\vec{\theta}_t$  are the network connection weights,  $\alpha$  is the learning rate,  $\delta_t$  is the temporal difference error and  $\vec{e}_t$  is the trace matrix distributing  $\delta_t$  over the states recently visited.  $\delta_t$  and  $\vec{e}_t$  are calculated as follows:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t), \quad (6)$$

$$\vec{e}_t = \gamma \lambda \vec{e}_{t-1} + \Delta_{\vec{\theta}_t} Q_t(s_t, a_t), \quad (7)$$

where  $\gamma$  and  $\lambda$  are respectively the discount factor and trace decreasing factor.  $\alpha$  is the step size by which the incremental update rule approaches the estimation of the state action value function. The state space is explored using a softmax method, employing the Boltzmann distribution. The probability of choosing an action  $a$  at time  $t$  is given as:

$$P(a|s_t) = \frac{e^{Q_t(a, s_t)/\tau}}{\sum_{i=1}^A (e^{Q_t(a_i, s_t)/\tau})}, \quad (8)$$

where  $\tau$  is the *temperature* of the algorithm. When the temperature is high ( $\lim_{\tau \rightarrow \infty}$ ), the probability of choosing any action is equal. As the temperature decreases, the probability of choosing an action at time  $t$  is proportional to the action's  $Q$ -value. When the temperature approaches zero, the action with the highest  $Q$ -value is selected. After each action taken by the agent, the temperature is decreased exponentially:

$$\tau = \tau_{df} \tau, \quad (9)$$

where  $\tau_{df}$  is the temperature decreasing factor. The RL performance is strongly related to the values of  $\vec{\theta}_{initial}$ ,  $\tau_{initial}$ ,  $\tau_{df}$ ,  $\alpha$ ,  $\gamma$ ,  $\lambda$  and the total number of steps. In our work, we considered the influence of the meta parameters  $\alpha$  and  $\tau_{df}$ . The values of other parameters are considered as follows:

$$\begin{aligned} \vec{\theta}_{initial} &\in [0, 0.05], \\ \tau_{initial} &= 10, \\ \gamma &= 1, \\ \lambda &= 0.1. \end{aligned}$$

## 4.2 GA

In our implementation, we used a real number GA [5]. Every individual of the population has genes encoding the meta parameters. The searching space for the meta parameters is predefined,  $\tau_{df}=[0.9, 1]$  and  $\alpha=[0.001, 0.2]$ . We evaluate the agent performance based on the following criteria:

1. Number of steps used to reach the battery pack when the agent is placed in predefined positions.
2. Reward accumulated by the agent during last part of learning.

When all individuals of a population have been evaluated, parents are selected based on the fitness value. Individuals of the next generation are created by applying crossover and mutation operators. We use the similarity of the individuals in the population as the termination criteria. The most common individual of the last generation are selected as the optimal solution. The genetic operators and their respective parameters are shown in Table 1. The number specifies how many times the operator is applied to each generation.

## 5 Simulation and experimental results

The experiments were performed in two different environments. Our goal was to know the interaction between  $\alpha$  and  $\tau_{df}$  and how simulation results can be implemented in real hardware.

### 5.1 Single food capture task

In the first experiment, the agent is placed in a simple environment containing a single battery pack. The agent starts learning in short distance relative to the battery pack (300 mm), orientated so that the battery

Type	Name	Number
Mutation	Uniform	5
	Non uniform	4
	Multi non uniform	6
	Boundary mutation	2
Crossover	Simple	2
	Heuristic	2
	Arithmetic	2
Reproduction	Simple	5

Table 1: Genetic operators and their parameters.

pack is always visible. In the late stage of leaning, the position and orientation of the agent relative to the battery are randomly selected. The episode length (number of steps) differs in the two stages. The fitness value is calculated when the learning is completed. The derived policy is evaluated by letting the agent capturing 18 battery packs from predefined positions. The population size is 50. After 7 generation the individuals becomes very similar with each other and the evaluation was terminated.

Figure 2 shows the evolved values of meta parameters from five simulations. The figure shows the relation between  $\alpha$  and  $\tau_{df}$ . If the algorithm cools of early (small  $\tau_{df}$ ) the step size towards the estimated value function must be large (large  $\alpha$ ). When the algorithm is allowed to cool off slowly, the step size can be smaller and the estimation of the value function is more correct. The average fitness (connected line) and the standard deviation (dashed lines) of the last populations shows the performance of the evolved solutions. The reliability increases when  $\tau_{df}$  decreases.

## 5.2 Multiple food capture task

In the second experiment, the battery packs are delivered in the environment and the agent searched the environment without getting repositioned. An avoiding behavior is generated when the agent gets at a 250 mm distance from walls. The number of battery packs decreases linearly from 15 to 1 during the agent’s lifetime. During the last 1300 steps of the agents lifetime, the fitness value is calculated directly from the collected reward.

Figure 3 shows the average reward curve for the initial and the last population of individuals with 12000 basic steps as lifetime. Also, the temperature  $\tau$  of the evolved individual and the number of battery packs are presented. The results is not what we expected. It is expected that the best solution should be to wait as

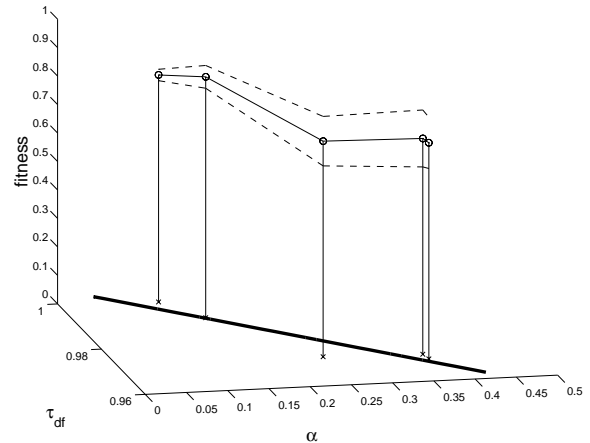


Figure 2: Relation between  $\alpha$  and  $\tau_{df}$ . The average fitness and fitness standard deviation of the last populations shows the performance of the solutions

long as possible to cool of ( $\tau$  approach zero) just before beginning of the fitness measurement. However, the solution shows that it is better to cool of somewhere at ten to eight batteries. This result shows that the learning procedure is very sensitive at the time of decision (when  $\tau$  approach zero). Searching the environment randomly by making big turns gives better results than almost following the optimal policy. Feedback is needed to confirm the right policy when cooling down, and therefore the agent is dependent of numerous batteries in this stage of learning. The course of evolution for the two parameters is presented in figure 4.

Figure 5 shows the relation between  $\alpha$  and  $\tau_{df}$  after evolution from eight experiments using different lifetimes.

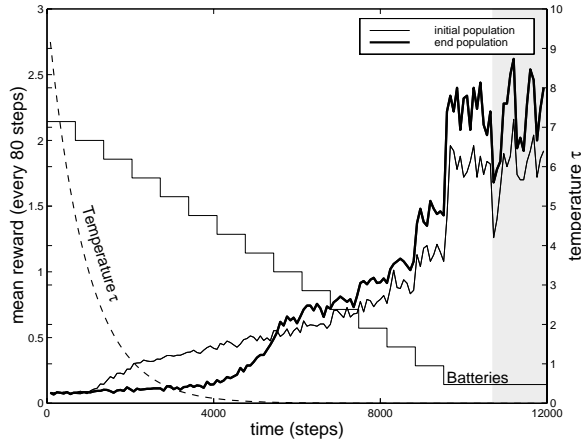


Figure 3: Reward curve for initial and last populations, battery level, fitness measure time interval and temperature curve of the best individual.

### 5.3 Hardware implementation

In order to see how the optimized meta parameters in simulation will perform in real hardware system, we implemented the RL algorithm by using the CR robot. The environment and the learning conditions are identical to the task in section 5.2. Meta parameters used in the experiment are evolved by GA where each individual has a lifetime of 7000 steps.

The processing time and the procedure after capturing a battery pack are different in the simulation and hardware implementation. Therefore, we increased the step size and reduced the speed in hardware implementation.

Figures 6 and 7 show the learned policy in simulation and hardware implementation, respectively. These figures show that there are two differences between the solutions. First, the straight forward action (action 4) is used for a wider range of angle state in the hardware solution. This is related to the CR design. The CR has two "claws" to collect in the battery pack, even if it is not straight in front of the agent. The other difference is the behavior when there is no battery pack visible. In the simulation, the agent turns around to search for the battery packs. In the hardware implementation, the agent uses the straight forward action. The reason is the CR vision range. In difference from the simulator the CR robot has shorter vision range. Therefore the hardware implementation found as a better action for searching behavior to move forward than to rotate.

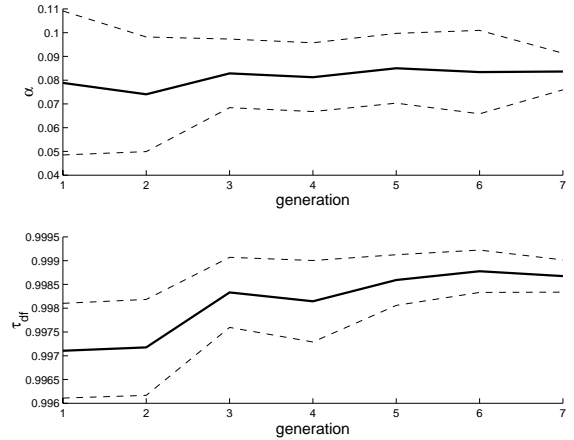


Figure 4: The course of evolution showing the evolution of mean and standard deviation values for  $\alpha$  and  $\tau_{df}$ .

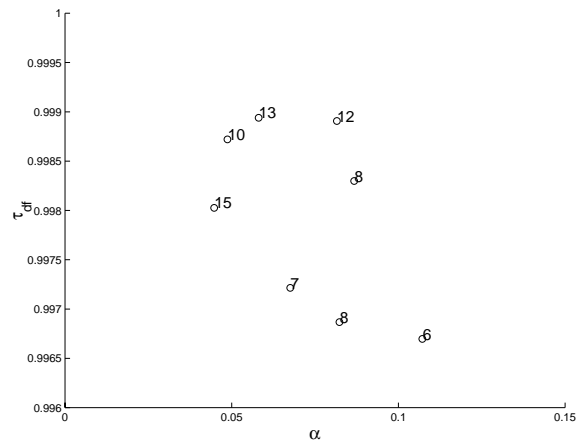


Figure 5: Evolved individuals from eight experiments in the multiple food task. The agents lifetimes in number of 1000 basic steps are specified for each experiment.

## 6 Conclusions

In this paper, we presented a new method to optimize the meta parameters in RL based on evolutionary approach. By combining RL and evolutionary framework, it has been shown that the optimal values of meta parameters can be found.

One important result is the interaction between the meta parameters  $\alpha$  and  $\tau_{df}$ . The meta parameters optimized in simulation were also applied to learn the capturing behavior in a real hardware system. The

implementation showed good results.

In the future, we plan to consider the optimization of more meta parameters. Most interesting are the discount factor  $\gamma$  and the trace decrease factor  $\lambda$ .

## Acknowledgment

This study was partially supported by the Sweden-Japan foundation.

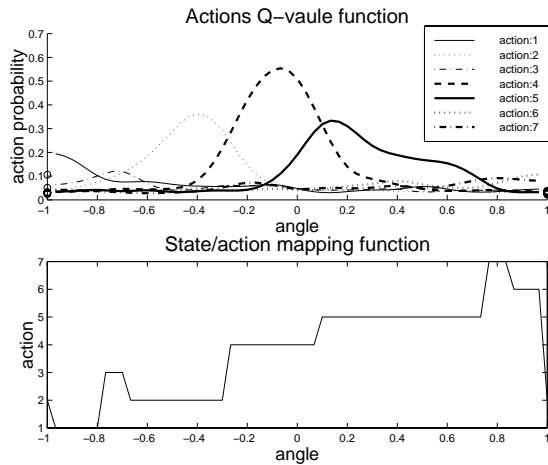


Figure 6: Action value functions and learned policy in simulation.

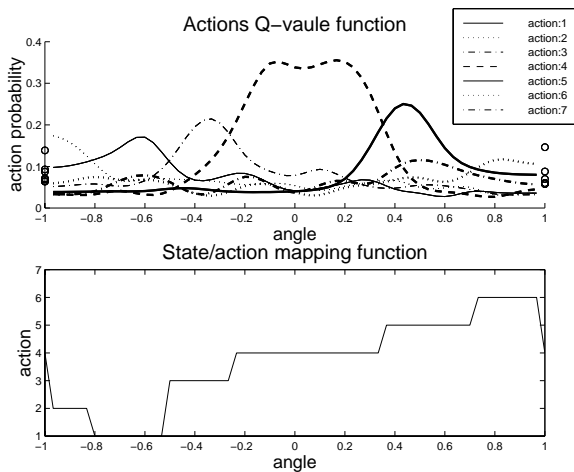


Figure 7: Action value functions and learned policy in hardware implementation.

## References

- [1] Sutton R. and Barto A. *Reinforcement learning: An introduction* MIT Press, Cambridge, MA, 1998.
- [2] Doya K. Metalearning and neuromodulation *Neural Networks*, vol 15, no 4/5, 2002.
- [3] Iskii S., Yoshida W. & Yoshimoto J. Control of Exploitation-exploration Meta-parameter in Reinforcement Learning. *Neural Networks*, 15(4/5), 2002
- [4] Unemi T. et. al. Evolutionary differentiation of learning abilities - a case study on optimizing parameter values in Q-learning by genetic algorithm, *Artificial Life IV*, 331-336, MIT Press, 1994.
- [5] Michalewicz Z. *Genetic Algorithms + Data Structures = Evaluation Programs*, Springer Verlag, 1994.