A Robotic Scenario for Programmable Fixed-Weight Neural Networks Exhibiting Multiple Behaviors

Guglielmo Montone, Francesco Donnarumma, and Roberto Prevete

Dipartimento di Scienze Fisiche, Università di Napoli Federico II

Abstract. Artificial neural network architectures are systems which usually exhibit a *unique/special* behavior on the basis of a fixed structure expressed in terms of parameters computed by a training phase. In contrast with this approach, we present a robotic scenario in which an artificial neural network architecture, the Multiple Behavior Network (MBN), is proposed as a robotic controller in a simulated environment. MBN is composed of two Continuous-Time Recurrent Neural Networks (CTRNNs), and is organized in a hierarchial way: Interpreter Module (IM) and Program Module (PM). IM is a fixed-weight CTRNN designed in such a way to behave as an *interpreter* of the signals coming from PM, thus being able to switch among different behaviors in response to the PM output *programs*. We suggest how such an MBN architecture can be incrementally trained in order to show and even acquire new behaviors by letting PM learn new programs, and without modifying IM structure.

Keywords: Multiple behaviors, fixed-weights, CTRNN, programmability, robotics.

1 Multiple Behaviors in a Fixed-Weight Neural Network

The ability of exhibiting rapid, substantial and qualitative changes of behavior as a consequence of environmental changes is typical of human beings. It is an open question whether this kind of ability can be always explained as the result of rapid changes of brain connectivity (potentiation and/or depression of the synapses), or could also be the consequence of a computation carried on by a fixed connectivity, i.e., without potentiantion or depression of the synapses. This issue is recently discussed, for example, in [1]. In spite of this debate, Artificial Neural Network (ANN) architectures are usually developed as special purpose systems, especially when they are used to model the activity of brain neuronal areas (see for example [13]). In other words, ANN architectures are developed in such a way as to exhibit, after a possible training phase, a *unique/special* behavior in response to the input signals while preserving the structure, i.e., network parameters such as weights and biases. Understanding if and how it is possible to achieve an ANN architecture with a fixed structure able to quickly switch among different behaviors is a recent and open problem. One of the first

A. Dobnikar, U. Lotrič, and B. Šter (Eds.): ICANNGA 2011, Part I, LNCS 6593, pp. 250–259, 2011.

papers in this field is due to Yamauchi and Beer[15]. More recently the study of ANNs able to show multiple behaviors has been associated with the realization of autonomous robots driven by *biologically plausible* artificial neural networks such as Continuous-Time Recurrent Neural Networks (CTRNNs)[11,3]. Blynel and Floreano built a robot able to display learning-like abilities driven by a network in which no modifications of synaptic strengths take place. Paine and Tani realize a network hierarchically organized in two layers, the lower layer learns to execute two behaviors while the upper level by receiving an environmental input leads the lower network to select reactively the correct appropriate behavior between the two learned ones.

In a series of papers, which include some of the authors of this paper, [9,14,6] the problem of multiple behaviors in ANNs is analyzed from a different point of view. They focused on the concept of *programmability,* as defined in computer science, explored in a biologically plausible neural network. By programmability in ANNs we mean the property of a fixed-weight ANN of working as an *interpreter* capable of emulating the behavior of other networks given the appropriate codes. In [7,5] a possible architecture is presented to provide the CTRNN framework with programmability. Building up on this approach, we are proposing here a CTRNN architecture, Multiple Behaviour Network (MBN), able to control a robot in a simulated environment. In our experiment the robot explores a maze environment using sonars and a camera and shows two different behaviors on different camera inputs. In particular, the robot can behave as a right-follower, following the wall on his right at every crossroads, or as a left-follower, following the wall on his left. The MBN is hierarchically organized into two CTRNN modules, the lower one is *programmable* and it is able to emulate the behavior of either a network that achieves a right-follower or a network that achieves a left-follower through the application of the appropriate codes for these two behaviors to special input lines, the *auxiliary inputs*, by the higher module of MBN.

In the next section we describe the architecture used to provide CTRNNs with programmability. Then in Section 3 we describe the MBN realization and the robotic scenario in which we test it. We conclude with a short discussion in Section 4.

2 A Programmable Fixed-Weight CTRNN

CTRNNs are networks of biologically inspired neurons described by the following general equations [10,2]:

$$\frac{dy_i}{dt} = \frac{1}{\tau_i}\Big(- y_i + \sum_{j=1}^{N} W_{ij}\sigma(y_j) + \sum_{k=1}^{L} WE_{ik}I_k\Big) \tag{1}$$

In equation (1) y_i is the *membrane potential* of the i^{th} neuron. The variable τ_i is the *time constant* of the neuron. It affects the rate of activation in response to the *external sensory inputs* I_k and the signals coming from the presynaptic neurons with output $\sigma(y_j)$. The function $\sigma(x)$ is the standard logistic function.

W_{ij} is the weight of the connection between the j^{th} and the i^{th} neuron, while WE_{ik} is the weight of the connection between the external input I_k and the i^{th} neuron.

The input to each neuron, as usual, consists of the sums of products between output signals coming from other neurons over weighted connections, and the weights associated with the connections. So the behavior of a CTRNN network is grounded into the sums of the products between weights and output signals. Here, following the architecture suggested in [7,5], we "pull out" the multiplication operation by using subnetworks providing the outcome of the multiplication operation between the output and the weight. Substituting a connection with a network realizing multiplication we are able to give the weight of the connection as *auxiliary* (or *programming*) input lines in addition to standard data input lines. To clarify our approach, let us suppose having an ideal CTRNN, *mul*, able to return as output the product of two inputs $a, b \in (0, 1)$. Given a network S of N neurons and having chosen two neurons i and j linked by one connection with weight $W_{ij} \in (0, 1)$, it is possible to build an "equivalent" *Programmable Neural Network* (PNN) according to the following steps (see figure 1):

1. redirect the output of the neuron j as input of a mul,
2. set the second input of the mul to W_{ij},
3. redirect the output of the mul network as input to the neuron i.

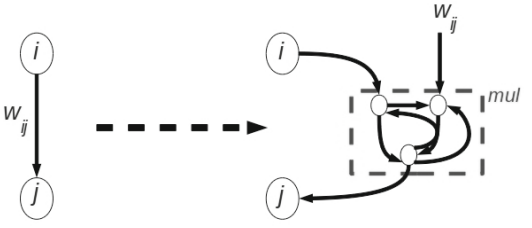

Fig. 1. The substitution which achieves a programmable connection in the target network

In the approximation that the neurons of the multiplicative networks have time constants much smaller than the time constants of the other neurons of the network, the dynamics of the constructed PNN, restricted to the neurons i and j, is identical to the original network S. As it is shown in [5], it is always possible to extend this procedure to a generic weight $W_{ij} \in (min, max)$ by rescaling the parameter W_{ij} with the transformation

$$p_{ij} = \frac{(W_{ij} - min)}{(max - min)} \qquad p_{ij} \in (0, 1) \tag{2}$$

setting the weight of the connection between the *mul* output and neuron j equal to $max - min$, and adding a connection between the neurons j and i with weight equal to min. Consequently, the weight W_{ij} is encoded by p_{ij}.

This process can be repeated for every connections of the network S. The resulting network will be a PNN. It will be able to emulate the behavior of any other CTRNN of N neurons when receiving its encoded weights as auxiliary inputs.

2.1 A Multiplicative CTRNN

In the previous section the substitution discussed involves a suitable multiplicative CTRNN, able of performing a multiplication of its input values. In order to effectively perform this subsitution we obtained an approximation of an ideal multiplicative network training a 3-neuron CTRNN with the Backpropagation Through Time (BPTT) learning algorithm [12]. We built a training set of 100 input-output pairs. The input is given by (x_1, x_2) equally spaced in $(0, 1) \times (0, 1)$. The output is the correspondent target value $t = x_1 \cdot x_2 \in (0, 1)$. Each pair (x_1, x_2) is given as input to the network for a time $T = 10 \cdot \tau_m$, where τ_m is the common value of the time constant of the three neurons of the network. After this period the output of the third neuron was recorded. During the BPTT training we initialize the values of the weights of the network in a range $[-10, 10]$. The validation set of the network was constituted of 100 randomly chosen input-target pairs. The network tested on this validation set associates to the couple in input the correct value of the relative output with an error never bigger than the 5% of the target value. This approximation could be improved by considering networks with more than three neurons, however this error was sufficient in order to successfully perform the experiments in the next section.

3 A Programmable Neural Network for Robot Control

We developed a CTRNN architecture, Multiple Behavior Network (MBN), able to govern a simulated robot equipped with sonars and a camera in a maze environment according to two different behaviors. Along the corridors one or two-colors tiles are located. The robot shifts from behaving like a right-follower to behaving like a left-follower when the camera sees a one-color or a two-colors tile respectively. The network is composed of two modules organized in a hierarchical way: the lower module, Interpreter Module (IM), is implemented by a PNN, and it behaves as an interpreter, defining the same mapping among sonars and motor controls as the one achieved by a right or a left-follower when receiving their codes/programs as auxiliary inputs. The upper level, Program Module (PM), has to pass the code to the lower one. PM implements a functional relation between the two kinds of camera inputs and the two programs of the right and the left-follower (see figure 2).

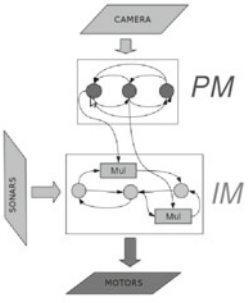

Fig. 2. The Multiple Behavior Network (MBN) that governs the robot

3.1 The Robotic Scenario

Robot simulations were carried out using the open source software **Player-Stage** to simulate a *Pioneer 3DX* robot. The robot is equipped with *10 sonars* placed on the frontal and the lateral parts of the robot ($s1$, ..., $s10$ in Figure 4) and a camera placed in the middle of the robot. The output of a *blobfinder* software, which is able to reveal the number of spots the camera sees, is given as input of the CTRNN controlling. The robot can be governed choosing its angular and linear velocity. The environment is constituted of corridors of different length and three times wide the robot size, which crosses with angles of 90^o. The steps through the realization of MBN controlling the robot are mainly three. The first step is the building of two CTRNNs, RF and LF, capable of behaving as a right and a left-follower, respectively. The knowledge of the structure of the networks RF and LF is fundamental for the second step, building the interpreter network IM. In fact, following the procedure proposed in 2, we will develop the interpreter network IM on the basis of RF and LF structure, considering that the connections that changes their values in passing from RF to LF are the connections that will become programmable in the interpreter network, i.e., the connections to which the procedure decribed in 2 will be applied.

In the last step, we realize and train the MBN, made up of two modules, consisting in the intepreter module IM and the program module PM. Using an evolutive algorithm, during the training phase, we compare the behaviors of robots driven by different MBN networks. Each MBN has the same lower level IM and differs in the upper level network PM. Consequently, the best MBN will be constituted by the fixed interpreter network IM and the network PM capable of associating to the two camera inputs the programs/codes of the most suitable right and left-follower behaviors.

The Right-Follower and the Left-Follower. As we explained, the first step in the building of the MBN is the construction of two CTRNNs, RF and LF able to behave as right and left-follower, respectively. We tried different sized networks, but the resulting ones are CTRNNs composed both of three neurons.

They receive three inputs that are respectively the weighted sum of three sonars facing right, three facing left and two frontal sonars as in the following equations:

$$\begin{aligned} I_1 =& 0.2 \cdot S_2 + 0.4 \cdot S_3 + 0.4 \cdot S_4; \\ I_2 =& 0.2 \cdot S_9 + 0.4 \cdot S_8 + 0.4 \cdot S_7; \\ I_3 =& 0.5 \cdot S_5 + 0.5 \cdot S_6; \end{aligned} \tag{3}$$

The first, the second and the third neurons respectively receive the first, the second and the third input on a weighted connection. The output of the network is constituted of the output of the second and the third neuron. In particular the output of the third neuron determines the linear speed of the robot, while the output of the second neuron controls the angular velocity. The neurons of the two networks share the same value of the characteristic time τ, that is of an order of magnitude bigger than the characteristic time of the multiplicative networks τ_m. We evolve the weights of the networks RF and LF using an algorithm of Differential Evolution (DE) [4], a population-based evolutionary algorithm. We initialized a population of 18 elements controlled by networks with weights randomly chosen in the range $[-100, 100]$. Every controller obtained is evaluated with a fitness function F while performing the task of behaving as a right or as a left follower. A new population is obtained using the best element of the previous population. In our training we used a crossover coefficient ($CR \in [0, 1]$) of 0.2 and a step-size coefficient ($ST \in [0, 1]$) of 0.85, this means that our algorithm builds the next generation preserving the architecture of the best element of the previous generation (the value of the crossover coefficient is low), but even preserving the variance of the previous generation (the value of the step-size coefficient is high). The task used to evaluate the robot is structured as follows. There are 7 kinds of crossroads present in the maze (see Figure 4). Each robot is placed at the beginning of each crossroad and is free to evolve for about 20 seconds. The final evaluation of the "life" of a robot is the sum of the evaluations obtained in each of the seven distinct simulations. The fitness function that evaluates the robot in every crossroads is made of two components. The first component F_M is derived by the one proposed by [8] and consists of a reward for straight, fast movements and obstacles avoidance. This component is the same in the left and the right-follower task. The second component changes between the two trainings; in the right-follower training it rewards the robot that turns right at a crossroad (F_R), in the left-follower training it rewards the robot that turns left (F_L). In equation (4) $\overline{V}$ is the average speed of the robot, V_A is the average angular speed. S_{min} is the values of the shortest distance measured from an obstacle during the task period.

$$F_M = \overline{V} \cdot (1 - \sqrt{V_A}) \cdot S_{min} \qquad V_A \in [0, 1], S_{min} \in [0, 1]; \tag{4}$$

$$F_R = \overline{S}_1 + \overline{S}_2 - \overline{S}_9 - \overline{S}_{10} \qquad F_L = \overline{S}_9 + \overline{S}_{10} - \overline{S}_1 - \overline{S}_2. \tag{5}$$

In F_R the average measure of the left sonars over the task period is subtracted to the average measure of the right ones, the opposite happens in F_L. While

the training to obtain the right-follower was done evolving the whole weights of the network, it was not the case of the left-follower training. In fact the connections that have the same values for the left and the right-follower will not be substituted by a multiplicative network in the programmable net. So the bigger the number of connections unchanged will be the easier the structure of the programmable network will remain. To obtain a left-follower we chose some of the connections of the right-follower and changed their values by evolving them with the algorithm of DE previously exposed. We succeed in obtaining a good left-follower changing the values of three of the connections of the right-follower. The networks obtained were tested placing the robot in 10 different positions in the maze and observing the robot behavior while driving through three consecutive crossroads. The positions were chosen in such a way that the robot starts its test in the middle of a corridor and oriented with its lateral part parallel to the wall. In this conditions the right and the left-follower were both successful in all the trials, showing the appropriate behavior in each of the corridors.

The Interpreter *IM*. The second step is the construction of the fixed-weight interepreter IM capable of switching between the two right and left-follower tasks in response to suitable input codes and without changing its structure. The starting point are the matrices obtained in the first step for the right-follower and the left-follower:

$$\mathbf{W}_{RF} = \begin{pmatrix} 178.2 & -457.4 & -335.1 \\ -365.3 & -73.4 & 66.3 \\ 883 & 422.2 & -266.1 \end{pmatrix} \qquad \mathbf{WE}_{RF} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -19 & 0 \\ 0 & 0 & 769.5 \end{pmatrix}$$

$$\mathbf{W}_{LF} = \begin{pmatrix} 178.2 & -457.4 & -335.1 \\ -365.3 & -73.4 & \underline{-4} \\ 883 & 422.2 & -266.1 \end{pmatrix} \qquad \mathbf{WE}_{LF} = \begin{pmatrix} 0 & 0 & 0 \\ \underline{18} & \underline{0} & 0 \\ 0 & 0 & 769.5 \end{pmatrix}$$

where the values of the connections changed are underlined for the left-follower.

The network that achieves the right-follower and the one that achieves the left-follower are CTRNNs both composed of three neurons. Accordingly the interpreter IM will consist of three neurons plus the "fast" neurons of the *mul* subnetworks, resulting in a network of 9 neurons. As we explained, the right-follower network and the left-follower network differ in the values of three connections. So we realized an PNN network substituting those three connections with three multiplicative networks. The network so obtained, as it is shown in figure 3, has three standard inputs and three auxiliary inputs.

MBN: training and test. In this section we describe the training and the test phase of the Multiple Behavior Network (MBN) system. As above described, the system is composed of two modules hierachically organized (see figure 2): the lower one, IM, that is the fixed-weight CTRNN described in the previous

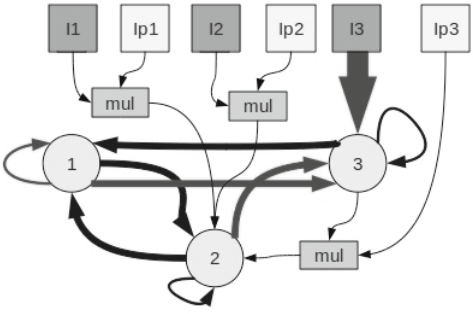

Fig. 3. The interpreter IM. I_1, I_2 and I_3 are the standard inputs coming from sonars; I_{p_1}, I_{p_2} and I_{p_3} are the auxiliary inputs

section, and a higher level PM, that sends suitable codes/programs to the interpreter. During the MBN training phase, the higher level of MBN, PM, learns to associate two different signals (one-color tile/two-colors tile), forwarded by the visual system, with two different codes (two pairs of three numbers), which encode two distinct CTRNNs capable of exhibiting two different behaviors: the right and the left-follower, respectively. Of course the codes (weights) in the matices of previous section would achieve this purpose, however in this section we let PM learn them. Note that in this case the codes are not necessarely unique, so different trainings let us find different codes, possibly different from the ones learned in the previous section. The training of the overall MBN system was achieved by the DE algorithm, as follows. We prepared 18 MBN individuals differing in the weights of the higher level PM. These weights were randomly initialized into $[-10, 10]$. Each network was evaluated on the basis of the behavior of the controlled robot. Each network was evaluated in the 7 crossroads maze used before. In two trials the robot had to travel for 20 seconds. During the first trial one-color tiles were placed in the environment and the robot was evaluated with the fitness function used to train the right-follower, F_R. During the second trial two-colors titles were placed in the maze and the robot was evaluated with the fitness function F_L. The product of these two values were considered as the robot reward. It is important to stress that in this way the DE rewards the robot that chooses the best programs of right and left-follower, none of the two motor primitives is defined before the training is on. We can say that the DE chooses at the same time the best complex behavior and the best primitives.

In order to test the obtained behaviors, similarly as described in section 3.1, the robot is placed in 10 different positions in the maze and its behavior is observed while the robot drives through three consecutive crossroads. Each test maze is prepared with different tiles (one or two-color) as shown in figure 4. During the test phase the trained MBN was always capable of changing its behavior in a left or a right-follower depending on the last type of tile visually recognized. The successful operation of the system in its simulated environment is documented at `http://people.na.infn.it/~donnarumma/files/Prog_CTRNN.avi`

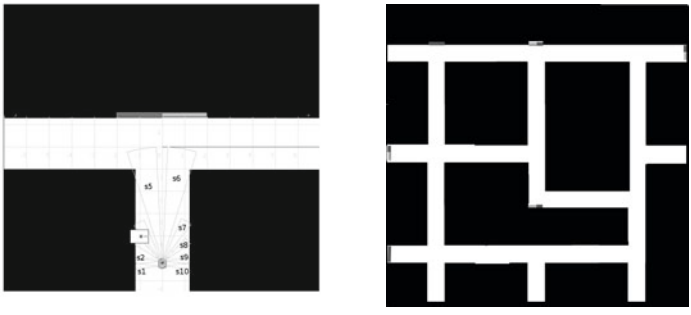

Fig. 4. On the right the robot sensor position is shown. On the left a picture of the maze in which one and two-color tiles are placed.

4 Discussion

We have endowed with programmability, originally associated with algorithmic computability, a specific variety of dynamical neural network: Programmable Neural Network (PNN), which consists of particular fixed-weight CTRNNs.

We accomplished a robotic application which suggests a way of building ANNs able to model more complex multiple behaviors. We focused in particular on the possibility to build networks that quickly change behaviors, showing how important programming is, giving the possibility to modify the neural network behavior by changing a code instead of modifying its structure. This capability is implemented by the use of a two layer architecture, MBN, in which the upper level learns to associate an environment input to the code of an appropriate CTRNN, while the lower level is able to interpret the code provided by the upper level and virtually realizes the coded network. The training of MBN presents two interesting aspects. Firstly, during the training phase, we do not restrict the system to the learning of sequences of predetermined motor primitives. On the contrary, the network architecture constrains the evolutionary algorithm to tune the appropriate motor primitives, setting the behavior of the lower level network IM in such a way as to obtain the best performance of the robot in the maze. Moreover by this kind of architecture we have a network that learns to realize different behaviors because the higher level network PM learns to provide the suitable codes to the *interpreter* IM. Both these aspects could be improved, building an interpreter network in which all the connections are programmable. In this process some difficulties must be addressed. In fact, in order to perform the experiments of section 3, we constructed multiplicative networks with an error of 5%, and for them we chose a time constant one order of magnitude less than the time constant of the neurons of the original network. Of course this heuristic choice worked well for our tests in the simulated environment, because we experimentally met the approximation hypothesis in section 2 on the multiplicative networks. This could be not the case for a real environment, in which even noise introduced by actuators and sensors should be compensated. More in general, as the interpreter network and the emulated CTRNN are both

non-linear systems, the bigger the number of programmable connections is, the higher the possibility for the two networks to show different dynamics is. Thus further studies are needed in order to be able to perform a robust PNN capable of interpreting a vast number of codes/programs.

Acknowledgments

We wish to thank Prof. Giuseppe Trautteur for his many helpful comments and discussions about programmability in brain, biological systems and nature.

References

1. Anderson, M.L.: Neural re-use as a fundamental organizational principle of the brain - target article. Behavioral and Brain Sciences 33(04) (2010)
2. Beer, R.D.: On the dynamics of small continuous-time recurrent neural networks. Adaptive Behavior 3(4), 469–509 (1995)
3. Blynel, J., Floreano, D.: Exploring the T-Maze: Evolving Learning-Like Robot Behaviors using CTRNNs. In: 2nd European Workshop on Evolutionary Robotics (2003)
4. De Falco, I., Cioppa, A.D., Donnarumma, F., Maisto, D., Prevete, R., Tarantino, E.: CTRNN parameter learning using differential evolution. In: ECAI 2008, vol. 178, pp. 783–784 (July 2008)
5. Donnarumma, F.: A Model for Programmability and Virtuality in Dynamical Neural Networks. PhD thesis, Università di Napoli Federico II (2010)
6. Donnarumma, F., Prevete, R., Trautteur, G.: Virtuality in neural dynamical systems. In: International Conference on Morphological Computation, Venice, Italy (2007)
7. Donnarumma, F., Prevete, R., Trautteur, G.: How and over what timescales does neural reuse actually occur? Behavioral and Brain Sciences 33(04), 272–273 (2010)
8. Floreano, D., Mondada, F.: Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In: Proceedings of the Conference on Simulation of Adaptive Behavior, pp. 421–430. MIT Press, Cambridge (1994)
9. Garzillo, C., Trautteur, G.: Computational virtuality in biological systems. Theoretical Computer Science 410(4-5), 323–331 (2009); Computational Paradigms from Nature
10. Hopfield, J.J., Tank, D.W.: Computing with neural circuits: A model. Science 233, 625–633 (1986)
11. Paine, R.W., Tani, J.: Evolved motor primitives and sequences in a hierarchical recurrent neural network. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 603–614. Springer, Heidelberg (2004)
12. Pearlmutter, B.A.: Gradient calculations for dynamic recurrent neural networks: A survey. IEEE Transactions on Neural Networks 6(5), 1212–1228 (1995)
13. Riesenhuber, M., Poggio, T.: Models of object recognition. Nature Neuroscience 3, 1199–1204 (2000)
14. Trautteur, G., Tamburrini, G.: A note on discreteness and virtuality in analog computing. Theoretical Computer Science 371, 106–114 (2007)
15. Yamauchi, B.M., Beer, R.D.: Sequential behavior and learning in evolved dynamical neural networks. Adaptive Behavior 2(3), 219–246 (1994)