Evolutionary Reinforcement Learning for Simulated Locomotion of a Robot with a Two-link Arm

Yohannes Kassahun and Gerald Sommer

Christian Albrechts University, Institute of Computer Science and Applied Mathematics, Department of Cognitive Systems, Olshausenstr. 40, D-24098, Kiel, Germany

Abstract. In this paper we present a neural controller design for robots using an evolutionary reinforcement learning system that is suitable for learning through interaction. The method starts with networks of minimal structures determined by the domain expert and increases their complexity along the evolution path. It uses a nature inspired meta-level evolutionary process where new structures are explored at larger time-scale and existing structures are exploited at smaller time-scale. The method introduces an efficient and compact genetic encoding of neural networks onto a linear genome that enables one to evaluate the network without decoding it. We demonstrate the method by designing a neural controller for a robot with a two-link arm that enables it to move forward as fast as possible. We first give an introduction to the evolutionary method and then describe the experiment and results obtained.

Keywords. Neural networks, Reinforcement learning, Evolutionary algorithms

1. Introduction

A meaningful combination of the principles of neural networks, reinforcement learning and evolutionary computation is useful for designing agents that learn and adapt to their environment through interaction [4]. The combination results in an evolutionary reinforcement learning system where each of the components of the learning system plays an important role.

Neural networks are useful for evolving the control system of an agent [8]. They provide a straight forward mapping between sensors and motors and this enables them to represent directly the policy (control) or the value function to be learned. Reinforcement learning is useful as a type of learning where the agent is not told directly what to do but fed with a signal (reward) that measures the quality of executing an action in a given state [12]. The purpose of the agent is to act optimally in its environment so as to maximize its rewards. It is one form of learning through interaction. Learning through interaction underlines nearly all the principles of intelligence [9]. This property of reinforcement learning makes it important in evolutionary reinforcement learning system. Like neural networks, evolutionary algorithms [2,7,10] are inspired from biology. Populations of organisms have been adapting to their particular environmental conditions through evolutionary selection (survival of the fittest) and variablity among them. From these principles of adaptation in nature, it is possible to derive a number of concepts and strategies for solving learning tasks and develop optimization strategies for artificial intelligent systems.

The evolutionary reinforcement learning system that combines the principles of neural networks, reinforcement learning and evolutionary algorithms is shown in figure 1. The evolutionary algorithm contains genotypes of neural networks to be evaluated in a given environment. Each neural network is evaluated and assigned a given fitness value (reward). Through genetic operators of the evolutionary algorithm, the agents will be improved. The process continues until a certain number of generations or until an agent is found that solves a given task.

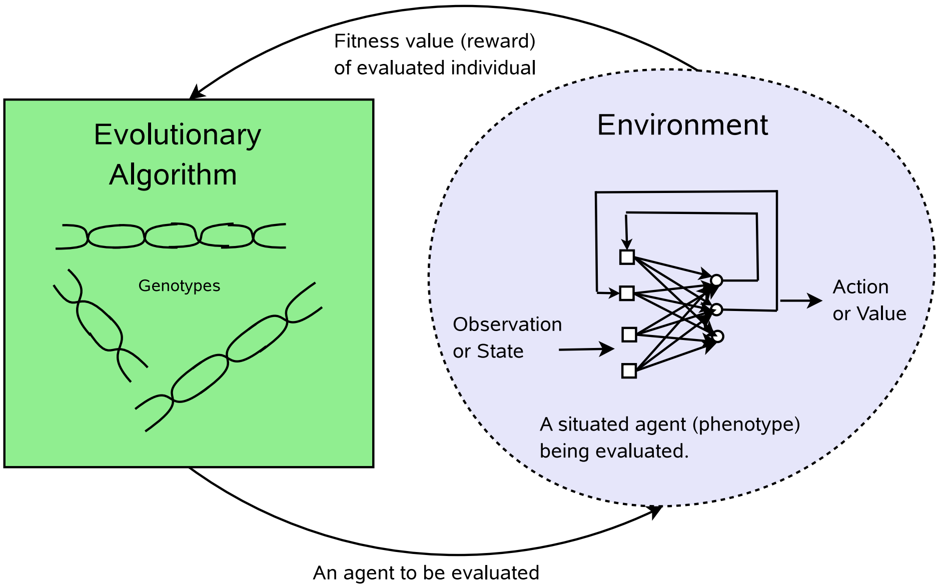
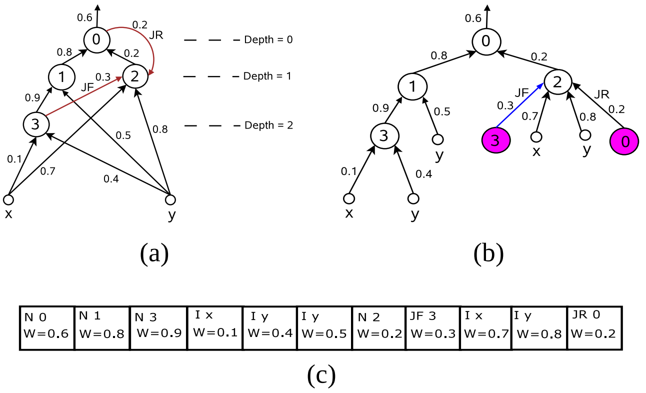


Figure 1. Evolutionary reinforcement learning system. The agents, where the neural networks are embedded in, are evaluated in the environment and their fitness values are returned to the evolutionary algorithm as rewards.

2. Evolutionary Acquisition of Neural Topologies

Evolutionary Acquisition of Neural Topologies (ENAT) [5] is an evolutionary reinforcement learning system that is suitable for learning and adaptation to the environment through interaction. It combines meaningfully the principles of neural networks, reinforcement learning and evolutionary methods.

The method introduces a novel genetic encoding that uses a linear genome of genes (nodes) that can take different forms. The forms that can be taken by a gene can either be a neuron, or an input to the neural network, or a jumper connecting two neurons. The jumper genes are introduced by the structural mutation along the evolution path. They encode either forward or recurrent connections. In addition to the synaptic weight, a neuron node has a unique global identification number and number of input connections to it. A jumper node has also additionally a global identification number, which shows the neuron to which it is connected. Figure 2 shows an example of encoding a neural network using a linear genome. As can be seen in the figure, a linear genome can be interpreted as a tree based program if one considers all the inputs to the network and all jumper connections as terminals.



(a)

(b)

N 0 W=0.6	N 1 W=0.8	N 3 W=0.9	I x W=0.1	I y W=0.4	I y W=0.5	N 2 W=0.2	JF 3 W=0.3	I x W=0.7	I y W=0.8	JR 0 W=0.2

(c)

Figure 2. An example of encoding a neural network using a linear genome. (a) The neural network to be encoded. It has one forward and one recurrent jumper connection. (b) The neural network interpreted as a tree structure, where the jumper connections are considered as terminals. (c) The linear genome encoding the neural network shown in (a). In the linear genome, N stands for a neuron, I for an input to the neural network, JF for a forward jumper connection, and JR for a recurrent jumper connection. The numbers beside N represent the global identification numbers of the neurons, and x or y represent the inputs coded by the input gene (node).

The linear genome has some interesting properties that makes it useful for evolution of neural networks. It encodes the topology of a neural network implicitly in the ordering of the elements of the linear genome. This enables one to evaluate the network represented by it without decoding the neural network. That means, it is possible to evaluate the neural network encoded by the linear genome without some type of ontological process of transforming the genotype into phenotype. The evaluation of a linear genome is closely related to executing a linear program using a postfix notation. In the genetic encoding the operands (inputs and jumper connections) come before the operator (a neuron) if one goes from right to left along the linear genome. The process of evaluating a linear genome without decoding the neural network encoded by it is performed as follows. One starts from the right most node of the linear genome and then moves to the left in computing the output of the nodes. If the current node is an input node, then its current value and the weight associated with it is pushed onto the stack. If the current node is a neuron node, then n values with their associated weights are popped from the stack and the result of computation with the weight associated with the neuron node is pushed onto the stack. If the current node is a recurrent jumper node, then the last value of the neuron node whose global identification number is the same as that of the jumper node is obtained. Then the value obtained with the weight associated with the jumper node is pushed onto the stack. If the current node is a forward jumper node, the sub-linear genome (sub-network) starting from a neuron whose global identification number is the same as that of the forward jumper node is copied. The response of the sub-linear genome is computed in the same way as that of the linear genome. Finally, the result of computation with the weight associated with the forward jumper node is pushed onto the stack. After traversing the genome from right to left completely, the resulting values are popped from the stack. The number of the resulting values is the same as the number of outputs of the neural network encoded by the linear genome. An example of evaluating the linear genome is shown in figure 3.

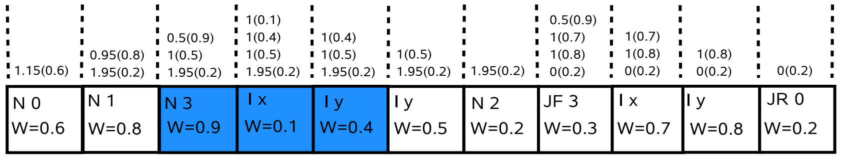
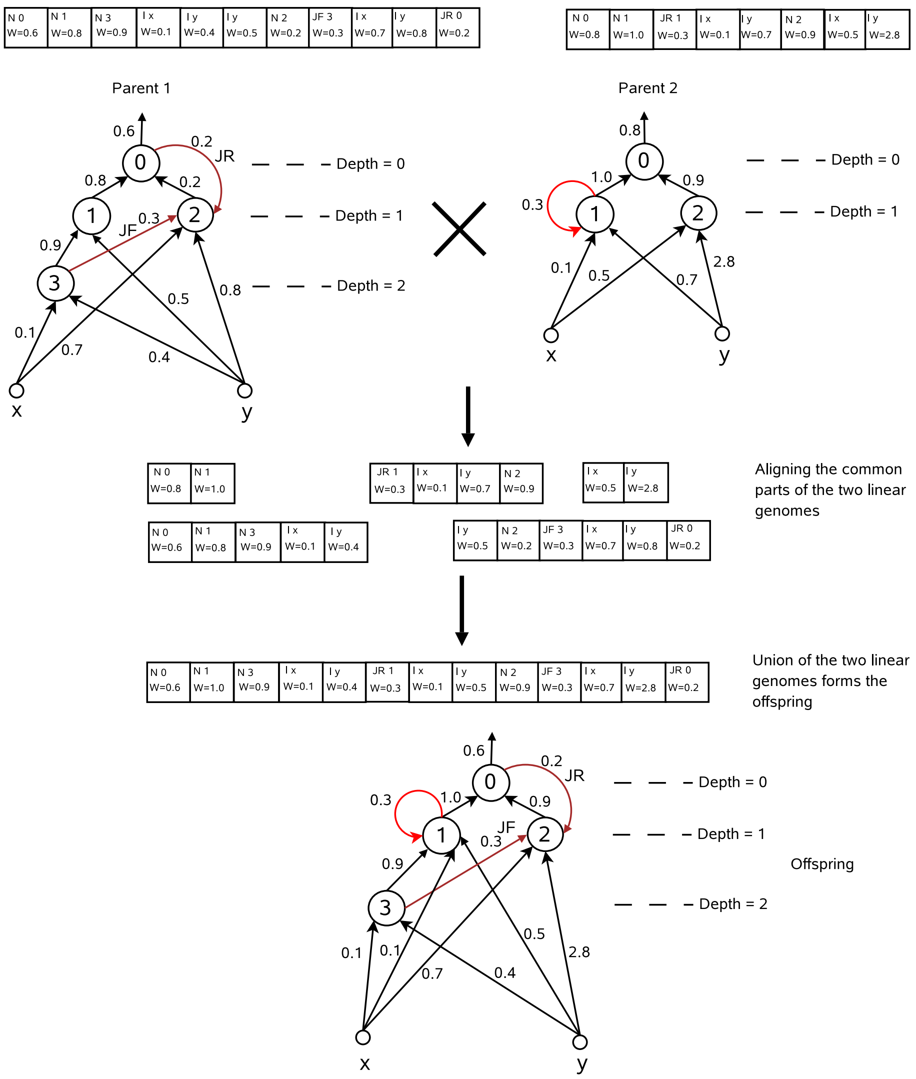


Figure 3. An example of evaluating a linear genome without decoding the neural network encoded by it. The linear genome encodes the neural network shown in figure 2. For this example, the current values of the inputs to the neural network, x and y, are both set to 1. In the example, all neurons have a linear activation function of the form $z = a$, where a is the weighted linear combination of the inputs to a neuron. The overlapped numbers above the linear genome show the status of the stack after computing the output of a node. The numbers in brackets are the weights associated with the nodes.

The presented linear genome is *complete* in that it can represent any type of neural network. It is also a *compact* encoding of neural networks since the length of the linear genome is the same as the number of synaptic weights in the neural network. It is *closed* under structural mutation and under a specially designed crossover operator. An encoding scheme is said to be closed if all genotypes produced are mapped into a valid set of phenotype networks [3]. The crossover operator exploits the fact that structures originating from some initial structure have some parts in common. By aligning the common parts of two randomly selected structures, it is possible to generate a third structure which contains the common and disjoint parts of the two mother structures. This type of crossover is introduced by Stanley [11]. An example of the crossover operator under which the linear genome is closed is shown in figure 4.

If one assigns integer values to the nodes of a linear genome such that the integer values show the difference between the number of outputs and number of inputs to the nodes, one obtains the following rules useful in the evolution of the neural controllers. The first is that the sum of integer values is the same as the number of outputs of the neural controller encoded by the linear genome. The second is that a sub-network (sub-linear genome) is a collection of nodes starting from a neuron node and ending at a node where the sum of integer values assigned to the nodes between and including the start neuron node and the end node is *one*. This property of the linear genome makes it important in the design of evolutionary methods for hierarchical and modular networks. Figure 5 illustrates the concept.

The main search operators in EANT are the structural mutation, parametric mutation and crossover operator. The structural mutation adds or removes a forward or a recurrent jumper connection between neurons, or adds a new sub-network to the linear genome. It does not remove sub-networks since removing sub-networks causes a tremendous loss of the performance of the neural network. The structural mutation operates only on neuron



N 0 W=0.6	N 1 W=0.8	N 3 W=0.9	I x W=0.1	I y W=0.4	I y W=0.5	N 2 W=0.2	JF 3 W=0.3	I x W=0.7	I y W=0.8	JR 0 W=0.2

N 0 W=0.8	N 1 W=1.0	JR 1 W=0.3	I x W=0.1	I y W=0.7	N 2 W=0.9	I x W=0.5	I y W=2.8

N 0 W=0.8	N 1 W=1.0

JR 1 W=0.3	I x W=0.1	I y W=0.7	N 2 W=0.9

I x W=0.5	I y W=2.8

N 0 W=0.6	N 1 W=0.8	N 3 W=0.9	I x W=0.1	I y W=0.4

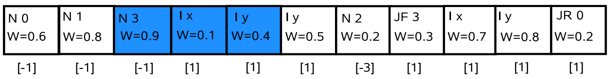
I y W=0.5	N 2 W=0.2	JF 3 W=0.3	I x W=0.7	I y W=0.8	JR 0 W=0.2

N 0 W=0.6	N 1 W=1.0	N 3 W=0.9	I x W=0.1	I y W=0.4	JR 1 W=0.3	I x W=0.1	I y W=0.5	N 2 W=0.9	JF 3 W=0.3	I x W=0.7	I y W=2.8	JR 0 W=0.2

Figure 4. Performing crossover between two linear genomes. The genetic encoding is closed under this type of crossover operator since the resulting linear genome maps to a valid phenotype network. The weights of the nodes of the resulting linear genomes are inherited randomly from both parents.

nodes. The weights of a newly acquired topology are initialized to zero so as not to disturb the performance of the network. The parametric mutation is accomplished by perturbing the weights of the networks according to the uncorrelated mutation in evolution strategy or evolutionary programming. Figure 6 shows an example of structural mutation where a neuron node lost connection to an input and received a self-recurrent connection.

Since a linear genome is equivalent to a tree based program, the initial structures are generated using either the grow or full method [1]. The initial complexity of the neural structures is determined by the domain expert and is specified by the maximum depth that



N 0 W=0.6	N 1 W=0.8	N 3 W=0.9	I x W=0.1	I y W=0.4	I y W=0.5	N 2 W=0.2	JF 3 W=0.3	I x W=0.7	I y W=0.8	JR 0 W=0.2
[-1]	[-1]	[-1]	[1]	[1]	[1]	[-3]	[1]	[1]	[1]	[1]

Figure 5. An example of the use of assigning integer values to the nodes of the linear genome. The linear genome encodes the neural network shown in figure 2. The numbers in the square brackets below the linear genome show the integer values assigned to the nodes of the linear genome. Note that the sum of the integer values is *one* showing that the neural network encoded by the linear genome has only *one* output. The shaded nodes form a sub-network. Note also that the sum of the integer values assigned to a sub-network is always *one*.

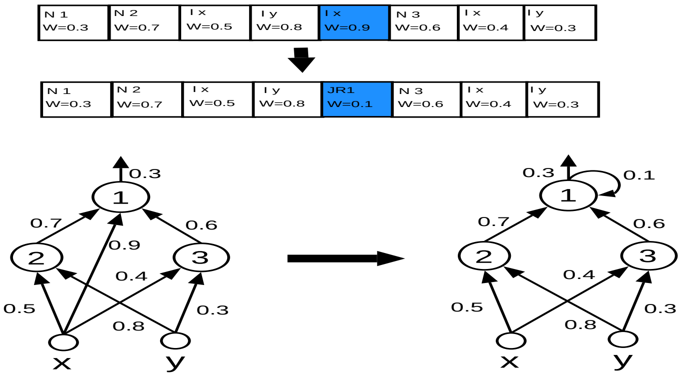


Figure 6. An example of structural mutation. Note that the structural mutation deleted the input connection to N1 and added a self-recurrent connection to it.

can be assumed by the initial structures. The depth of a neuron node in a linear genome is the minimal number of neuron nodes that must be traversed to get from the output neuron to the neuron node, where the output neuron and the neuron node lie within the same sub-network that starts from the output neuron. The evolution of neural networks starts with exploitation of structures that are already in the system. By exploitation, we mean optimization of the weights of the structures. This is accomplished by an evolutionary process that occurs at smaller time-scale. The evolutionary process at smaller time-scale uses parametric mutation and recombination operators as a search operator. An example of the exploitation process is shown in figure 7. Exploration of structures is done through structural mutation and crossover operator. The structural selection operator that occurs at larger time-scale selects the first half of the structures (species) to form the next generation. In order to protect the structural innovations or discoveries of the evolution, young structures that are less than few generations old with respect to the larger time-scale are carried over along the evolution regardless of the results of the selection operator. New structures that are introduced through structural mutation and which are better according to the fitness evaluations survive and continue to exist. Since sub-networks that are introduced are not removed, there is a gradual increase in the number of structures and their complexity along the evolution path. This allows EANT to search for a solution starting from a neural network with minimum structural complexity specified by the domain expert. The search stops when a neural network with the necessary optimal structure that solves a given task is obtained.

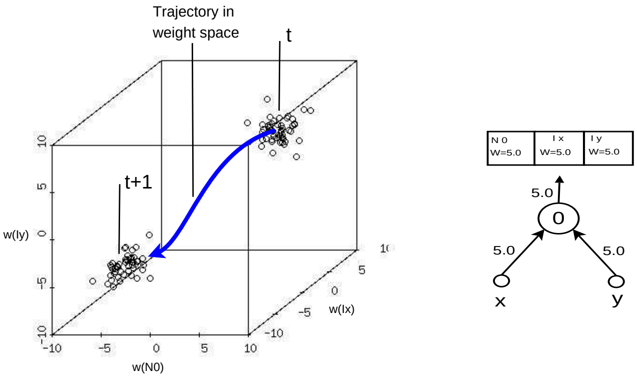


Figure 7. The weight trajectory of the linear genome shown at the right side while it is being exploited. The quantities t and $t+1$ are time units with respect to the larger time-scale. The weights of the existing structures are optimized between two consecutive time units with respect to the larger time-scale. The point clouds at t and $t+1$ show populations of individuals from the same species.

3. Learning to Move Forward

The crawling robotic insect introduced by Kimura and Kobayashi [6] is used for this experiment. It is used for reinforcement learning task where the agents learns to move forward as fast as possible through interaction with the environment. The crawling robotic insect has one arm having two joints where the joints are controlled by two servo motors. It has also a touch sensor which detects whether the tip of the arm is touching the ground or not. The schematic diagram of the robot is shown in figure 8.

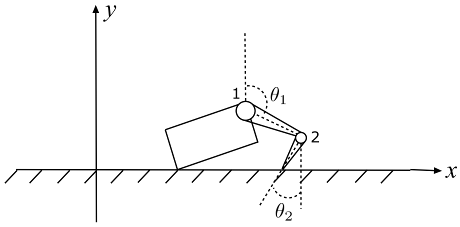


Figure 8. The crawling robotic insect. The robot has one arm with two joints and a touch sensor for detecting whether the tip of the arm is touching the ground or not.

The robot has bounded continuous and discrete state variables. The continuous state variables are the joint angles and the discrete state variable is the state of the touch sensor. The controller observes the joint angles and the state of the touch sensor. Depending on the state it perceives, the controller is expected to change the angles of the joints appropriately so that the robot can move forward as fast as possible. The first joint angle θ_1 is bounded between 55° and 94°, and the second joint angle θ_2 lies in the range $[-34^\circ, 135^\circ]$. For both of the joints, the angles are measured from the vertical as shown in figure 8. The angle ranges are chosen so that they are equivalent to the angle ranges chosen by Kimura and Kobayashi. They measured the first joint angle from the horizontal and the second joint angle from the first link. The touch sensor ϕ takes the value 0 for non-touch state and 1 for touch state.

Let the coordinates of the first and the second joints be (x_0, y_0) and (x_1, y_1), respectively and let the coordinate of the tip of the arm be (x_2, y_2). The state of the robot at each time step $t = 0, 1, \ldots$ is given by $s_t = (x_0, y_0, x_2, y_2, \theta_1, \theta_2, \phi)$. Since the coordinate (x_1, y_1) can be calculated given a state s, it is not listed in the definition of the state of the robot. The state transition of the system is governed by equations (1) and (2). If the tip of the arm is not touching the ground ($\phi(t) = 0$), then the state transition equation is given by

$$\begin{array}{l} \theta_1(t+1) = \theta_1(t) + \delta_1 \\ \theta_2(t+1) = \theta_2(t) + \delta_2 \\ x_0(t+1) = x_0(t) \\ y_0(t+1) = y_0(t) \\ x_2(t+1) = x_0(t+1) + l_1 \sin\theta_1(t+1) + l_2 \sin\theta_2(t+1) \\ y_2(t+1) = y_0(t+1) + l_1 \cos\theta_1(t+1) - l_2 \cos\theta_2(t+1) \end{array}, \tag{1}$$

and if the tip of the arm is touching the ground ($\phi(t) = 1$), then the state transition equation takes the form

$$\begin{array}{l} \theta_1(t+1) = \theta_1(t) + \delta_1 \\ \theta_2(t+1) = \theta_2(t) + \delta_2 \\ x_2(t+1) = x_2(t) \\ y_2(t+1) = y_2(t) \\ x_0(t+1) = x_2(t+1) - l_2 \sin\theta_2(t+1) - l_1 \sin\theta_1(t+1) \\ y_0(t+1) = l_2 \cos\theta_2(t+1) - l_1 \cos\theta_1(t+1) \end{array}. \tag{2}$$

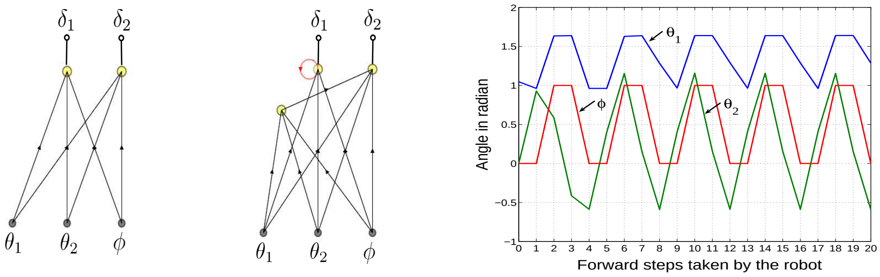
The quantities δ_1 and δ_2 are the outputs of the neural controller to be designed, and l_1 and l_2 are the lengths of the first and the second link. The first link is between the first joint and the second joint while the second link is between the second joint and the tip of the arm. For the experiment, $l_1 = 34$ cm and $l_2 = 20$ cm are chosen. The first joint is located at right upper corner of the rectangular body of the robotic insect which has a height of 18 cm and width of 32 cm. A trial contains 50 time steps and at the beginning of a trial the robot is placed at the origin. The fitness function used to evaluate a neural controller is given by

$$f = \frac{1}{N} \sum_{t=1}^{N} (x_0(t) - x_0(t-1)), \tag{3}$$

where the difference $x_0(t) - x_0(t-1)$ is the velocity of the system at time t in the direction of the $x-$axis and f is the average velocity of the robot for a trial. The number of time steps used per trial is represented by N.

Tsuchiya et al. [13] applied their policy learning by genetic algorithm using importance sampling (GA-IS) for learning to move forward. They defined a three dimensional vector $X = (x_1, x_2, x_3)$ for representing the state space. The dimensions of the state space is made up of the joint angles and the state of the touch sensor. The policy used in their experiment is a 7 dimensional feature vector $F = [x_1, x_2, x_3, x_4\,(= 1 - x_1)\,, x_5(= 1 - x_2), x_6(= 1 - x_3), 1.0]$. A weight vector $\Theta = (\theta_{1,i}, \theta_{2,i}, \theta_{3,i}, \theta_{4,i}, \theta_{5,i}, \theta_{6,i}, \theta_{7,i})$ is used to select the action $a_i(t)$ from normal distribution with mean value $\mu_i = 1/(1 + \exp(-\sum_{k=1}^{6} \theta_{k,i} x_k))$ and standard deviation $\sigma_i = 1/(1 + \exp(-\theta_{7,i})) + 0.1$. If the selected action is out of range then it is resampled. The number of the policy parameters is 14 and hence the search space for the genetic algorithm has 14 dimensions.

In our experiment, the structure shown in figure 9 (a) containing two output neurons connected to three input nodes (θ_1, θ_2, ϕ) is used as the initial controller. The best controller shown in figure 9 (b) is found after running EANT. Note that the best controller is more complex than the initial structure. Figure 9 (c) shows the waveforms of the joint angles and the touch sensor for the first 20 time steps as the robot moves forward and being controlled by the best controller.



(a) (b) (c)

Figure 9. Learning to move forward. (a) The initial structure. (b) The best controller found by our algorithm that enables the robot to move forward. (c) The waveforms of the joint angles and the touch sensor as the robot moves forward.

The method introduced by Tsuchya et al. (GA-IS) needed on the average 10000 interactions with the environment for learning to move forward. We have run EANT 50 times and obtained on the average 3520 intractions for learning the task. As compared to the GA-IS, EANT has reduced the number of interactions with the environment necessary to learn to move forward. The reduction in the number of interactions is due to the direct search for an optimal policy in the space of policies, starting minimally and increasing the complexity of the neural network that represents the policy.

4. Conclusion and Outlook

An evolutionary reinforcement learning system that is suitable for learning through interaction is presented. The system exploits both types of adaptations: namely evolutionary adaptation and adaptation through learning. It starts with networks of minimal structures and complexifies them along the evolution path.

The method introduces a compact genetic encoding that enables one to evaluate the neural network encoded by it without some type of ontological process of transforming the genotype into phenotype. In addition to this, a meta-level evolutionary process is introduced that is suitable to explore new structures incrementally and exploit the existing ones.

The system can be extended to handle the evolution of hierarchical structures and modular networks. In addition to this, ways of describing the search space as well as the final resultant networks can be included in order to direct the evolution.

Acknowledgments

This work is sponsored by the German Academic Exchange Service (DAAD) under grant code R-413-A-01-46029 and the COSPAL project under the EC contract no. FP6-2003-IST-2004176, which are duly acknowledged.

References

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, San Francisco, CA, 1998.

[2] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Massachusetts, London, 1975.

[3] J. Jung and J. Reggia. A descriptive encoding language for evolving modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 519–530. Springer-Verlag, 2004.

[4] Y. Kassahun and G. Sommer. Improving learning and adaptation capability of agents. In *Proceedings of 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 472–481, Amsterdam, November 2004.

[5] Y. Kassahun and G. Sommer. Evolution of neural networks through incremental acquisition of neural structures. Technical Report 0508, Institute of Computer Science and Applied Mathematics, Christian-Albrechts University, Kiel, Germany, June 2005.

[6] H. Kimura and S. Kobayashi. Reinforcement learning for locomotion of a two-linked robot arm. In *Proceedings of the 6th European Workshop on Learning Robots*, pages 144–153, 1997.

[7] J. R. Koza. Genetic programming: A paradigm for genetically breeding population of computer programs to solve problems. Technical Report STAN-CS-90-1314, Computer Science Department, Standford University, Stanford, CA, USA, 1990.

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

[9] R. Pfeifer and C. Scheier. *Understanding Intelligence.* The MIT Press, Massachusetts, London, 1999.

[10] H. P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[11] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA, August 2004.

[12] R. Sutton and A. Barto. *Reinforcement Learning. An Introduction*. The MIT Press, Massachusetts, London, 1998.

[13] C. Tsuchiya, H. Kimura, and S. Kobayashi. Policy learning by GA using importance sampling. In *Proceedings of 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 385–394, Amsterdam, November 2004.