Evolving Modular Fast-Weight Networks for Control

Faustino Gomez[1] and Jürgen Schmidhuber[1,2]

[1] IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland
[2] TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany
{tino, juergen}@idsia.ch

Abstract. In practice, almost all control systems in use today implement some form of linear control. However, there are many tasks for which conventional control engineering methods are not directly applicable because there is not enough information about how the system should be controlled (i.e. reinforcement learning problems). In this paper, we explore an approach to such problems that evolves *fast-weight* neural networks. These networks, although capable of implementing arbitrary non-linear mappings, can more easily exploit the piecewise linearity inherent in most systems, in order to produce simpler and more comprehensible controllers. The method is tested on 2D mobile robot version of the pole balancing task where the controller must learn to switch between two operating modes, one using a single pole and the other using a jointed pole version that has not before been solved.

1 Introduction

All real-world systems are non-linear to some degree, yet almost all control systems in operation today employ some variant of linear feedback control. The wide applicability of linear methods relies on the fact that most non-linear systems of interest are either nearly linear around some useful operating point or can be decomposed into multiple linear operating regions. Methods such as *gain-scheduling* provide powerful tools to control such systems [1]: first a linear model is built for each operating mode, and then a linear controller (e.g. PID) is designed with parameters (i.e. gains) that are switched by a *scheduler* when the system transitions from one mode to another.

Gain scheduling works well when the mode of the system is observable, and, like all classical approaches, when the appropriate type of strategy is known *a priori*. For very complex tasks, such as those encountered in robotics, the designer often does not know what action should be taken in each system state. One method for solving control tasks under these more general conditions is neuroevolution [2] where a genetic algorithm is used to search the space of neural network controllers by repeatedly recombining the best performing candidates according to the principle of natural selection.

Artificial neural networks can potentially implement *global* non-linear controllers, but ensuring their stability and analyzing their behavior is difficult [3].

W. Duch et al. (Eds.): ICANN 2005, LNCS 3697, pp. 383–389, 2005.

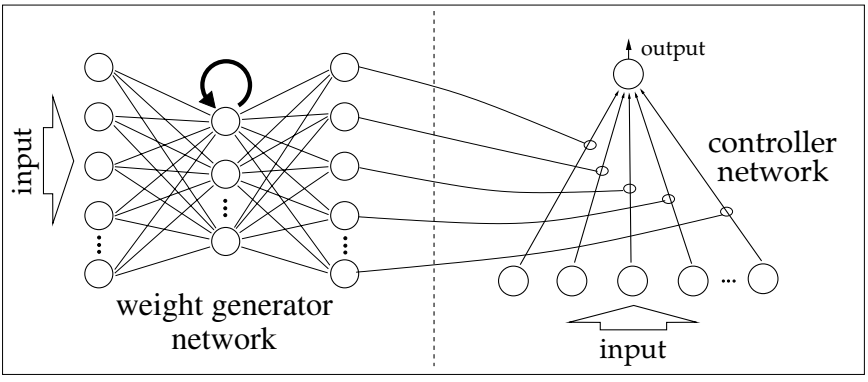


Fig. 1. Fast-Weight Network Module. The figure shows the two components of a Fast-Weight Module. On the left is the *generator* network that is evolved by the GA. This recurrent network receives input from the environment and outputs a set of weight values for the single-layer *controller* network at right. The controller network also receives input from the environment and outputs the control action.

This paper explores a method for evolving a special kind of *fast-weight* neural network that can potentially provide simpler automatically designed controllers. The networks consist of separate modules containing a recurrent neural network that generates weights for a linear controller.

The next section describes the fast-weight network architecture. Section 3 describes the neuroevolution method, Hierarchical Enforced SubPopulation (H-ESP) that is used to evolve the fast-weight networks. Section 4 presents our experimental results in applying the method to a two-mode robot pole balancing task, and section 5 concludes with a brief discussion of the results.

2 Modular Fast-Weight Networks

When neural networks are used to solve tasks in which the output depends on a history of inputs, they usually contain recurrent connections that feed back previous activations. Temporal information is encoded in the form of internal activation patterns (i.e. state) generated by propagating external inputs and previous activations through a fixed set of weights. Another possibility is to have dynamic weights or *fast weights* that can change in value over time. The little work that has been done using this concept has either used fast-weights as a mechanism to provide more robust associative memories [4], or to reduce network learning complexity [5]. Here we use the idea of fast-weights to generate controllers capable of switching easily between linear functions.

Networks are composed of a separate fast-weight module for each output unit. Each module consists of a recurrent *generating network* and a single-layer, feedforward *controller network* (figure 1). The output o_m of module m is:

$$o_m = \delta\left(\sum_{k=1}^{I} x_k \widehat{w}_{km}\right) \tag{1}$$

$$\widehat{w}_{km} = \sum_{j=1}^{H}\left(w_{jk}\ \delta\left(\sum_{i=1}^{I} x_i w_{ij} + \sum_{h=1}^{H} a_h w_{hj}\right)\right) \tag{2}$$

where $x \in \Re^I$ is the external input, $a \in \Re^H$ is the hidden layer activation from the previous time step, w_{ij} is the weight from unit i to unit j in the generating network, $\widehat{w}_{ij}$ is the weight from i to j in the controller network, and δ is the sigmoid function. Equation 1 computes the output of the controller network *after* the generating network has produced the I weights according to equation 2.

This network architecture is theoretically no more powerful than a standard fully recurrent network. The underlying intuition behind its design is that such an architecture will bias the search toward controllers that are potentially simpler and better suited to non-stationary environments characterized by transitions between operating modes. Although a module can generate weights that are a non-linear function of the entire history of inputs, it can easily implement a linear controller, if it is all that is required, by having the generating network output constant values.

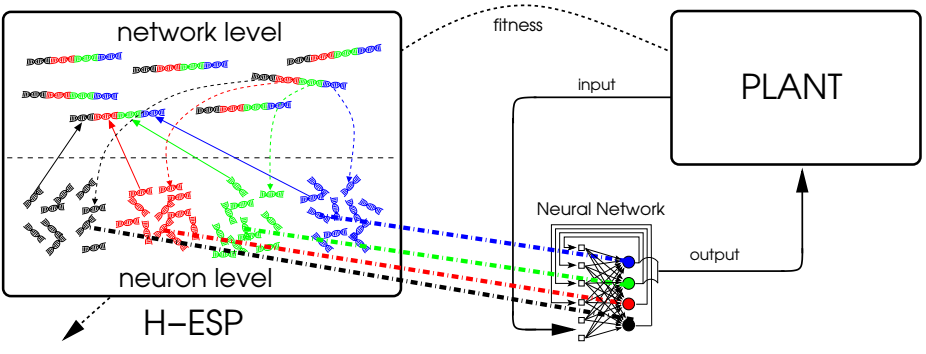


Fig. 2. H-ESP. Evolution occurs at both the level of neurons and of networks. The neuron level (L1) consists of multiple subpopulations of neurons, shown here in different colors. The network level (L2) consists of complete network representations that have either migrated up from below or have been created by recombining networks. During evolution, networks are evaluated in two possible ways: from L2 directly, and from L1 by randomly selecting a neuron from each subpopulation and combining them into a complete network. The dashed lines from the neuron level to the network being evaluated indicate a network formed in this manner. A network from L2 that has higher fitness than any network formed so far in L1, has its neurons copied into their corresponding subpopulations in L1 (shown with the dashed arrows from L2 to L1). A network form in L1 that has higher fitness than the worst L2 network is copied into L2 (the solid arrows from L1 to L2). In this way, the two levels supply each other with new genetic material with which to search in their respective weight spaces.

3 Hierarchical Enforced Subpopulations

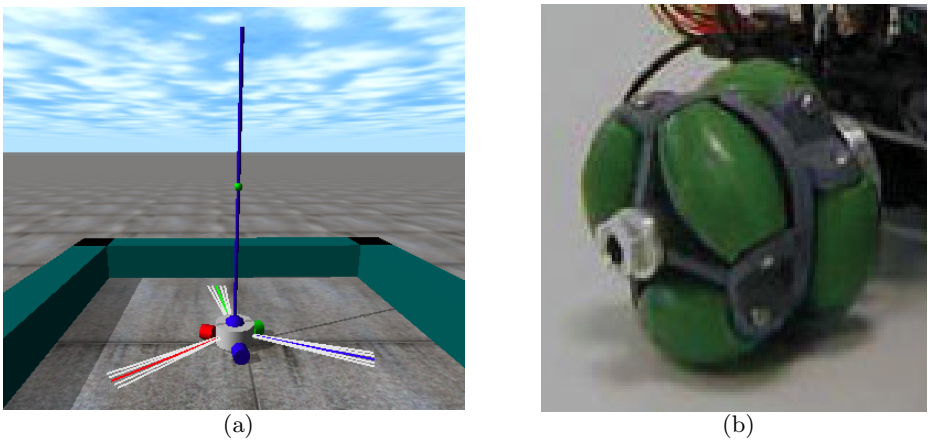
Fast-weight networks are evolved using a method introduced in [6] called Hierarchical Enforced SubPopulations (H-ESP). H-ESP searches the space of recurrent networks by evolving at two levels in tandem: the level of network components or *neurons*, and the level of full networks. The neuron level (i.e. plain ESP) searches the space of networks indirectly by sampling the possible networks that can be constructed from the subpopulations of neurons. Network evaluations provide a fitness statistic that is used to produce better neurons that can eventually be combined to form a successful network. Figure 2 shows the basic operation of the algorithm (see [6] for further details).

The network level provides a repository or "hall of fame" of the best networks found so far by the neuron level, and allows H-ESP to search within the space of highly fit neuron combinations in a way that is not possible at the neuron level because it constructs networks at random.

To evolve fast-weight networks each neuron encodes the input, recurrent, and output weights of one of the units of a generating network.

4 Experiments

To evaluate approach, we evolved controllers for a simulated version of the three-wheeled Robertino mobile robot (figure 3a). Each wheel can slide along its rotational axis using six small sub-wheels (figure 3b). This *holonomic drive* enables the robot to change direction without having to rotate. On top of the robot is a vertical pole that is attached to the chassis with a ball joint. The pole can be either a single rigid rod or two rods, one on top of the other, with a ball joint



(a) (b)

Fig. 3. Robertino with pole. (a) A snapshot from the Robertino ODE simulation showing the robot in the center of a walled arena. (b) Robertino wheel. The small sub-wheels allow the Robertino to slide as well as roll.

connecting them. The objective is to balance the both types of poles by applying a torque to each of the three wheels. Balancing each type of pole requires a different strategy. Both systems are nearly linear around their unstable equilibrium points (i.e. poles in vertical position), but when the angle of the pole(s) increases they become non-linear, more so in the case of the jointed pole.

Note that unlike the 2-dimensional version of the classic pole balancer [7], the system cannot be controlled by solving the 1-dimensional case and then using two copies of this controller, one for each principle axis. Because the robot can rotate around its vertical axis using 3 wheels spaced 120° from each other, this simple symmetry cannot be exploited. To move in a given direction, the velocity of all 3 wheels must be correctly modulated.

H-ESP was used to evolve networks consisting of three fast-weight modules, one for each wheel as shown in figure 4. The weight generating network of each module had 5 hidden units and 8 inputs scaled to the range [-1.0,1.0]: 3 proximity sensors, the angle of the lower pole in the x-axis θ_x^l and y-axis θ_y^l, the angles for the upper pole θ_x^u, θ_y^u, and the rotation of the chassis; all angles were measured in absolute (global) coordinates. For the single pole mode $\theta_{x,y}^u$ were set to zero. The neuron level subpopulations consisted of 200 neurons, and the network level population of 100 networks. The robot was simulated using the Open Dynamics Engine (www.ode.org) with a 0.01 second integration time.

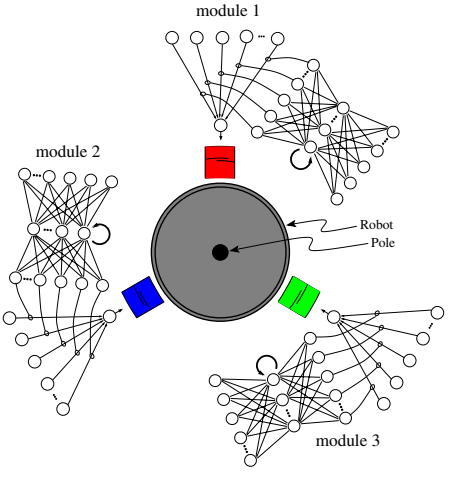


Fig. 4. Control Architecture for the Robertino robot. The Robertino (overhead view) is controlled by of three fast-weight modules, one for each wheel. At each time step, the generator networks produce weights for and activate their respective controller network.

During evaluation the controllers were tested in two trials: one with a single pole of 1.0 meter in length, and one with a jointed pole with two 0.5 meter segments. Each trial starts with the robot sitting in the center of a 1.5×1.5 meter walled arena (see figure 3a) with the pole(s) leaning 6° (0.5°) from vertical in the x direction. Every 0.04 seconds (i.e. 25Hz control frequency) each module outputs the desired angular velocity for its corresponding wheel $[-3.6\pi, 3.6\pi]$, until the pole angle(s) exceed 36°. The fitness of the controller was the number of time steps the pole(s) could be balanced in the shorter of the two trials. The task was considered solved with a fitness of 10 thousand. In order to solve the task, the controller must determine which mode it is in and apply the appropriate strategy for that mode.

4.1 Results

Figure 5 shows the controller network weight values produced during the successful operation of a typical controller for the first 100 time steps of operation, in each mode. The solid curve is for the single pole and the dotted curve is for the jointed pole. For most weights, the difference between the modes occurs at the beginning of the trial when the pole angles are relatively large, and the controller must employ a different strategy to bring the pole(s) into the linear region. Other weights, specifically those in module 3, quickly reach a constant value for the jointed pole mode, and then transition to the same value used for the single pole after about 80 time steps, by which time the jointed pole has been stabilized.

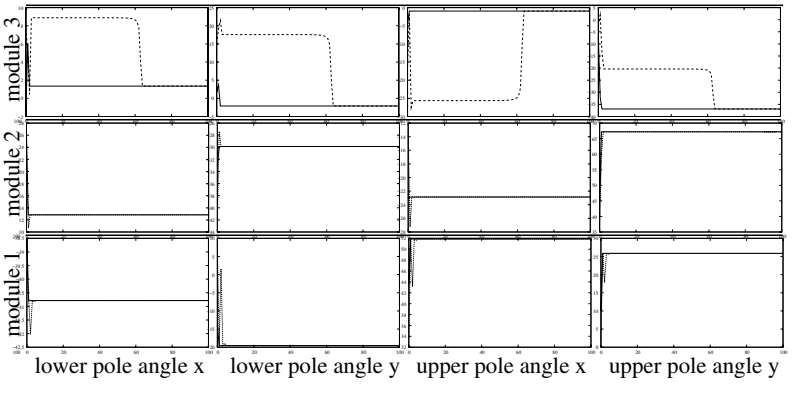


Fig. 5. Fast-weight values during control. Each plot shows the weight value for one of the inputs for the two modes. Each row corresponds to one of the 3 modules. The solid curve is for the single pole, the dotted curve is for the jointed pole.

5 Discussion and Conclusion

The experiments show that fast-weight networks can be evolved to produce relatively simple controllers. The weights produced by the generating networks implement almost piecewise linear controllers. While simpler architectures such as fully recurrent networks can solve each case, we were unable to do so for the two mode problem, and even the jointed pole version by itself could not be solved reliably. Furthermore, with such networks it is often difficult to understand the strategy being implemented. Using fast-weight networks, each period of constant weight values is a linear controller that can be "cut away" from its generating network during testing, leaving a set of simple linear filters that are more amenable to formal control theory analysis.

Future work will apply this approach to bipedal robot walking where it might be possible to implemented controllers for different, potentially non-linear, gait modes by using fast-weight networks.

Acknowledgments

This research was partially funded by CSEM Alpnach and the EU MindRaces project: FP6 511931. We would like to thank Frank Pasemann and Keyan Mahmoud Ghazi-Zahedi of Fraunhofer AIS for the Robertino ODE code.

References

1. Rugh, W., Shamma, J.: A survey of research on gain-scheduling. Automatica (2000) 1401–1425
2. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87** (1999) 1423–1447
3. Kretchmar, R.M.: A Synthesis of Reinforcement Learning and Robust Control Theory. PhD thesis, Department of Computer Science, Colorado State University, Fort Collins, Colorado (2000)
4. Hinton, G.E., Plaut, D.C.: Using fast weights to deblur old memories. In: Proceedings of the Ninth Annual Cognitive Science Society Conference, Hillsdale, NJ, Lawrence Erlbaum Associates (1987) 177–186
5. Schmidhuber, J.: Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computation **4** (1992) 131–139
6. Gomez, F.J., Schmidhuber, J.: Co-evolving recurrent neurons learn deep memory pomdps. Technical Report 17–04, IDSIA, Lugano, Switzerland (2004)
7. Gomez, F., Miikkulainen, R.: 2-D pole-balancing with recurrent evolutionary networks. In: Proceedings of the International Conference on Artificial Neural Networks, Berlin; New York, Springer-Verlag (1998) 425–430